Inaugural-Dissertation

zur Erlangung der Doktorwürde der

Naturwissenschaftlich-Mathematischen
Gesamtfakultät

der

Ruprecht-Karls-Universität
Heidelberg

vorgelegt von
Master of Science Informatik

Bartosz Ryszard Bogacz

geboren in Gdynia, Polen

Tag der mündlichen Prüfung: _____

Analyzing Handwritten and Transcribed Symbols in Disparate Corpora

Bartosz Bogacz

October 23, 2017

Advisor: Dr. Hubert Mara
Co-advisor: Prof. Dr. Micheal Gertz

Abstract

Cuneiform tablets appertain to the oldest textual artifacts used for more than three millennia and are comparable in amount and relevance to texts written in Latin or ancient Greek. These tablets are typically found in the Middle East and were written by imprinting wedge-shaped impressions into wet clay. Motivated by the increased demand for computerized analysis of documents within the Digital Humanities, we develop the foundation for quantitative processing of cuneiform script.

Using a 3D-Scanner to acquire a cuneiform tablet or manually creating line tracings are two completely different representations of the same type of text source. Each representation is typically processed with its own tool-set and the textual analysis is therefore limited to a certain type of digital representation. To homogenize these data source a unifying minimal wedge feature description is introduced. It is extracted by pattern matching and subsequent conflict resolution as cuneiform is written densely with highly overlapping wedges.

Similarity metrics for cuneiform signs based on distinct assumptions are presented. (i) An implicit model represents cuneiform signs using undirected mathematical graphs and measures the similarity of signs with graph kernels. (ii) An explicit model approaches the problem of recognition by an optimal assignment between the wedge configurations of two signs. Further, methods for spotting cuneiform script are developed, combining the feature descriptors for cuneiform wedges with prior work on segmentation-free word spotting using part-structured models. The ink-ball model is adapted by treating wedge feature descriptors as individual parts. The similarity metrics and the adapted spotting model are both evaluated on a real-world dataset outperforming the state-of-the-art in cuneiform sign similarity and spotting.

To prove the applicability of these methods for computational cuneiform analysis, a novel approach is presented for mining frequent constellations of wedges resulting in spatial n-grams. Furthermore, a method for automatized transliteration of tablets is evaluated by employing structured and sequential learning on a dataset of parallel sentences. Finally, the conclusion outlines how the presented methods enable the development of new tools and computational analyses, which are objective and reproducible, for quantitative processing of cuneiform script.

Zusammenfassung

Keilschrifttafeln gehören zu den ältesten Textzeugen, die im Umfang und Bedeutung mit den Texten in lateinischer und alt-griechischer Sprache vergleichbar sind, da diese Tafeln aus dem gesamten Alten Orient über beinahe viertausend Jahre in Verwendung waren. In die aus Ton geformten Tafeln wurden mit einem eckigen Stylus Zeichen als keilförmige Abdrücke eingedrückt. Sie erfordern zur Dokumentation und Analyse, anders als die in Archiven üblichen Flachwaren, neue Methoden der Informatik.

Keilschrifttafeln werden mit verschiedensten Methoden in 2D und 3D digitalisiert und in untereinander nicht kompatible Formate übertragen. Jede dieser Repräsentationen erfordert ein eigenes Tool-Set zur Analyse. Zur Homogenisierung der Daten wird eine minimale und einheitliche Beschreibung von Keilabdrücken mit Merkmalsvektoren eingeführt. Da sich die einzelenen Keile stark überlagern, wird bei der Extraktion eine Untermenge von Vektoren gewählt, die Keilmodelle optimal den jeweiligen Keilabdrücken zuordnet.

Ähnlichkeitsmetriken werden basierend auf zwei Modellen präsentiert: (i) Ein implizites Modell stellt Keilschrift als ungerichtete Graphen dar und macht sich Graphkernel zur Berechnung der Ähnlichkeit zunutze. (ii) Ein explizites Modell repräsentiert Keile als Merkmalsvektoren und definiert die Ähnlichkeit zwischen Zeichen unter Verwendung von einer optimalen Zuordnung von Keilkonfigurationen. Darauf aufbauend werden Methoden zur Suche von Keilschriftszeichen, ohne Notwendigkeit zur a-priori Segmentierung, auf Basis von teilstrukturierten Modellen entwickelt. Die Merkmalsvektoren bilden hierbei deren grundlegende Strukturelemente. Der Vergleich mit neusten Ansätzen im Document Retrieval zeigt eine höhere Genauigkeit der hier entwickelten Methoden bei gleichzeitig mehr gefundenen Zeichen.

Die Anwendbarkeit der vorgestellten Verfahren für die computergestützte Analyse von Keilschrift, wird durch die Entwicklung von darauf aufbauenden Methoden belegt: Die homogene Beschreibung von Keilen ermöglicht eine Methode zur Assoziationsanalyse von räumlichen N-Grammen. Eine andere Anwendung ist die automatisierte Transliteration von Keilschrift, durch sequentielles Wortmodell auf einer Datenbasis von parallelen Sätzen. Abschließend zeigt der Ausblick, wie diese Arbeit die Voraussetzung für die Entwicklung neuer Werkzeuge und quantitativer Analysemethoden zur Erschließung von Keilschrifttexten schafft, die dazu objektiv und reproduzierbar sind.

Danksagung

An dieser Stelle möchte ich mich bei all den Menschen bedanken, die mich bei dieser Arbeit unterstützt haben, mir mit gutem Rat den Weg aufgezeigt und mir beiseite gestanden haben.

Zu aller Erst gilt mein Dank meinem Doktorvater Dr. Hubert Mara dessen Rückhalt ich mir immer sicher sein konnte. Stets nahm er sich Zeit mit mir über Ideen zu diskutieren, begegnete mir mit Großzügigkeit und Interesse. Unter seiner Leitung konnte ich mich als Wissenschaftler entwickeln. Dr. Mara stellte mich wichtigen Kooperationspartnern vor und erschloss mir immer wieder Orte und Möglichkeiten das Beste aus meiner Arbeit zu schöpfen.

Besonders danken möchte ich Prof. Micheal Gertz. Er widmete mir Zeit und Geduld, mit mir über meine Fortschritte zu sprechen und mir Alternativen aufzuzeigen. Seine fachliche Erfahrung, die er mit mir teilte, zeigte mir die Vorgänge des wisschenschaftlichen Arbeitens.

Ich möchte mich sehr bei Prof. Howe für seine Gastfreundschaft und Zeit bedanken, die er mir während meines Aufenthaltes in seiner Arbeitsgruppe bot. Unsere Kooperation ist ein wichtiger Bestandteil meiner Arbeit und der gemeinsame Austausch an Ideen, hat mir neue Möglichkeiten eröffnet.

Ein besonderer Dank gilt Prof. Stefan Maul, der mir mit viel Geduld eine Thematik erklärt hat, die mir bisher unbekannt war. Seine Feedback zu meinen Ansätzen bilden einen wichtigen Grundstein dieser Arbeit.

Auch gehört mein Dank Dr. Micheal Winckler und der Graduiertenschule HGS MathComp, ohne deren Unterstützung und Angebote zum wissenschaftlichen Austausch diese Arbeit nicht möglich gewesen wäre.

Mein besonderer Dank von Herzen gilt Dr. Julia Portl. Sie hat mir eine neue Welt erschlossen, widmete mir ihre Zeit und Aufmerksamkeit. Ihre fachlichen Kenntnisse und stetige Hilfe im grafischen Design und visueller Sprache bringen meinen Ideen Klarheit und Verständlichkeit.

Außerdem gehört ein großer Dank der Verwaltung, den Sekretariaten und den zentralen Diensten. Oktavia Klassen, Christina Pietsch, Anne Paulski, Maria Rupprecht, Markus Ridinger, Sarah Steinbach und Ria Lynott haben mir stets geholfen und sich für mich eingesetzt und so manches Unmögliche möglich gemacht.

Meine Mentoren Dr. Susanne Krömker und Dr. Bastian Rieck standen mir persönlich und fachlich zur Seite. Sie haben mich motiviert und über Ideen und Lösungen gesprochen, die mir oft weitergeholfen haben. Ich danke meinen Kollegen Dr. Andreas Beyer, Prof. Filip Sadlo, Boyan Zheng, die für mich immer ein Ohr offen hatten.

Meine Eltern Elżbieta und Ryszard, mein Bruder Daniel waren mir stets eine moralische und seeliche Hilfe. Sie ermutigten mich und standen mir bei jeglichen Schwierigkeiten und Nöten beiseite. Auch haben sie mich auf unzählige Weisen unterstützt und sich um mich gesorgt.

Contents

1 Digitalizing the Analysis of Ancient Handwriting

Motivated by the increasing digitization and demand for computational text analysis in Assyriology, we introduce and develop methods for the fundamental constituents of written cuneiform script enabling computational modeling and processing of cuneiform characters.

For more than three millenia in the ancient Middle East, scribes wrote documents using cuneiform script [Sod94]. Cuneiform tablets belong to the oldest textual artifacts. The number of known tablets is assumed to be in the hundreds of thousands, which is constantly increasing as new tablets are excavated by archaeologists on a regular basis. By roughly estimating the number of words on those tablets, we can assume that the total amount of text in cuneiform script is comparable to those in Latin or Ancient Greek. Since those tablets were used in all of the ancient Near East for over three thousand years [Sod94], interesting research questions can be answered regarding the development of religion, politics [Mau17], science, trade and climate change [Kan+13]. These tablets were formed by clay and written on by impressing a rectangular stylus [Bor04] into a wet clay tablet. The result is a wedge-shaped impression in the clay tablet. The word cuneiform derives from the Latin word "cuneus" wedge and "forma" shaped. Constellations of wedge-shaped impression form thousands of cuneiform words and are documented in the work of Borger [Bor04] and the work of Soden [Sod94]. As clay was always cheaply and easily available, those capable of writing could produce a multitude of documents. Therefore, the content of cuneiform tablets ranges from mundane shopping lists to treaties between empires.

There is an increasing demand in the Digital Humanities domain for handwriting recognition, i.e. machine reading of handwritten script, focusing on historic documents. Even the recognition of ancient characters sharing shapes with their modern counterparts, e.g. ancient Chinese Sutra [MHK09], is a challenging task. For digitally processing cuneiform script, there exist only few

related approaches. Fisseler et al. [Fis+14] uses geometric features of cuneiform tablets acquired with a 3D-scanner [Mar+10] and Sperl [Spe81] constructs a labeling for wedge types. With the increased availability of high-resolution 3D-measurement technology [MMS08] it became possible to rapidly acquire and process cuneiform tablets in 3D [HW11; Fis+14; Chn+14], which allows to extract [MK13] outlines of cuneiform characters, i.e. groups of wedges.

However, most of the data does only exist in printed books with manual drawings, which are only available as raster images. Furthermore, modern Assyriologists manually create digital line drawings of tablets with vector graphics editors, e.g. Inkscape and the proprietary Adobe Illustrator, using the underlying photographs. In this case we get an eXtensible Markup Language (XML) based Scalable Vector Graphics (SVG), which can have internal variations depending on the manual drawing technique.

Manual transliteration of cuneiform is a challenging task. Cuneiform script has no whitespace between words and contains upwards of 20 known grammatical cases. Words are conjugated by adding a prefix, infix or postfix. Additionally, the same cuneiform characters may have different readings that can only be identified by knowing the current grammatical case. Assyriologists reference other tablets containing the same character being translated to understand the context in which it is used. This work is done manually by sifting through tracings.

Automating this process by means of a wedge constellation spotting tool, provides experts with a significantly broader base of references to create more accurate and less time consuming transliterations and translations. Currently, text analysis of cuneiform script is done only on the Latin transliterations and translations which are incomplete and influenced by the knowledge and experience of the expert that created them. Statistical analysis directly on the wedge patterns is free from interpretation bias and has a significantly greater data basis to work from.

1.1 Challenges and Objectives

The development of computational tools for cuneiform analysis presents many opportunities. An efficient and accurate sign spotting enables cross-referencing and statistical analyzes that are infeasible to perform manually. Yet, cuneiform

script has since resisted efforts to computational processing on basis of its basic constituents, i.e. wedge-shaped impressions and signs. We identify the following challenges which have to be overcome:

1. Original cuneiform tablets and tracings thereof are available in many heterogeneous and incompatible formats. The usage of different tools for every representation is not a workable solution.

2. Cuneiform script has no whitespace, highly complex and flexible grammar and words with multiple conflicting interpretations. This precludes word segmentation without actual understanding of the underlaying language.

3. The very dense writing and high variability of wedge-shaped impressions makes basic approaches, like template matching, too limited for extracting wedges. As a consequence, a closed enumerable discrete representation, such as letters in Latin, is insufficient. Wedges require a free-form continuous and open representation.

4. Even though there are thousands of cuneiform tablets present in museums, only very few are accessible to us and available digitally for processing. Therefore, methods requiring large datasets for learning are ruled out.

Given the current state of research of digitalized cuneiform, which is limited to visualization [MK13] or analysis of manually collected facts as is the case for the Cuneiform Commentaries Project (CPP), we set out for the following objectives:

1. Develop methods and tools which enable the processing and manipulation of heterogeneous sources of cuneiform in a unified fashion.

2. Research cuneiform sign search facilities that circumvent the need for segmentation. Since neither writing direction or tablet layout are fixed in cuneiform, a cuneiform sign search independent of this assumptions is advantageous.

3. Develop feature vector representations and distance functions for wedges and signs enabling the usage of common machine learning algorithms. The application of common transformations, e.g. space embeddings and clustering, and learning methods, e.g. classification and pattern-mining,

opens a plethora of opportunities for novel research in statistical language analysis of cuneiform.

4. Apply the developed tools with methods in computer linguistics on cuneiform. Gain new insights into quantitative properties and patterns in cuneiform script which are as of yet not possible.

1.2 Structure of the Thesis

We organize this work into seven chapters and distinct steps of abstraction. These are visualized in Figure 1.1. Chapter 1 provides an introduction into the topic of historical handwriting recognition and motivates the need for novel general tools for computational analysis. Chapter 2 provides background on cuneiform script, the methods and results of acquiring cuneiform tablets, and their contents.

In Chapter 3 we define mathematical models for cuneiform script and extract these from tracings of tablets. We introduce a unified semantic description of cuneiform. We published our findings in [BMM15]. Chapter 4 introduces the concept of measurable equality and inequality for cuneiform script and provides an evaluation of different similarity schemes. Work on these methods has been published in [BGM15b] and [BGM15a]. In Chapter 5 we extend approaches for spotting Latin words with our novel methods to arrive at an algorithm for spotting cuneiform signs. We evaluated competing approaches and published our results in [BHM16].

Chapter 6 proves the applicability of our contributions by performing computational analysis of cuneiform script. We publish these analyses in [BM16] and [BKM17]. Finally, in Chapter 7 we conclude with a highlight of key achievements and provide a detailed overview of our contributions. An outlook presents our impact on computational research in Assyriology and poses novel research questions which are first enabled by our research.

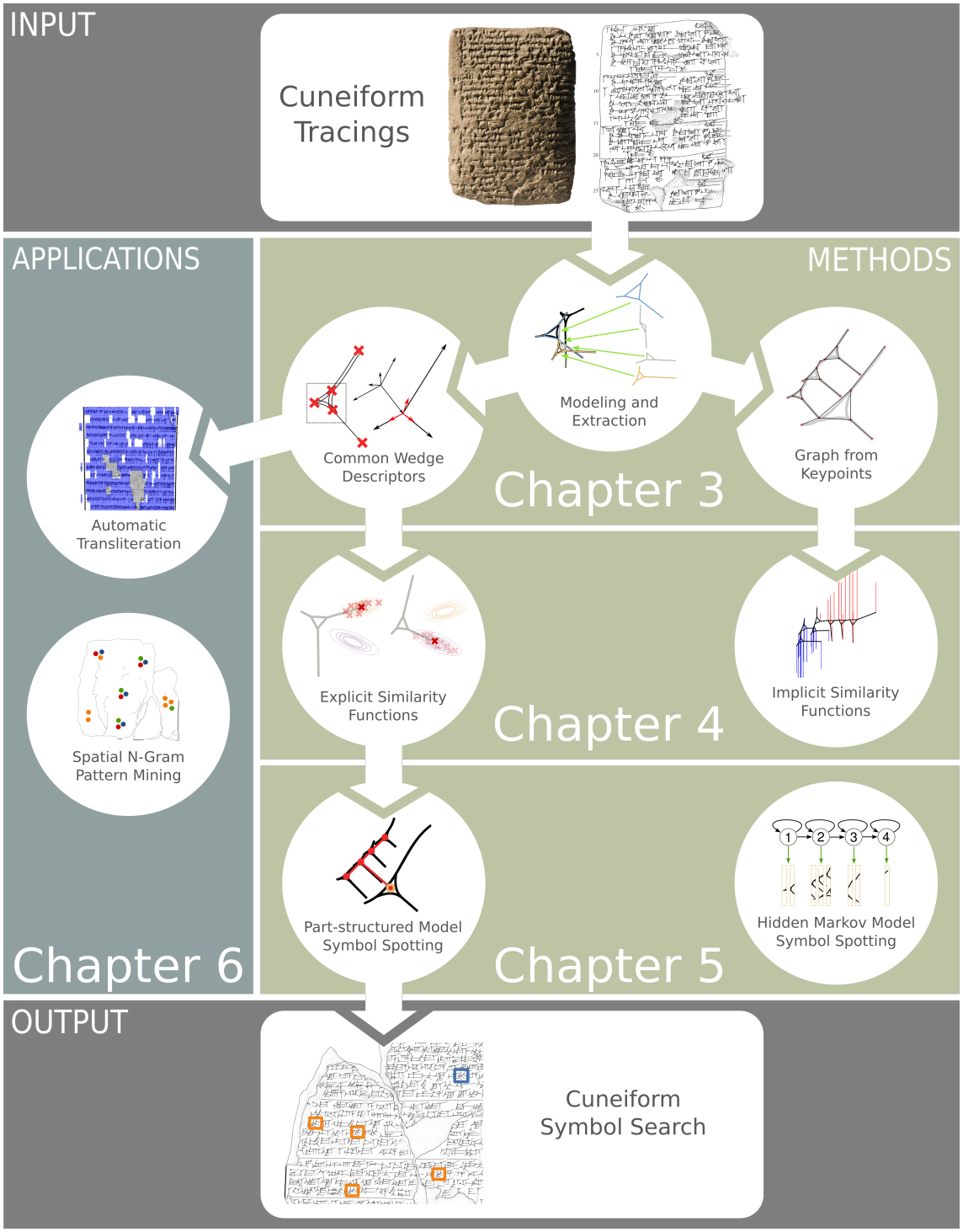


Figure 1.1: We visualize the structure of this thesis as distinct steps of increasing abstraction of cuneiform script and analyses possible hereby. Steps connected by arrows indicate a dependency on the results of the previous step.

2 Background on Cuneiform Script and Cuneiform Research

Cuneiform tablets are one of the oldest textual artifacts comparable in extent to texts written in Latin or ancient Greek. They have seen use in the ancient orient from 3000 B.C to 1000 A.D. In this timeframe many different languages based their writing on the cuneiform script and the writing itself evolved from a pictographical to a syllabic script. These tablets were formed from clay and written on by impressing a rectangular stylus. The result is a wedge shaped impression in the clay tablet. More than half a million cuneiform tablets have been excavated from regions including todays Iran, Iraq, Syria and Turkey.

2.1 Cuneiform Signs and Writing System

Cuneiform signs are written in left-to-right or top-down lines using a two dimensional arrangement of wedge-shaped impressions. Wedges have an inner approximately triangular form which we denote as the wedge-head and rifts extending from the triangle corners which we denote as the wedge-arms. Figure 2.3 on page 21 illustratively highlights these portions of a wedge-shaped impression and Figure 2.2 shows the difference of appearances of wedge-shaped impressions between original tablets and their tracings. Assyriologists differentiate between five types of wedges [Cam14]. Figure 2.1 illustrates the different types of wedges.

Standing wedges with a vertical stem and the wedge-head on top.

Prone wedges with a horizontal stem and the wedge-head on the left.

Upward askew wedges with diagonal stems and wedge-heads right of and below the stem.

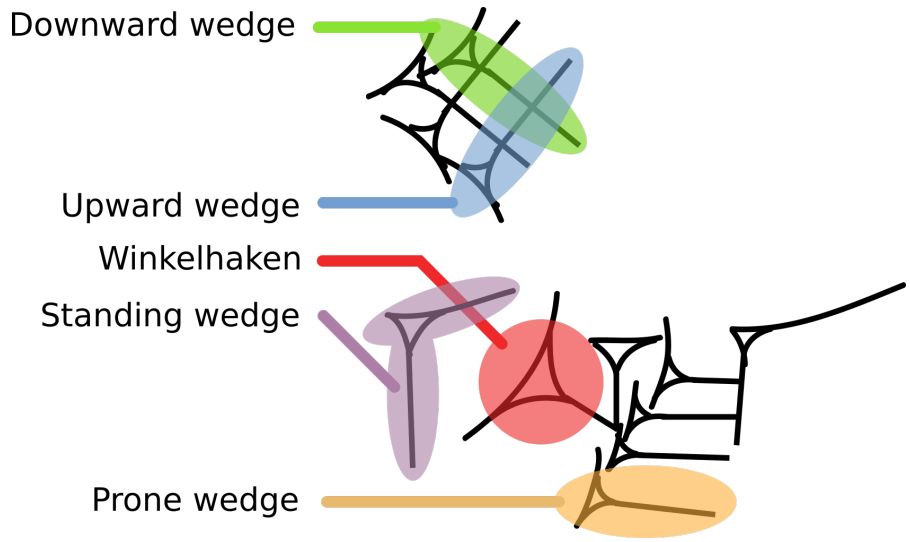


Figure 2.1: The different types of wedges annotated in two cuneiform signs.

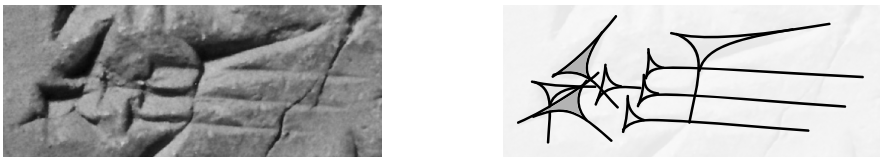
Downward askew wedges with diagonal stems and wedge-heads left of and above the stem.

Winkelhaken that is being impressed by stabbing the rectangular stylus without any additional movement. Winkelhaken are denoted with filled wedge-heads in some tracings as shown in Figure 2.2. They are identified by their wedge-head only as they are not traced with any wedge-arms.

In early cuneiform script written pictographic wedges have also been used decoratively and in ways not clearly differentiated by the five recognized types. Some amount of creative interpretation is not avoidable in transliteration and translation of cuneiform.

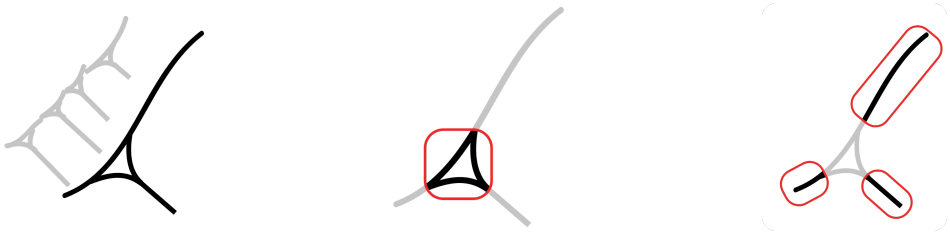
Cuneiform is polyvalent, i.e. written cuneiform signs carry different meanings, and the transliterating scholar has to decide its reading. A meaningful collection of wedge-shaped impressions forms cuneiform sign, whereas a meaningful but not unique collection of cuneiform signs forms a word in cuneiform script. The difficulty stems form the fact that the same succession of cuneiform signs can be read as different words, especially if the language, e.g. Assyrian or Sumerian, is not yet determined. Thus, the same sequence of cuneiform signs split differently, yielding different subsequences of cuneiform signs, yields differently read words.

Transliteration of cuneiform is therefore related to the exact set cover problem [Kar72]. A sentence of cuneiform signs is the universe of elements and words are sets whose union equals the universe. A correct transliteration is



(a) (b)

Figure 2.2: Excerpt of wedge-shaped impression on a a) cuneiform tablet and b) its tracing.



(a) (b) (c)

Figure 2.3: Parts of wedge-shaped impression: a) The complete wedge, b) the wedge-head and c) the wedge-arms.

therefore a collection of sets (words) that covers the universe (the sentence) so that each element (cuneiform sign) is covered by exactly one set (word).

In this work we concern ourselves only with wedge-shaped impressions and wedge constellations without regard to their meaning. Such an approach allows us to develop methods for analysis and search that do not require a language model or a word model. These languages are neither fully understood yet nor static through their lifetime, writing developed from pictographic to syllabic and differs between languages. Hence, our methods work on the visible representation of written cuneiform.

2.2 Digital Databases & Projects

The digitization of cuneiform tablets and the development of respective databases provides us with open access corpora of photographs, line tracings, transliterations and translations. Major examples are the Cuneiform Digital Library Initiative (CDLI, `http://cdli.ucla.edu/`) and the Open Richly Annotated Cuneiform Corpus (ORACC, `http://oracc.museum.upenn.edu/`). For many of its tablets the Cuneiform Commentaries Project (CPP) from Yale University provides line tracings with annotated transliterations and

1. *ana-ku dasar-lú-ḫi šá ina ra-ma-ni-šú* DÙu ˹*ana*˺*-[ku]*
2. *ma-a ina* UGU *ú-lu-lu an-šár qa-*˹*bi*˺
3. *ana-ku dasar-lú-ḫi šá a-*˹*šar*˺ *šil-la-te! la i-qab-bu-u* ˹*ana*˺*-[ku]*
4. *ina* ŠÀ *kam-me šá* dMES DÙ-*šú* UM.ME.A *la* ˹*i*˺*-sal-lit*
5. *la i-*˹*tak*˺*-kip ma-a* dAMAR.UTU *ka-mu-u* AD-MEŠ-šú ŠEŠ-MEŠ-˹*šú*˺
6. *ina lìb-bi* ÉN *dup-pir lem-nu*

Figure 2.4: Excerpt from a cuneiform tablet (CCP 2.2.1.A.b) (top) and its transliteration (bottom). Areas with ink points denote tablet damage. Transliteration [FJF17] contains both radicals (lowercase) and fully identified letters (uppercase).

translations. However, the library cannot be searched using cuneiform signs as queries. Only the transliterations can be searched using Latin query words. Figure 2.4 illustrates a retro-digitized cuneiform tablet tracing and the associated transliteration as available in the CCP.

Cuneiform tablets are being acquired from different sources and require different methods for digitalization. In the following sections we describe the formats of three sources and the methods necessary to digitize the information on the tablets.

2.3 3D-Scanned Cuneiform Tablets

Cuneiform tablets available as originals are acquired using a stereo and structured light 3D-scanner [SM92] as shown in Figure 2.5. An alternating pattern of structured light is projected on the cuneiform tablet and is deformed by its surface shape. These deformations are measured by two cameras of the scanner. Then, the software associated with the 3D-scanner calculates a mesh from the set of captured images. The result is a mesh of triangulated measuring

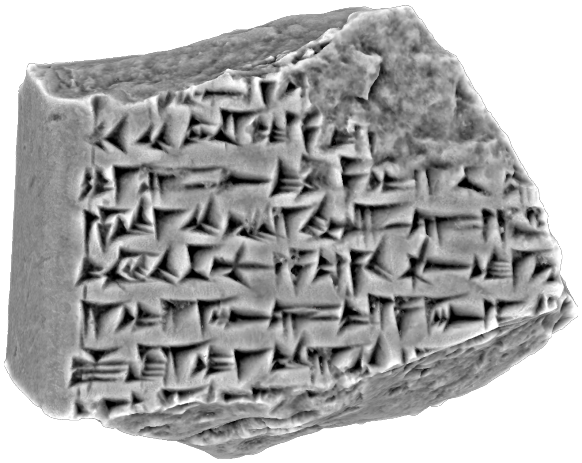


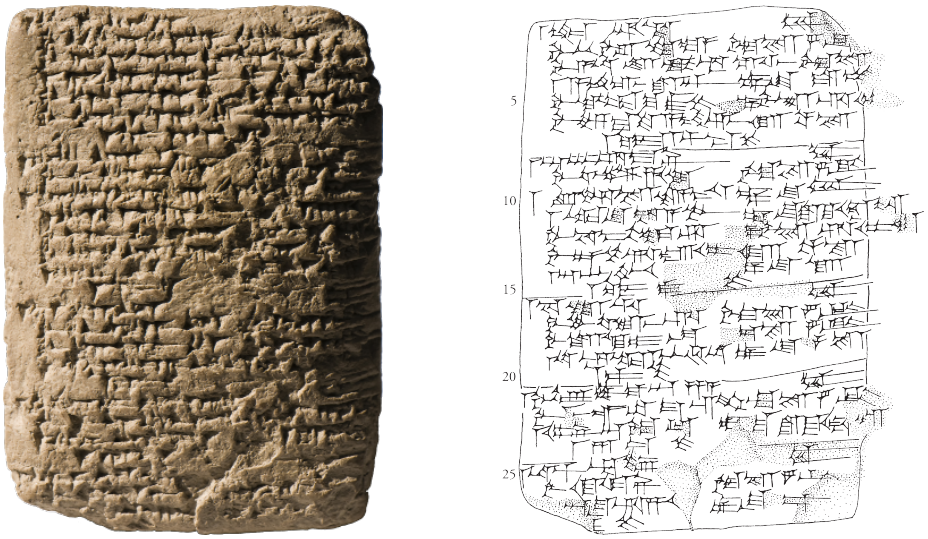
Figure 2.5: *Königsinschrift* found 1975 in Assur, Iraq [Mar12]. Rasterized 3D-scan of a cuneiform tablet shaded with the MSII [MK13] feature descriptor.

points, the vertices of the mesh, capable of resolving features up to 2mm. The distance between the vertices is $\frac{1}{100}$mm.

2.4 Retro-Digitized Cuneiform Tablets

Typically tracings of cuneiform tablets are done manually with paper and ink. An Assyriologist traces the wedge-shaped impressions, the shape and the damage of an original cuneiform tablet or the photograph of the original tablet. Flat-bed scans of such tracings and the photographs of the original cuneiform tablets are freely available to download from the CDLI.

In tracings of cuneiform tablets, different conventions are used to describe the same concepts. Most cuneiform tablets are moderately damaged or broken into parts, for example. Damage and break lines are often drawn using either crosshatching or indicated with dots in differing densities. The shape and damage of the original cuneiform tablet is indicated by contour lines outlining areas of broken off parts. Wedge-shaped impressions are typically drawn as triangles with arms. The wedge-head can either be a filled triangle or three slightly curved pairwise intersecting strokes. Figure 2.6 shows an original cuneiform tablet and a scan of a manually created tracing.



(a) (b)

Figure 2.6: Cuneiform tablet No. TCH92, G127 [Jak09]: (a) Photograph and (b) its retro-digitized tracing.

2.5 Born-Digital Cuneiform Tracings

A more modern process used by Assyriologists are born-digital tracings of cuneiform tablets. Comparable to tracings drawn on paper, digital tracings are drawn manually on graphics tablet using a vector graphics editor, such as Inkscape (`https://inkscape.org/de/`). The shape, damage and wedges are drawn using the same conventions as in retro-digitized tracings of cuneiform tablets. Figure 2.7 shows a born-digital tracing of a cuneiform tablet.

A born-digital tracings of a cuneiform tablet has two very important advantages over a retro-digitized tracing for automated analysis. First, each stroke drawn by an Assyriologist, no matter how short, small or occluded, is directly enumerable from underlaying source code as shown in Figure 2.8. This allows for pattern matching and the restoration of the underlying data. Secondly, the strokes used to indicate wedges are typically on a separate layer than strokes indicating tablet shape or tablet damage. Extraction can be performed without the need of first segmenting the foreground text from the damaged tablet. Figure 2.9 shows a born-digital tracing decomposed into three layers.

Although there is only a limited set of wedge types, wedges are traced by individual strokes drawn on a graphics tablet instead of being drawn as templates. Hence, tracings of cuneiform tablets are a handwriting of a handwriting, the cuneiform script on tracings is twice perturbed. First, by the original

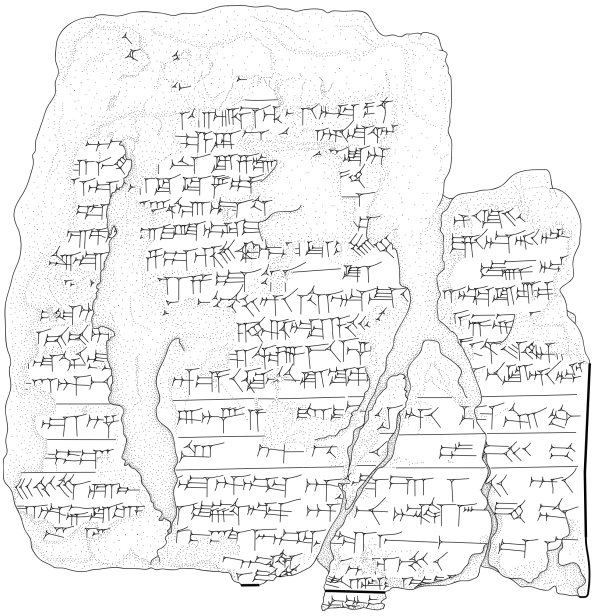
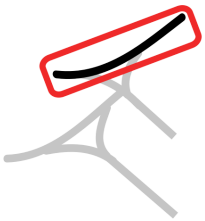


Figure 2.7: A born-digital tracing of a cuneiform tablet created using a vector graphics editor and a graphics tablet.



```
<g
   id="g9803"
   transform="translate(57.498721,273.05622)">
  <g
     id="g9805">
    <path
       inkscape:connector-curvature="0"
       id="path9809"
       d="m 310.834,284.028 c 1.488,0.175 2.635,0.851 3.656,
           1.927 0.223,0.234 0.575,-0.12 0.354,-0.354 -1.122,
          -1.182 -2.381,-1.883 -4.01,-2.073 -0.32,-0.037 -0.316,
           0.463 0,0.5 l 0,0 z" />
  </g>
</g>
```

Figure 2.8: The stroke of a cuneiform sign and its respective description as a spline in the SVG file format.

(a) (b) (c)

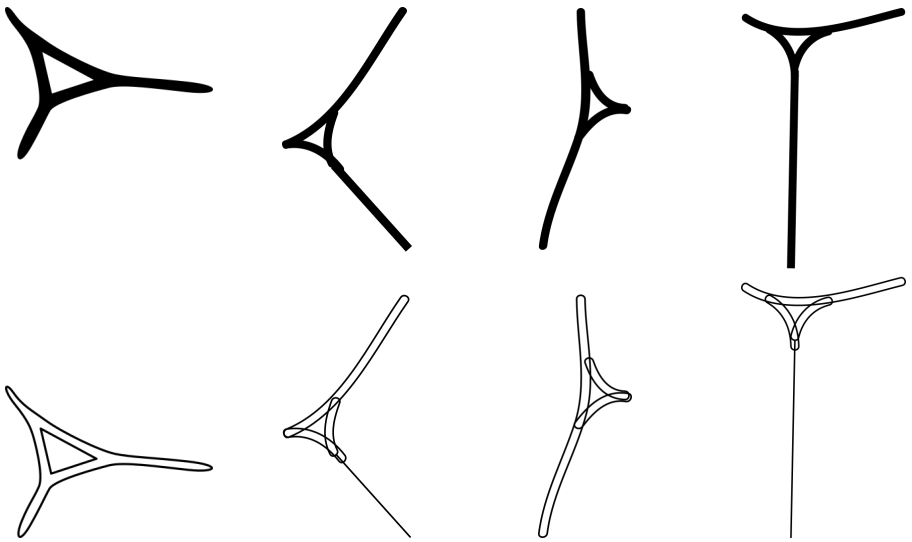
Figure 2.9: a) The script layer of born-digital tracing of a cuneiform tablet containing the tracing of cuneiform script. b) The damage and shading layer detailing geometric features of the cuneiform tablet. c) The tablet shape layer outlining the tablet shape and hard edges. As small errors creep in during the tracing process, layers are not always perfectly separated. The script layer a) contains shading and the shading layer b) contains some wedges from the script layer.

author writing in cuneiform and second, by the expert tracing the tablet, each bringing their own idiosyncrasies.

Figure 2.10 illustrates this difference. As a consequence, not only is the visual appearance of each stroke different but also the internal structure between wedges appearing to be similar may be completely different. The count of strokes between the same wedge types and the usage of these strokes, as part of the wedge-head or part of the wedge-arms, differs. Additionally, vector graphics programs may decide to represent strokes using different style elements, e.g. using straight segments as shown in Figure 2.10 b) and c) for strokes that are straight enough, i.e. below an internally defined threshold. We choose to overcome the challenge by supporting all these variations in expressing wedges. To this end, we develop methods and strategies in Chapter 3 to extract the underlaying wedges which are expressed by these strokes.

2.6 Definition of Strokes in Born-Digital Tracings

In SVG files, strokes are represented as closed cubic spline [Boo01] paths, parametrized by four control points. For this work we consider the closed



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

Figure 2.10: Different representations of wedges in born-digital tracings of cuneiform tablets. While a template representation a) would seem natural due to the limited count of wedge types, traced wedges outlines are drawn individually and differently for each new wedge b) c) d).

spline paths as abstract objects supporting intersection and discretization of their boundary. Let $s \in S$ be a stroke s in the set of strokes S. We define a set $I_S \subseteq S^2$ of intersecting stroke pairs $(s_i, s_j) \in I_S$. Then, the center of the intersections is defined for intersecting strokes (s_i, s_j) as follows.

$$\text{center}(s_i, s_j) \in \mathbb{R}^2 \;\; \forall s_i, s_j \in I_S \tag{2.1}$$

Additionally, we define the endpoints of a stroke $s_i \in S$ as two points in $\mathbb{R}^2$ on the boundary of the stroke that are most distant from each other.

$$\text{endpoints}(s_i) \in (\mathbb{R}^2, \mathbb{R}^2) \tag{2.2}$$

The spline of a stroke is planar manifold with a boundary which can be discretized into a set of points. Since the exact definition of the discretization is implementation dependent, in this case on the SVG implementation, we limit ourselves to the following definition.

$$\partial s_i \subset \mathbb{R}^2 \tag{2.3}$$

Let $\|x\|_2$ denote the Euclidean 2-norm. In this work we elide the subscript $\|x\|_2$ since we only use the 2-norm to compute lengths. The definition is as follows.

$$\|x\| := \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \tag{2.4}$$

Given two vectors a, b we denote a vector pointing from a to b with $\vec{ab}$. The distance between a, b is then written as $\|\vec{ab}\|$. The two most distant points of a stroke can be found by a greedy algorithm as shown in Algorithm 1.

In this work we describe the runtime of the presented algorithms asymptotically, i.e. ignoring coefficients and lower-order terms, using big-O notation. We say an algorithm has a runtime complexity of $O(n)$ for an input size n if it completes its execution in linear time, for example.

For a set of strokes s with size $|s| = n$ Algorithm 1 completes n iterations in which vectors a and b increase in distance. Any other operations are of constant time with respect to the input size n. The asymptotic complexity is therefore $O(n)$.

Algorithm 1 Greedily find two most distant points on a closed boundary.

procedure ENDPOINTS(s)
▷ Initialize most distance points as empty at first.
$a \leftarrow \perp$ and $b \leftarrow \perp$
▷ For each point on the boundary s.
for $p \in \partial s$ **do**

▷ Initialize both points with some points
▷ on the boundary s.
if $a = \perp$ **then**
$a \leftarrow p$
else if $b = \perp$ **then**
$b \leftarrow p$
▷ If a connection with the current point p results
▷ in a greater distance than by connecting ab
▷ replace the respective a or b. Thus, increasing
▷ the greatest distance currently found.
else if $\|\vec{ap}\| > \|\vec{ab}\|$ **then**
$b \leftarrow p$
else if $\|\vec{bp}\| > \|\vec{ab}\|$ **then**
$a \leftarrow p$
end if

end for

▷ Return points farthest apart on the boundary of s.
return a, b
end procedure

3 Wedge Detection and Extraction

Born-digital tracings describe the shape and semantics of a cuneiform tablet. Many different types of vector objects, e.g. splines, lines, points, are used for visualization, each with a unique representation. For computational analysis and search of cuneiform signs, we develop a minimal uniform mathematical representation that discards visual noise and keeps semantic information.

Our common uniform representation of cuneiform script is used as an interchange format for cross data format analysis, e.g. between retro-digitized tracings and born-digital tracings. Incomplete or damaged wedge-shaped impressions or annotations and tablet damage erroneously added to script layer are challenging for a straight-forward pattern-matching and extraction of wedge-shaped impressions. In this chapter, we introduce two models of cuneiform script and means of extracting them.

We approach pattern-matching wedges from two different assumptions. First, we assume that wedges and, subsequently, cuneiform signs can be completely described by an undirected graph. Extracting cuneiform signs is then reduced to the subgraph isomorphism problem. Our second assumption uses a higher level description of wedges. We posit that wedges can be described by a model consisting of a triangle with three arms extending from its vertices.

3.1 Challenges and Objectives

In the following we provide a structured overview and emphasize the main aspects of this chapter.

Motivation Computational tools for linguistic and pattern analysis depend upon a uniform mathematical description. In state-of-the-art literature, only the visual outlines [MK13] or raster images [Rot+15] of wedge-shaped impressions have been researched. We develop a purely semantic and uniform representation of cuneiform script.

Challenges Cuneiform script is a three dimensional, handwritten and highly variable script. Additionally, transcriptions are also handwritten and add another layer of idiosyncrasies. The script has no whitespace, is written very densely and is polyvalent. Therefore, cuneiform signs overlap and intersect and preclude automated segmentation.

Objectives We focus on modeling the features of wedge-shaped impressions that are necessary to decipher meaning and shared in all representations, e.g. in retro-digitized tracings, born-digital tracings and 3D-scans. It is our explicit non-goal to model visual features such as surface texture and the tablet shape and parts as those do not contribute to pure linguistic understanding of the written script.

Related Work We review previous methods used to categorize cuneiform signs and state the modeling objectives of our approach for a mathematical description of cuneiform script.

Input Our dataset consists of born-digital cuneiform tablet transcripts in the SVG format. These transcriptions are drawings of the visual outlines of the tablets and wedge-shaped impressions.

Output From our dataset we compute 1) a set of wedges in uniform feature vector representation and 2) an undirected topological graph representation on basis of semantic keypoints in the transcriptions.

Methods We introduce an implicit model of cuneiform script that represents cuneiform signs as graphs between key-points and an explicit representation as a set of wedge-shaped impressions modeled by a feature vector. We pattern-match the basic shapes of traced outlines to over-segment

wedges from our data. The over-extraction creates overlapping model interpretations and conflicts that are resolved by an heuristic rule-based approach as well as an optimizing optimal assignment based approach.

Evaluation We evaluate the precision and recall to compare the extraction performance of the presented strategies for retrieving implicit and explicit wedge models.

Publications Work on standardizing representations for heterogeneous sources has been published in [BMM15]. Work on optimally resolving conflicts from over-segmentation has been published in [BHM16].

3.2 Modeling Objectives and Related Work

In this section, we present related work in modeling cuneiform and describe our objectives for a model of the cuneiform writing system. Previous modeling approaches of cuneiform script and wedge-shaped impressions were limited to either enumeration and qualitative assessment of features or to purely visual approaches bound to the underlaying representation.

Borger [Bor04] maintains a comprehensive list of known cuneiform signs, their meanings and associated information. Signs can be found in this list by counting wedge types. For our purposes of computational analysis a set of prototypes is not flexible enough. It cannot represent as of yet unknown wedge constellations or writer specific idiosyncrasies. Counting or enumerating wedge types, in any sequential direction, is ambiguous as cuneiform script is a two dimensional writing system. While insufficient as an interchange representation in Chapter 4, Section 4.5 pp. 81, we evaluate two related approaches for similarity computation that make use of templates and feature enumeration.

Mara et al. [MK13] presents an approach for representing cuneiform signs by outlining them with parametric splines by extracting these from 3D-scans of cuneiform tablets. These outlines are strongly related to the underlaying format of 3D-scans and exhibit noise from the surface features of the acquired tablets. Rothacker et al. [Rot+15] use raster images to represent tablets and signs. While such an approach is highly general, it makes analyses depending

on an understanding of the constituents of the signs, e.g. n-gram clustering and pattern mining, not feasible.

Our approach to modeling cuneiform script focuses on a direct description of features necessary for linguistic analysis. Fitting models and learning any template wedges is avoided to provide a model capable to represent both pictographs and syllables. Further, the resultant model is independent from the underlaying representation and can be extracted from different heterogeneous sources.

3.3 Implicit Wedge Modeling

Wedges can be seen as distinctive elements of cuneiform script or as structures arising from written cuneiform signs. Modeling signs as a whole gives us the flexibility to model wedge constellations that arise from unique usages of the stylus and do not follow the typical wedge-shaped impressions, a style of writing in old ideographic cuneiform script.

3.3.1 Discovery of Connected Components

For born-digital tracings we divide the document into isolated components for faster processing. On page 46 we can see that this procedure reduces the processing time 32-fold for a highly polynomial algorithm in our pipeline. The following pattern matching algorithms work locally on individual strokes or connected components of strokes. For cuneiform script, extracted from 3D-acquired tablets by means of the approach presented by Mara et al. in [MK13], such a decomposition into components is already present.

We construct a set of connections by computing all intersecting strokes. Partitioning this set of connections into independent connected components yields sets of strokes which are then independently processed. Figure 3.1 on page 37 shows an excerpt of traced cuneiform script and components of intersecting strokes. Algorithm 2 on page 35 illustrates the process of splitting the set of strokes S into non-intersecting subsets $S_1 \cdots S_k$.

Algorithm 2 is composed of two parts. First, the construction of the graph structure by pairwise intersection of all combinations of strokes. For $n = |S|$

Algorithm 2 Split set of strokes into disconnected components.

procedure SPLIT(S)
▷ Here, we consider strokes as vertices
▷ and intersections of strokes as edges.
$V \leftarrow S$
$E \leftarrow \emptyset$

for $(s, s') \in S^2$ **do**
▷ For each pair of strokes, if they intersect
if $(s, s') \in I_S$ **then**
▷ add those to the list of edges.
$E \leftarrow (s, s')$
end if
end for

▷ Prepare a collection of independent sets.
$S_1 \cdots S_k \leftarrow \emptyset$
▷ Compute connected components of the graph
for $(l, s) \in$ components(V, E) **do**
▷ and add strokes to the respectively labeled set.
$S_l \leftarrow S_l \cup s$
end for

▷ Return collection of independent sets of strokes.
return $S_1 \cdots S_k$
end procedure

the count of strokes this operation has $O(n^2)$ complexity, since all pairs have to be enumerated and checked for presence in the set of all intersections I_S. Second, the construction of the subsets. Computing the connected components of a graph has a complexity of $O(n)$ [HT73]. There can be at most n disconnected components, the count of strokes, and assignment to the collection of sets of strokes has then a complexity of $O(n)$. The final complexity is then $O(n^2 + n + n) = O(n^2)$.

3.3.2 Definition of Keypoints

All our methods describing cuneiform script and wedge-shaped impressions use the notion of keypoints. These two-dimensional points are salient identifying features of wedges and wedge constellations and identify meaning without depending on the underlaying representation. We construct keypoints from two sources of points, endpoints and intersection points. Endpoints are constructed by finding two points farthest apart on the spline boundary of a stroke. Intersection points are created by pairwise intersecting all strokes in the document and placing points at the centers of the intersection areas. Figure 3.2 illustrates these two types of points used.

In this section, for modeling cuneiform signs without assumptions on wedge structure, we use keypoints as vertices of an undirected topological graph. Connections are computed by recovering the implied connections of intersecting strokes. Thus, the resultant graph represent the stroke connections between the salient features of a wedge constellation.

3.3.3 Graph Construction by Sweep-line

Recovering intersections of strokes for graph edges will not yield a proper representation of a cuneiform sign. We want graph edges to follow the visual shape of the strokes used to represent a sign. Therefore, graph edges are constructed by successively connecting keypoints that originate from the same stroke or have been created on said stroke through intersections with other strokes. Subsequently, by merging the created vertices before connecting those, we create a graph that visually resembles the underlaying visual representation of the wedge constellation. We denote a graph with the letter G and its vertices and edges with the letters V and E.

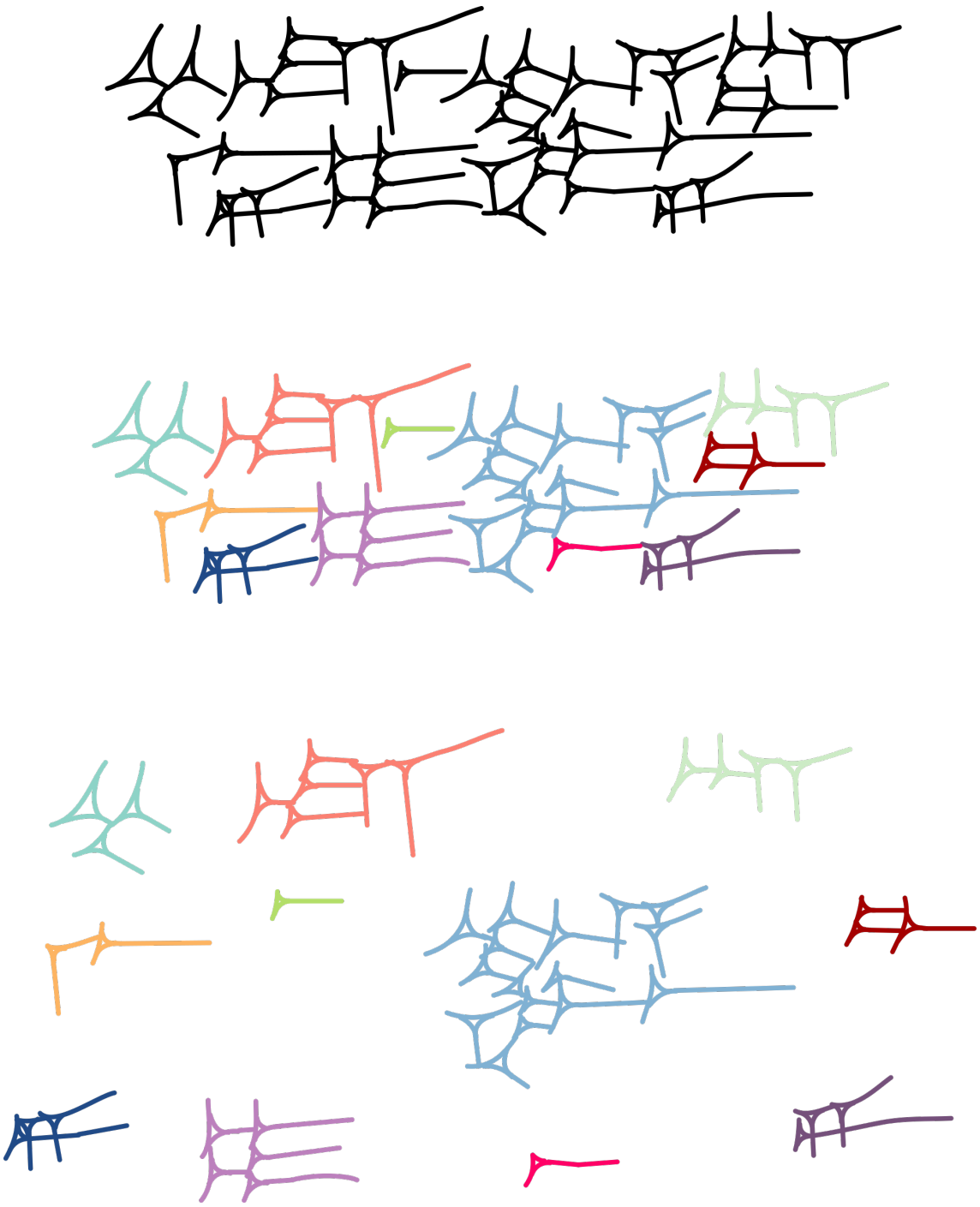
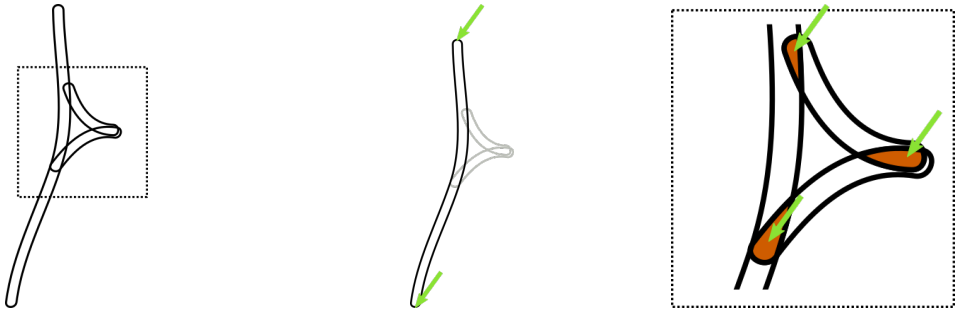
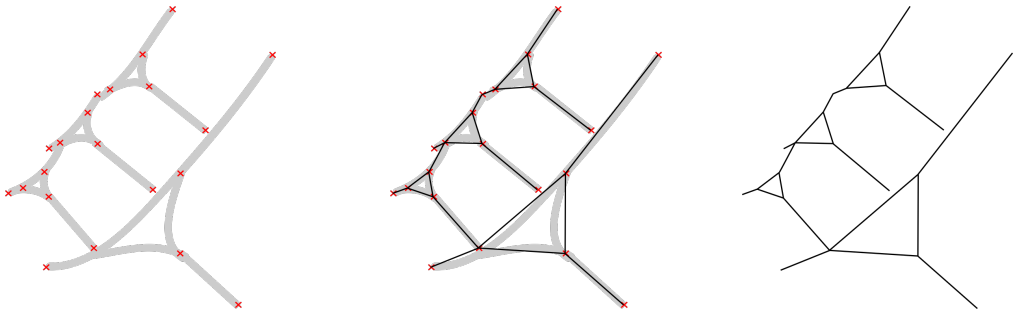


Figure 3.1: Excerpt from a born-digital tracing and an exploded view of its connected components of self-intersecting script. It is clearly visible that even signs from different lines intersect.



(a) (b) (c)

Figure 3.2: The keypoints of a cuneiform wedge. a) An exemplary wedge is outlined by three strokes. A different combinations of strokes outlining is possible as shown in Figure 2.10 on page 27. b) In this sign, endpoints of a stroke are used to denote the extents of the wedge-arms and c) intersection points are used to denote the vertices of the wedge-head.



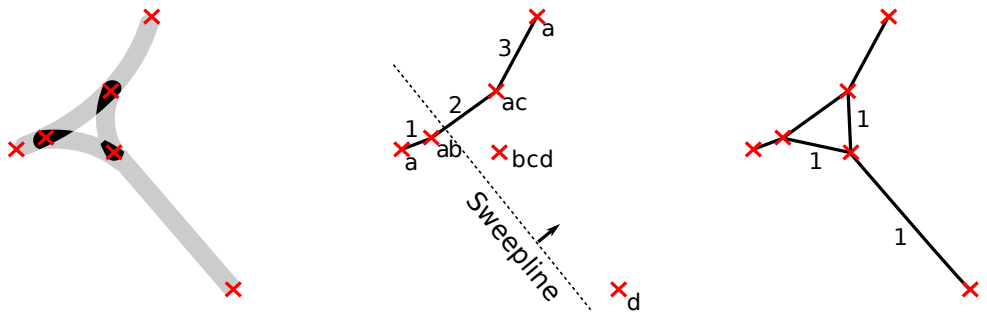
(a) (b) (c)

Figure 3.3: a) Endpoints and intersection points of a set of stroke in a cuneiform sign highlighted with red X marks. b) Graph overlaid on top of the strokes. c) The resulting graph of the cuneiform sign.

$$G = (V, E) \tag{3.1}$$

Figure 3.3 illustrates the visual and semantic equivalence of graph obtained from a set of strokes used to describe the tracing of a cuneiform sign.

Since there is no inherent order of intersection and endpoints on a stroke, we order all points originating from a stroke geometrically from bottom-left to top-right and then connect them in sequence. This approach resembles a sweep-line moving diagonally from bottom-left to top-right. Figure 3.4 illustrates this process. In rare cases this may create incorrectly connected points, i.e. the chain of edges is self-intersecting, if the stroke is slightly curved



(a) (b) (c)

Figure 3.4: A cuneiform wedge in spline representation. (a) Closed spline paths forming strokes (gray) are pairwise intersected (black). (b) The keypoints (marked X) of the strokes (marked with letters) are first ordered from bottom right to top left, then (c) the edges of the final graph are created by connecting the keypoints of each stroke in turn. The sequence is indicated by the numbers.

and the geometric ordering does not correctly represent the curvature of the stroke. Only few instances have been observed where this is the case. We present a closer evaluation of the extraction performance in the evaluation of this chapter in Section 3.5, pp. 54.

Algorithm 3 describes the process of reconstructing a graph from a set of strokes S. Let $n = |S|$ be the count of strokes in that set. The algorithm iterates over each stroke in turn and computes vertices for the resulting graph. Finding all incident intersecting strokes for a strokes has complexity of $O(n)$, each stroke in turn has to be checked for presence in the intersection set I_S. The computation of endpoints is independent of the input size n and is therefore taken to be constant. Since all possible strokes may be incident and intersecting, the ordering takes $O(n \log n)$ time. In summary, the time complexity of the sweep-line algorithm is $O(n * (n + n \log n)) = O(n^2 \log n)$.

Vertex de-duplication The intersection of two wedges is semantically meaningful. Yet, in some cases, intersection points and endpoints are very close, but do not share the same position. These are inaccuracies natural to human handwriting and artifacts of the tracing process. We recover implied connections by merging vertices that are closer than some threshold e. This threshold is set to be $\frac{1}{100}$ of the width of a typical stem of stroke in a born-digital tracing. Figure 3.5 illustrates this process.

Algorithm 3 Construct graph of keypoints from a set of strokes

procedure SWEEPLINE(S)
▷ Initialize empty lists for vertices and edges
$V \leftarrow \emptyset, E \leftarrow \emptyset$

▷ For each stroke in the current component S
for $s \in S$ **do**

▷ Compute endpoints for current stroke and
▷ all intersection points with other strokes
▷ resulting in the set of all keypoints.
$V' \leftarrow \text{endpoints}(s) \cup \{\text{center}(a, b)\ \forall (a, b) \in I_S \mid a = s \wedge b \in S\}$

▷ Order points diagonally, the same order a
▷ diagonal sweepline would encounter these points.
$V' \leftarrow \text{sort}(V')$ by increasing x and decreasing y

▷ Connect points in sequence to reconstruct
▷ original stroke.
for $(v_n, v_{n+1}) \in V'$ **do**
$E' \leftarrow (v_n, v_{n+1})$
end for

▷ Return vertices and edges of the graph.
$V \leftarrow V \cup V'$
$E \leftarrow E \cup E'$
end for
return (V, E)
end procedure

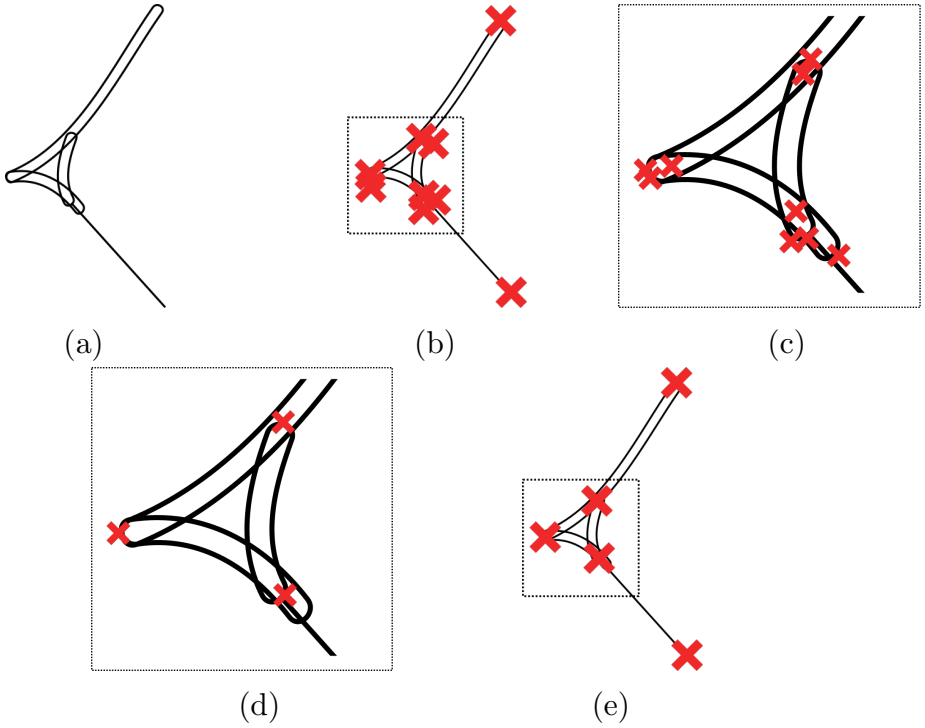


Figure 3.5: The same wedge type a) can be expressed by vastly different sets of strokes yielding different and duplicated intersection and endpoints b) and c). For a consistent labeling of keypoints we de-duplicate these intersection and endpoints d) and e).

The process of greedy de-duplication is described in Algorithm 4. It iterates over all vertices and builds a list of seen vertices. Any new vertex is first checked before its added to a new set of vertices. Iteration has $O(n)$ time complexity, distance checking all added vertices has $O(n)$ time complexity. Since this check is performed for all vertices the resulting time complexity for Algorithm 4 is $O(n^2)$.

Steps of Reconstruction The process of extracting a graph from the set of strokes of a born-digital tracing of a cuneiform tablet can be thus described in the following steps.

1. Find connected components of self-intersecting strokes to reduce working set size (Algorithm 2 on page 35).

2. Compute graph from set of strokes by connecting keypoints (Algorithm 3).

3. Reconstruct implied wedge connections by merging close keypoints using a greedy approach (Algorithm 4).

The resultant graph is then used in Chapter 4, Section 4.4.2, pp. 69 for comparing cuneiform symbols. Representing symbols by means of a graph does not presume any underlying structuring elements and allows the representation of any set of intersecting strokes. But the free-form expression allowed by graph models makes defining a semantically meaningful similarity metric, one that models changes in graphs similar to how an expert perceives changes in the respective symbols, difficult. In the following section we introduce wedge models based on tool-usage.

Algorithm 4 Greedily de-duplicate vertices

procedure DEDUPLICATE(V)
▷ Initialize empty new set of vertices.
$V' \leftarrow \emptyset$

▷ For each vertex in the set of vertices.
for $v \in V$ **do**

▷ If there is no other vertex in the new set of
▷ vertices closer than some ϵ.
if $\forall v' \in V' : \|\vec{vv'}\| > \epsilon$ **then**

▷ Add it to the new set of vertices.
$V' \leftarrow v$

▷ This approach is greedy since the first
▷ vertex to be added to the set of new vertices
▷ suppresses all following even if their
▷ placement were more optimal.
end if
end for

▷ Return new set of vertices where no two vertices
▷ are closer than some ϵ.
return V'
end procedure

3.4 Explicit Wedge Modeling

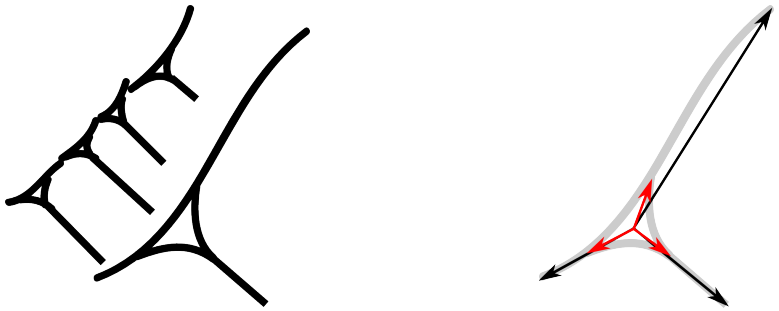
In contrast to implicit wedge modeling, in this section we model the wedge-shaped impressions directly as the most basic constituent of written cuneiform script. Since wedge-shaped impressions are the result of using a rectangular stylus to write, we therefore parametrize the tool usage to describe handwriting. This approach makes the results of our presented methods more interpretable to Assyriologists where understanding of cuneiform signs begins with understanding tool usage.

Since wedge-shaped impressions are concrete objects in this section, we introduce the notation of W which represents the set of all wedges. We also decompose a single wedge $w \in W$ into a wedge-head and its wedge-arms $w^h \cup w^a = w$. Wedges are sets of strokes from the set of all strokes $w \subset S$ and its constituents are, in turn, disjoint subsets of the wedge $w^h \cap w^a = \emptyset$.

3.4.1 Keypoint Features as Shared Representation

Here, we introduce a common representation of wedges and cuneiform signs across heterogeneous formats of tablets, e.g. retro-digitized tracings, born-digital tracings and 3D-acquired tablets. A common representation has to describe the most salient features of wedge-shaped impressions and has to leave out any unnecessary detail. One important feature of a wedge is the shape of the initial impression in the clay, the wedge-head. The other important feature is the direction and length of any wedge-arms, the ridges extending from the wedge-head created by pulling the rectangular stylus. Experts categorize and recognize wedges by the size and orientation of their wedge-head and wedge-arms. The keypoint feature descriptor derives directly from the way wedges are drawn in transcriptions. It models wedges using six two-dimensional points as shown in Figure 3.6. The first three points are the vertices of the three pairwise intersecting strokes forming the wedge-head. The last three points are endpoints of the wedge-arms attached to the respective wedge vertices.

These points form the keypoints of the keypoint feature descriptor and are the same keypoints as introduced in Section 3.3.2, pp. 36 of this chapter. We base our description on the same set of keypoints as in the implicit wedge model, but choose six specifically designated ones to describe a wedge-shaped



(a) (b)

Figure 3.6: a) A wedge constellation and b) the keypoints of one of its wedges.

impression. The choice which keypoints are used is determined by one of the conflict resolution strategies detailed later.

Let the $F^k \subset \mathbb{R}^{12}$ be the set of twelve-dimensional feature descriptors of a wedge $f^k \in F^k$. It contains six two-dimensional points, three for the triangular wedge-head $h_1 \cdots h_6$ and three for the endpoints of the wedge-arms $a_1 \cdots a_6$.

$$f^k = (\underbrace{h_1, h_2, h_3, h_4, h_5, h_6}_{\text{Wedge-head}}, \underbrace{a_1, a_2, a_3, a_4, a_5, a_6}_{\text{Wedge-arms}}) \tag{3.2}$$

The keypoint feature descriptor of wedges, and subsequently signs as sets of wedges, is used as a common interchange format between all of our cuneiform tablet sources. Each of these has different representations, e.g. as 3D-scans, retro-digitized tracings, born-digital tracings. The keypoint feature descriptor encodes all semantically relevant information to reconstruct the represented wedge-shaped impressions while leaving out any medium related artifacts, e.g. lighting conditions, ridge depth or material condition. All subsequent feature descriptors, e.g. the property feature descriptor and the template feature descriptor, are derived from the keypoint feature descriptor.

3.4.2 Over-Segmentation for Robust Extraction

Our modeling approach makes use of the unique and very regular shape in which wedge-shaped impressions are traced. We match wedge-head patterns by finding three pairwise intersecting strokes and associated wedge-arms by finding all strokes intersecting the strokes of the wedge-head. Since cuneiform

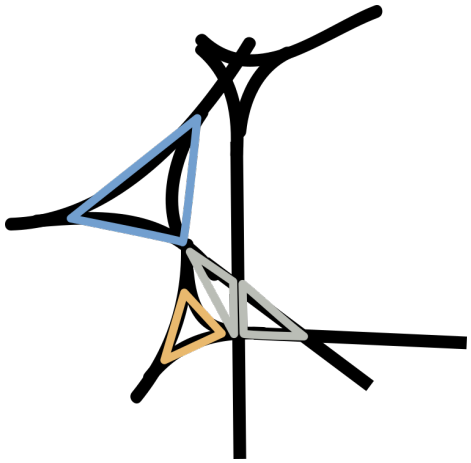


Figure 3.7: Hypothesized exemplary wedge-heads from Algorithm 5 on set of strokes depicting a wedge constellation. Some, but not all walks of 3, i.e. triangles of three pair-wise intersecting strokes, are true-positives wedge-heads (blue and orange). Our over-segmentation approach finds all possible wedge-like patterns and removes false-positives (gray) through subsequent pruning and conflict resolution.

is densely written and wedges intersect each other, many false-positive wedge-heads and wedge-arms are detected.

We generate wedge candidates for all possible combinations of the matched wedge-heads and wedge-arms. These candidates intersect each other and are in potential conflict since they share some of the underlaying strokes. That is, the same stroke may be responsible for two different wedge-heads, one of those being a false-positive, as shown in Figure 3.7. This a case of over-segmentation where more model instances are computed than are supported by the underlaying data.

Algorithm 5 describes the process of enumerating wedge candidates. A set of three pairwise intersecting strokes, a candidate wedge-head, is denoted by w^h. A candidate subset of incident wedge-arms is denoted by w^a and generated by enumerating the power-set $P_{1\cdots3}()$ of all incident strokes limited to sets with cardinalities between 1 and 3. This procedure generates all possible combinations of wedge-heads with between one or three wedge-arms.

For $n = |S|$ the count of strokes in S, enumeration of all walks of 3 from the set of intersections I_S has a time complexity of $O(n^3)$. All intersecting 3-tuples are enumerated and only those which are walks of 3 are kept. Enumerating all subsets for the limited power set of 3 elements, also has a time complexity of $O(n^3)$. All strokes could possibly be incident and intersect the wedge-

head. Since the power set is constructed for each wedge-head, the final time complexity of Algorithm 5 is $O(n^3 * n^3) = O(n^6)$.

This is the most expensive algorithm in our extraction pipeline and the reason why the dataset from born-digital tracings has to be divided into connected components. For such high polynomial runtime a simple division into two independently processed parts already results in a reduction of processing steps to the account of $O((n * \frac{1}{2})^6) = O(n^6 * \frac{1}{2^6})$, therefore for two invocations a $(\frac{2}{2^6} = \frac{1}{32})$-fold reduction in runtime.

Algorithm 5 Generate wedge candidates

procedure HYPOTHESIZE(S)
▷ Initialize the empty set of hypothetical wedges
$W \leftarrow \emptyset$

▷ For every triangle, find walks of 3
▷ on the graph of intersecting strokes.
for $w^h \in \{\{a, b, c\} \mid (a, b), (b, c), (c, a) \in I_S\}$ **do**

▷ Choose every permutation of incident intersecting
▷ strokes as possible wedge-arms for the current
▷ wedge-head triangle.
for $w^a \in P_{1\ldots3}(\{a \mid (s, a) \in I_S \wedge s \in w^h\})$ **do**

▷ Record the hypothesized wedge-model for later
▷ pruning.
$W \leftarrow W \cup \{w^h \cup w^a\}$
end for
end for

▷ Return the generated set of hypothesized wedge-models.
return W
end procedure

A-Priori Modeling Constraints After over-segmenting the set of strokes, we prune the set of generated candidate wedges by applying a set of thresholds. These thresholds are a-priori expert knowledge of written cuneiform. The area of the wedge-heads has to be between the thresholds for *small-head* size t^{sh} and *big-head* size t^{bh}. The area is computed using Heron's formula.

$$\begin{aligned}
\text{area}(w^h) &= \frac{\sqrt{(a+b+c)(a+b-c)(b+c-a)(c+a-b)}}{4} \\
\text{where } a &= \|\vec{b}-\vec{c}\|, b = \|\vec{a}-\vec{c}\|, c = \|\vec{a}-\vec{b}\| \\
\text{and } \{\vec{a},\vec{b},\vec{c}\} &= \{\text{center}(s_1,s_2), \text{center}(s_2,s_3), \text{center}(s_3,s_1)\} \\
\text{and } \{s_1,s_2,s_3\} &= w^h
\end{aligned} \tag{3.3}$$

The angle of incidence of the wedge-arms to the triangle corners and the lengths of the wedge-arms are constrained. Since a single stroke can contribute to an edge of a wedge-head and a wedge-arm, we consider endpoints from all strokes of a wedge candidate. That is, an endpoint from a stroke used to draw a part of a wedge-head also contributes points that are interpreted as endpoints for wedge-arms.

$$\begin{aligned}
\text{incidence}(w^h, w^a) &= \left\{ \max_{\vec{a}\in A} \langle \vec{a}-\vec{c}, \vec{b}-\vec{c}\rangle \;\middle|\; \vec{b} \in B \right\} \\
\text{angle}(w^h, w^a) &= \left\{ \max_{\vec{a}\in A} \cos^{-1} \frac{\langle \vec{a}-\vec{c}, \vec{b}-\vec{c}\rangle}{\|\vec{a}-\vec{c}\|\|\vec{b}-\vec{c}\|} \;\middle|\; \vec{b} \in B \right\} \\
\text{where } A &= \bigcup_{s\in w^h \cup w^a} \text{endpoints}(s) \\
\text{and } \vec{c} &= \text{mean}(B) \\
\text{and } B &= \{\text{center}(s_1,s_2), \text{center}(s_2,s_3), \text{center}(s_3,s_1)\} \\
\text{and } \{s_1,s_2,s_3\} &= w^h
\end{aligned} \tag{3.4}$$

The dot-product of the vector from the center of the wedge-head to a wedge-head vertex $\vec{a}-\vec{c}$, and the vector from the center to a wedge-arm endpoint $\vec{b}-\vec{c}$ has to be greater than the threshold t^α. We set $t^\alpha = 45°, t^{\text{sh}} = 10, t^{\text{bh}} = 50$.

$$\begin{aligned}
W \leftarrow \{w \in W \mid\; & t^{\text{sh}} < \text{area}(w) < t^{\text{bh}} \wedge \\
& \forall \alpha \in \text{angle}(w) : \alpha > t^\alpha\}
\end{aligned} \tag{3.5}$$

Even after pruning the set of candidate wedges, the resulting set of wedges is not guaranteed to be free of conflicts where candidate wedges share the same strokes. To remove any conflicts between wedge candidates, we employ a set of conflict resolution strategies.

3.4.3 Conflict Resolution using Heuristics

Our first approach uses a set of rules and heuristics to resolve conflicts. Strokes that contributed to identifying wedge-heads are never assigned to wedge candidates to fill the role of wedge-arms.

$$W \leftarrow \{\{w^h, w^a\} \in W \mid \nexists\{w'^h, w'^a\} \in W : s \in w^a \wedge s \in w'^h\} \tag{3.6}$$

If two wedge candidates share the same stroke making up their wedge-head the smaller candidate wedge is discarded.

$$\begin{aligned} W \leftarrow \{\{w^h, w^a\} \in W \mid \nexists\{w'^h, w'^a\} \in W : \\ s \in w^h \wedge s \in w'^h \wedge \text{area}(w) < \text{area}(w')\} \end{aligned} \tag{3.7}$$

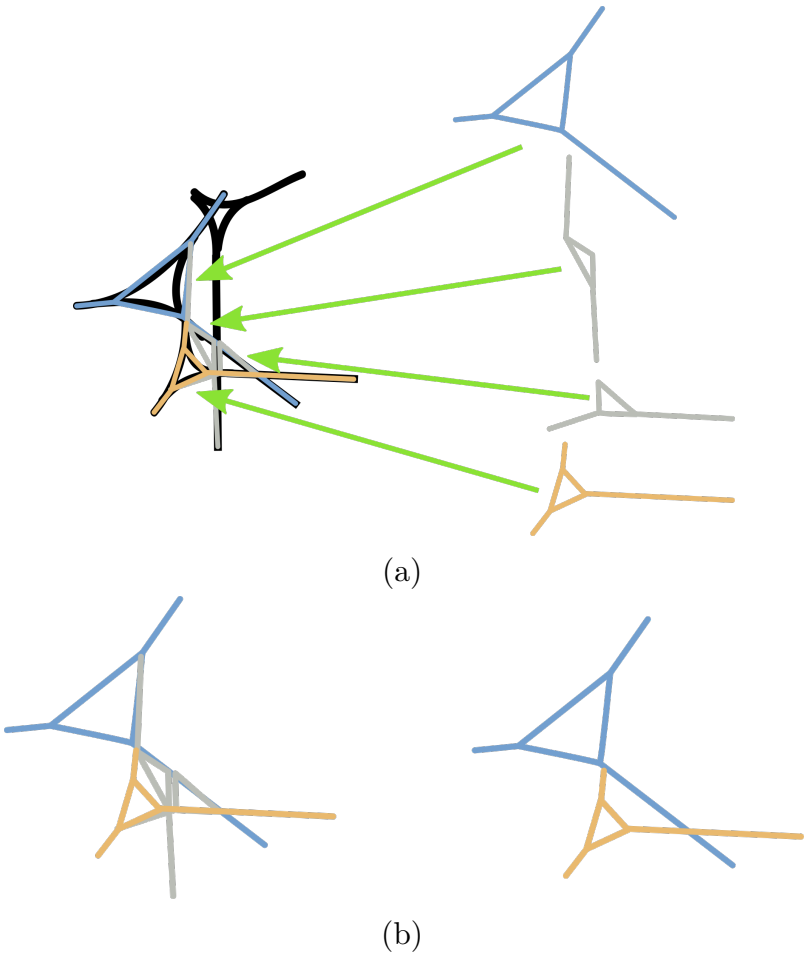
Strokes that contribute to wedge-arms are assigned the candidate wedge where the stroke is most in line with one of the wedge-head corners.

$$\begin{aligned} W \leftarrow \{\{w^h, w^a\} \in W \mid \nexists\{w'^h, w'^a\} \in W : \\ s \in w^a \wedge s \in w'^a \wedge \max_{d \in \text{incidence}(w^h, s)} d < \max_{d' \in \text{incidence}(w'^h, s)} d' \end{aligned} \tag{3.8}$$

The resultant set of wedge-models is non-overlapping and follows the interpretation approach of an expert. Hypotheses most closely resembling ideal wedges are kept, while conflicting less ideal hypotheses are discarded. Figure 3.8 shows an overview of our heuristic pruning pipeline.

3.4.4 Conflict Resolution by Optimal Assignment

Our previous approach is sensitive to the ordering of its rules and brittle, i.e. produces unpredictable assignments if complex conflicts are present. Here, we introduce a conflict resolution strategy that works with the complete connected set of intersecting strokes. We model the conflict resolution as an optimization of an assignment problem. On one side are strokes and on the



(a)

(b)

Figure 3.8: a) Hypothesized exemplary wedge-models in a set of strokes depicting a wedge constellation. The intersecting wedge-models are shown in non-overlapping detail to the right. The gray wedge models receive worse score from the heuristic functions since their wedge-heads are small and their wedge-arms are at acute angles. b) From a set of hypothesized wedge-models our heuristic model prunes all unlikely candidates and leaves behind a non-overlapping set of well-formed wedges.

other are hypothesized wedge candidates. The optimization function prefers assigning strokes to bigger wedge-heads and to wedge-arms where the strokes are more in line with a wedge-head corner. The assignment problem is solved using the Hungarian algorithm. From the resulting assignment we recover wedge candidates that are viable, that is, where more than 3 strokes have been assigned to form a complete wedge.

The Hungarian Algorithm Given a set of worker and a set of tasks we want to find an assignment of workers to tasks. Any assignment of a worker to a task incurs some varying cost. The assignment problem is the problem of finding an assignment that minimizes that cost. The Hungarian algorithm by Munkres [Mun57] traverses the edges of the simplex constraining the solution space to find an assignment. The runtime of the algorithm with respect to the count of tasks n is contained in $O(n^3)$.

Let c_{ij} be the costs of assigning some task i to worker j and let x_{ij} denote the assignment of task i to worker j. Then the typical assignment problem is:

$$\begin{aligned} \text{Minimize} \quad & \sum_i \sum_j c_{ij} x_{ij} \\ \text{subject to} \quad & \sum_i x_{ij} = 1 \quad \text{and} \quad \sum_j x_{ij} = 1 \\ \text{and} \quad & x_{ij} \in \{0, 1\} \end{aligned} \tag{3.9}$$

The first constraint ensures that every worker is assigned to exactly one task and the second constraint ensures that every task is assigned to exactly one worker. The third constraints ensures that the resulting assignments are sensible, i.e. either a worker is assigned a task $x_{ij} = 1$ or he is not assigned to it $x_{ij} = 0$.

Mathematical Reformulation Let A be a set of assignments (s, w) of strokes s to wedge candidates w. From this set, only assignments are valid where a stroke are capable being part of the wedge candidate, i.e. they are either part of the wedge-head or one of its wedge-arms.

$$A = \{(s, w), s \in S, w \in W \mid s \in w^h \vee s \in w^a\} \tag{3.10}$$

We define two score functions c^h_{sw} and c^a_{sw}. The function c^h_{sw} computed the benefit of assigning stroke s to wedge candidate w as part of the wedge-head, and the function c^a_{sw} computes the benefit of assigning the stroke s as a wedge-arm. If an assignment is not valid, the score is set to -1, to avoid assigning a stroke to the wedge candidate. In the score matrix a block is reserved as a sink with scores set to 0 for such strokes that violate the constraints in the score functions. The weighting factor ω^h prioritizes the formation of wedge-heads instead of assigning additional wedge-arms to unfinished wedge-heads. Different weightings have been tested $(0.1, 5, 10, 20)$ with ω^h providing the best results. Wedge-heads with bigger head area are prioritized over smaller wedge-heads.

$$c^h_{sw} = \begin{cases} \text{area}(w) * \omega^h, & \text{if } (s,w) \in A, \\ -1, & \text{else} \end{cases} \tag{3.11}$$

Wedges have obtuse angeled wedge-arms, we disallow angles smaller than the threshold t^α. Wedge-arms are preferentially assigned to wedge candidates where the wedge-arms are long and in line with the wedge-head vertices.

$$\begin{aligned} c^a_{sw} &= \begin{cases} \hat{d}, & \text{if } (s,w) \in A \wedge \hat{\alpha} < t^\alpha \\ -1, & \text{else} \end{cases} \\ \text{where } \hat{d} &= \max_{d \in \text{incidence}(w^h, s)} d \\ \text{and } \hat{\alpha} &= \max_{\alpha \in \text{angle}(w^h, s)} \alpha \end{aligned} \tag{3.12}$$

Each wedge candidate is given 6 columns for the six positions a stroke can be assigned to. Three slots in the wedge-head and three slots for the wedge-arms. For reasons of readability, we denote this group of columns for one wedge candidate with one index and elide complex modulo operations. Therefore, while there are m wedge candidates, there are $6m$ columns reserved for those. We arrange the costs in a $(6m + n) \times n$ matrix for optimization. Figure 3.9 shows a simplified and illustrative version of this matrix for an exemplary conflict resolution task. The matrix has n rows for each of the strokes to be uniquely assigned. Columns $1 \cdots 6m$ represent the wedge candidates the strokes can be assigned to. For strokes that cannot be assigned to any wedge candidates, because they violate the constraints in the score functions, encoded as -1 scores, a block reserved in $(1 \cdots n) \times (6m \cdots (6m + n))$ functions as a

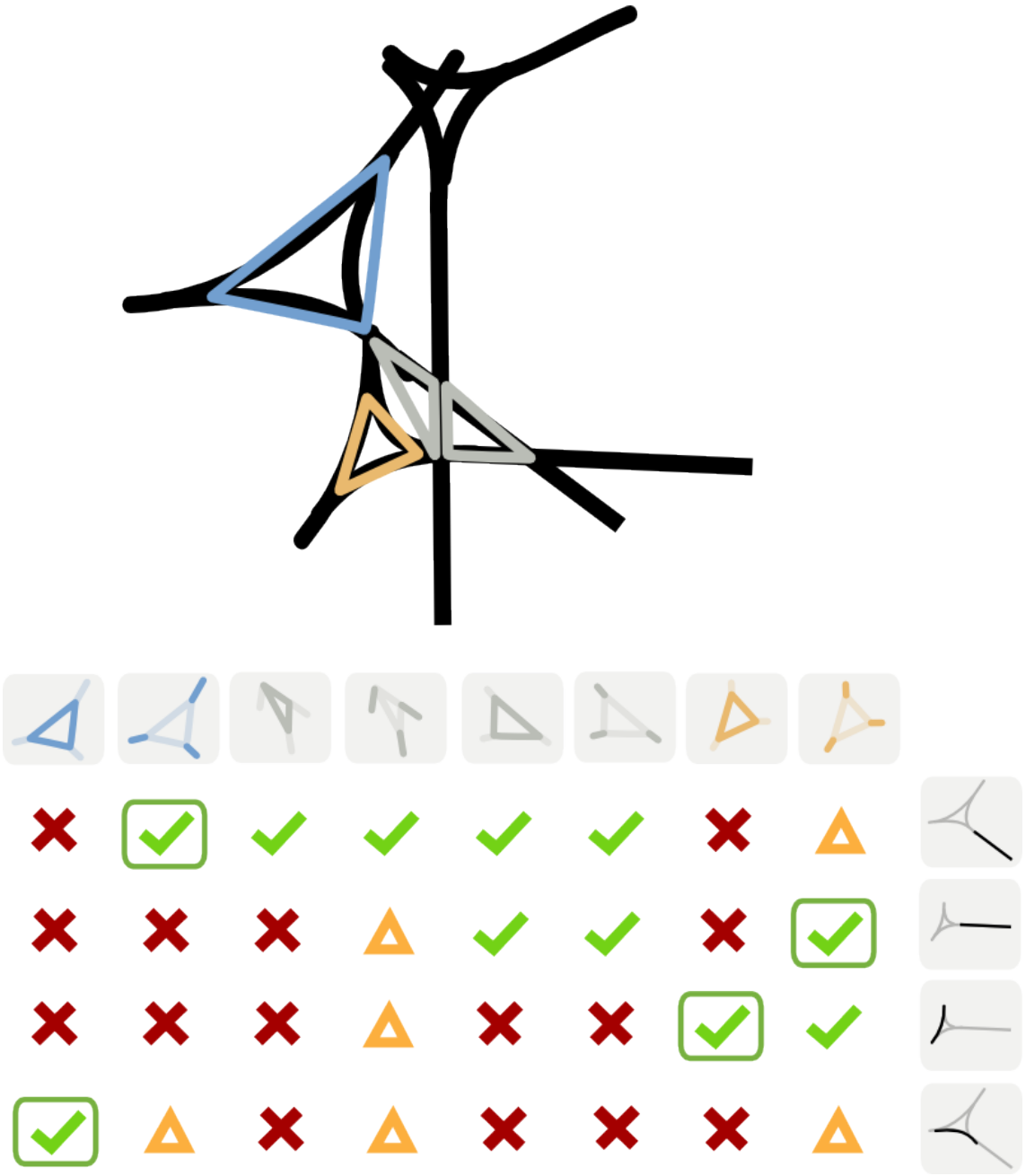


Figure 3.9: A simplified visual representation of the score matrix used for optimally assigning strokes to wedges. False positive candidate wedges are gray triangles. True positive candidate wedges are orange and blue. The green check mark signifies a match with high score, the orange triangle a match with low score and the red cross a match that is not allowed.

sink. Strokes are assigned to this block only if any other assignment would result in a score less than zero.

$$C = \begin{pmatrix} \overbrace{c^h \dots c^a}^{w_1} & \dots & \overbrace{c^h \dots c^a}^{w_m} & \overbrace{0 \dots -1}^{s_1 \dots s_n} \\ c^h \dots c^a & \dots & c^h \dots c^a & -1 \dots -1 \\ c^h \dots c^a & \dots & c^h \dots c^a & -1 \dots -1 \\ \vdots & \ddots & \vdots & \ddots \\ c^h \dots c^a & \dots & c^h \dots c^a & -1 \dots 0 \end{pmatrix} \begin{matrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_n \end{matrix} \tag{3.13}$$

We find a score maximizing assignment by solving the assignment problem with the Hungarian algorithm [Mun57]. Viable assignments are recovered by retrieving all wedge candidates that were at least assigned three strokes for the wedge-head. Wedge-arms assignments are optional. Since one wedge candidate spans 6 columns, $\sum C_{mn}$ denotes the sum over these six columns.

$$W = \{w_i \in \{w_1 \cdots w_m\} \mid \sum_{j \in \{1 \cdots n\}} C_{ij} \geq 3\} \tag{3.14}$$

Having determined the strokes that constitute the tracing of a wedge-shaped impressions, we move on to extracting the six defining keypoints from this set of strokes.

3.4.5 Construction of the Keypoint Feature Descriptor

Regardless of the method used to determine the strokes representing a wedge-shaped impression, the process of constructing a keypoint feature descriptor does not change. The vertices of the wedge-head are computed from pairwise intersections of the strokes constituting the wedge-head in the prior assignment. The vertices of the endpoints of the wedge-arms are the most distant endpoints of any stroke from a vertex in the wedge-head.

Let $\{w^h, w^a\} = W$ be the sets of strokes in the wedge-head w^h and the wedge-arms w^a of an extracted wedge model W. All keypoint feature descriptors define wedge-arms endpoints but, by definition, not all wedge models contain strokes representing wedge-arms. Wedge-arms vertices are therefore computed

from all endpoints in a wedge model W, including strokes contributing to the wedge-head tracing.

$$
\begin{aligned}
f^k = \{ & \operatorname{center}(w_1^h, w_2^h), \\
& \operatorname{center}(w_2^h, w_3^h), \\
& \operatorname{center}(w_3^h, w_1^h), \\
& \max_{p \in \text{endpoints}(W)} \|p - \operatorname{center}(w_1^h, w_2^h)\|, \\
& \max_{p \in \text{endpoints}(W)} \|p - \operatorname{center}(w_2^h, w_3^h)\|, \\
& \max_{p \in \text{endpoints}(W)} \|p - \operatorname{center}(w_3^h, w_1^h)\| \}
\end{aligned}
\tag{3.15}
$$

This description of wedge-shaped impressions describes the most basic and semantically meaningful keypoints and serves as a basis for any other feature vector descriptions of wedges in our work.

3.5 Evaluation of Extraction Accuracy

In this chapter we introduced three methods for extracting cuneiform script from a born-digital tracing. We evaluate the extraction accuracy of these methods by computing the precision and recall. Wedges and signs are recovered from Cuneiform tablets and correctly extracted wedges (true positives), incorrectly extracted wedges (false positives), correctly ignored strokes (true negatives), and incorrectly ignored wedges (false negatives) are counted. We perform the evaluation manually with the help of expert knowledge and decide whether the results correctly represents the underlaying tracing of wedge-shaped impressions. Figure 3.10 shows the various extraction strategies and their results applied to a single born-digital tracing. Since the implicit graph model uses no preconceived notions of a wedge-shaped impression, it is able to capture all strokes used to trace wedges accurately. At same the time it also identifies and extracts many non-script elements, such as dividing lines and damage. Therefore, the evaluation metrics presented below are not applicable to the graph extraction.

The table in Figure 3.11 summarizes the results of evaluation of the explicit extraction of wedge-shaped impressions. Even though the precision and recall

Figure 3.10: Extraction results of the three presented approaches, sweep-line extraction of graphs from keypoint, heuristic conflict resolution of wedge models and optimizing conflict resolution. Very dense cuneiform script is annotated in red circles. The optimizing is able to correctly infer wedge placement and configuration. The graph based approach is visually correct but has no concept of wedges, e.g. the separator line is also extracted.

	TP	FP	TN	FN	Precision	Recall
Heuristic	1780	95	96	43	0.949	0.976
Optimizing	1947	26	91	34	0.987	0.983

Figure 3.11: Extraction accuracy values of the three presented strategies to extract wedge-shaped impressions from born-digital tracings.

values are very similar for the heuristic and optimizing extraction approach, Figure 3.10 shows that for some very densely written cuneiform signs the optimizing approach provides significantly more correct results. For complex and dense signs the optimizing extraction thus performs significantly better.

3.6 Summary

In this section we introduced the concept of a feature descriptor, to mathematically describe properties of an object, and the Hungarian algorithm, to optimally solve an assignment problem. We then described the approach to pattern matching wedges in an unstructured and unordered set of intersecting strokes. Connected components of strokes are independently processed for reasons of computational efficiency. Two approaches were presented in this chapter. An implicit approach that models whole cuneiform words as mathematical graphs which are extracted by means of a sweep-line. The other approach models wedge-shaped impressions explicitly by means of a keypoint feature descriptor with six keypoints, three for the triangular wedge-head and three for the endpoints of the wedge-arms.

We established our common interchange format for semantically representing wedge-shaped impressions and wedge constellations by introducing the keypoint feature descriptor which represents wedges by the three vertices of its wedge-head and the three endpoints of its wedge-arms. These methods and results have been published in our work on evaluating cuneiform and wedge representations [BGM15b; BGM15a] and our work establishing common interchange format for semantically representing cuneiform tables from disparate sources [BMM15; MB15; Mas+16].

Since cuneiform script is densely written, many wedges are intersecting and creating false-positives. We solve this challenge with two different methods. A greedy method applies a set of heuristics to prune false-positives and then

greedily resolves conflicts by preferring bigger wedge-heads with obtuse wedge-arms. Our second method uses a globally optimizing method, first transforming the conflict set into an assignment problem with two score functions. Then the Hungarian algorithm is used to find an optimal assignment from which wedges are recovered. We present these results in our work on symbol spotting in [BHM16].

4 Wedge Similarity

With the extraction of the basic geometrical structures and patterns from tracings of cuneiform tablets, in the form of a graph of keypoints or as a set of wedges, we now approach the challenge of comparing wedge constellations for similarity. First, we define the notion of similarity functions and kernel functions. These are used to measure an intuitive understanding of similarity between objects. Based upon these definitions, we then introduce five similarity functions with no a-priori assumption about the structure of wedge constellations and three similarity functions that require wedges to be the most basic component of wedge constellation.

4.1 Challenges and Objectives

In this chapter we define the properties of a suitable similarity function and develop six similarity functions specifically for cuneiform and evaluate those to three state-of-the-art similarity function for Latin script.

Motivation Computational analysis of handwritten signs requires a robust definition of equality and inequality in the presence of noise. Any analysis algorithms, e.g. clustering and pattern mining and learning algorithms including Support Vector Machine (SVM) and visualizations including t-stochastic neighborhood embedding (TSNE), make use of similarity functions for the objects under study.

Challenges Different from Latin script, cuneiform script is arranged in two dimensions, diminishing the accuracy of sequential models. Additionally, the shapes and bounds of cuneiform signs are irregular and cannot be contained in a fixed-size raster image feature descriptor as used for Chinese signs.

Objectives A similarity function for cuneiform script must capture its horizontal and vertical complexity and be robust to two-dimensional perturbations. Additionally, for cuneiform sign search in large datasets it has to be computationally efficient.

Related Work We review work in historical word-spotting and approaches for computing word image features. Similarity computations are based on linear sequential models, e.g. Markov chains, and feature matching with zones of interest.

Input We develop novel similarity functions based on the structures extracted in Chapter 3. Additionally, we adapt state-of-the-art similarity functions to our data for evaluation.

Output Our similarity functions measure the notion of semantic similarity between two cuneiform sign. They are robust w.r.t. to noise and perturbations which preserve the meaning of the signs. They models the expectations of similarity of experts in Assyriology.

Methods We employ methods from graph similarity computation such as graph kernels and the spectral decomposition for our graph representation from Chapter 3. For the explicitly modeled set of wedges, we develop methods based on bag-of-properties approaches and Gaussian mixture models.

Evaluation We create ground truth of similar signs by employing an expert to tag a set of signs by their meaning in the Borger [Bor04] list of cuneiform signs. Then, we evaluate our similarity functions and the state-of-the-art functions on basis of a precision-recall graph. We also evaluate the runtime performance of an assignment approach to show how brute-force algorithms can outperform optimized algorithms for very small data sizes.

Discussion We find that current state-of-the-art methods for retrieval of handwritten Latin words do not model vertical complexity of cuneiform signs sufficiently. We discuss the behavior of the similarity functions on our data.

Publications Work in this chapter on implicit representation of cuneiform signs has been published in [BGM15b] and work on explicit representation has been published in [BGM15a].

4.2 Background on Similarity

In this section we introduce mathematical concepts and terms used throughout this chapter to characterize similarity measures and the computation thereof.

4.2.1 Feature Descriptors

In machine learning a feature describes a measurable, independent and discriminative property of an object under study. A feature-vector combines many such properties into a vector. Paired with a distance function supporting the triangle inequality and a well-defined inner product, a Hilbert space is created. These properties enable the use of machine learning algorithms, including Gaussian Mixture Model (GMM), Hidden Markov Model (HMM), K-Nearest Neighbors (KNN) classifiers, SVM classifiers, and k-Means clustering.

4.2.2 Distance Functions

Regardless of the representation of wedge constellations, we have to define a distance function which accurately models the judgment of an expert as to how similar two different wedge constellations are. A distance function (or metric) is used to define a distance between two objects. For the purpose of similarity comparisons, a distance function is chosen that models a perceptive quality of the objects under study, e.g. the presence or absence of a feature or the sum movements necessary to align two sets of points. An ideal distance function satisfies the following conditions.

$$\begin{aligned} &d(x,y) \geq 0 \\ &d(x,y) = 0 \Leftrightarrow x = y \\ &d(x,y) = d(y,x) \\ &d(x,z) \leq d(x,y) + d(y,z) \end{aligned} \tag{4.1}$$

The distance functions and kernel functions presented in this chapter do not satisfy these conditions in all cases. For our purposes of comparing wedge constellations for retrieval and analysis, an approximate adherence to these

conditions is sufficient. Instead of strict adherence to these conditions, a usable distance function needs to be robust and discriminatory. Robustness implies that small perturbations in the perceived similarity of two objects should only result in a small change in distance between those objects. A discriminatory distance function should assign a significantly higher distance value to distinctively dissimilar objects. For an ordered set of points, the keypoint feature description of wedges, the sum of squared Euclidean distances satisfies these requirements.

$$d(x, y) = \sum_i \|\vec{x_i} - \vec{y_i}\|^2 \tag{4.2}$$

The Euclidean distance is a special case of the p-Norm [BS61]. Other distance functions include the cosine distance [TSK05].

4.2.3 Kernel functions

Instead of a distance function, for machine learning algorithms, including those based on the SVM framework [CV95], a kernel is used to define the similarity of two objects. The kernel computes the dot-product of two objects lifted into a higher dimensional space, also known as the kernel trick [Sch+00], since no mapping into the high-dimensional space is actually performed.

$$k(x, y) = \langle \phi(x), \phi(y) \rangle \tag{4.3}$$

The feature map ϕ does not need to be defined as long as the kernel k is an inner product of two objects x and y. Schölkopf [Sch00] presents a method to compute a distance function from a kernel formulation.

$$\|d(x, y)\|^2 = k(x, x) + k(y, y) - 2k(x, y) \tag{4.4}$$

Using this equivalence, we can convert the kernel functions, presented in this chapter, into functions behaving like distance functions to compare the similarity between two objects.

4.2.4 Usage of Points and Keypoints as Feature Descriptors

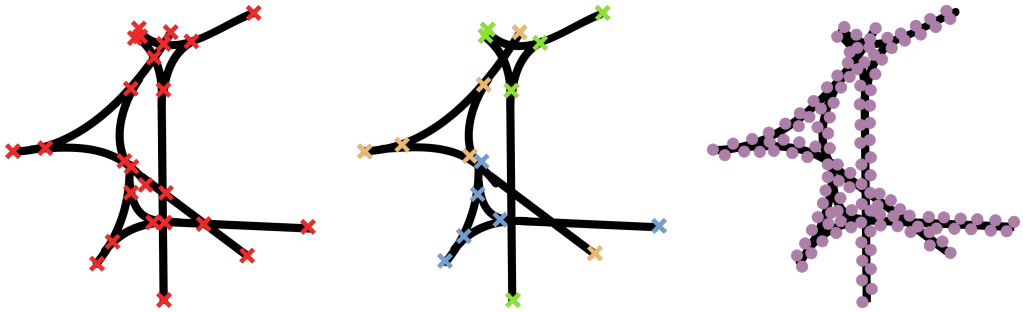
In Chapter 3, Section 3.3.2 pp. 36 we introduced the concept of keypoints that describes points of interest in wedge-shaped impressions, that is, intersection points and endpoints of strokes. Further, we derived the keypoint feature descriptor from this set of points to describe wedges with a fixed-size vector. In this Chapter we will introduce the concept of a point-cloud, a set of two-dimensional points without any specific semantic meaning. To summarize, these three sets of points can differentiated as follows.

Keypoints Intersections between strokes and endpoints of strokes are extracted as in Chapter 3, Section 3.4.2, pp. 44. From this set of keypoints a graph of wedge constellations is constructed by a sweepline algorithm to recover the topology induced by the strokes, i.e. to represent strokes and their intersection using a graph.

Keypoint feature descriptors We assume that all strokes are describing wedge-shaped impressions created by a rectangular stylus. Therefore, we extract a fixed-size reduced set of points. These define the keypoint feature descriptor and three intersection points for the wedge-head three endpoints of adjacent strokes forming the wedge-arms. These six keypoints form a 12-dimensional vector.

Point-clouds By sampling the spline boundary of the initial set of strokes equidistantly without regard to any meaning of the strokes, we arrive at point-cloud, a set of points, that accurately and directly describes the exact appearance of the strokes. We use this method as a baseline to show the abstractive power and to make sure that no meaning is lost in our high-level description methods.

Figure 4.1 illustrates the difference between the three points which are computed using different means from the same set of strokes.



(a) (b) (c)

Figure 4.1: The same set of strokes and three different sets of points computed from it. a) Keypoints are computed by finding intersection points between strokes and endpoints of strokes. b) Keypoint feature descriptors are derived from the previous set. We take the three intersection points of a wedge-head forming a triangle and three endpoints of adjacent strokes forming the wedge-arms. The association of keypoints to a specific wedge is indicated by color. c) A point-cloud is computed from a set of stokes by equidistantly sampling the spline boundary of all strokes and deriving from it a set of points. No additional information, i.e. the original stroke they were derived from, is associated with these points beyond their two-dimensional position.

4.3 Related Work in Feature Description for Handwriting

The challenge of defining an accurate and computationally efficient distance between signs of any script is the basis of word-spotting systems if combined with word segmentation [RM07] or an efficient search strategy [LLE07].

In their work on historical word-spotting, Rath et al. [RM07] present an approach to similarity computation using the projection profiles of words. These are compared using Dynamic Time Warping (DTW) as shown in their earlier work [RM03]. Word images are segmented from the document using a scale space approach from the work of Manmatha et al. [MR05]. Four different features of word images are extracted, the upper profile, the lower profile, the count of background and ink transitions and the sum of foreground ink pixels.

Another approach is the work by Kennard et al. [KBS11] where segmented word images are aligned using a flexible grid. First, the authors perform a

coarse alignment of the grid using DTW. Then, the word images are thinned by the medial axis algorithm [Blu67]. The dissimilarity of the query word to the candidate word is computed by distance transforming [Bor86] of both images and summing the distance when the medial axis overlaps the query. Grid alignment is performed in alternating steps of deforming and subdivision of the grid. This saves computational resources and results only in an insignificant loss of precision.

The vertical complexity present in the slices used for HMM training requires features that are robust against noise and writing variations, but also retain spatial relationships between the information present in the slice. Fischer et al. [FRB10] choose to use graphs to model handwritten script and to slice those graphs into windows for learning a HMM. To transform graphs into feature vectors, a necessity for HMM, dissimilarity space embedding is used [RB10]. This technique uses a set of prototype vectors, against which a candidate graph is compared. During training the authors manually select a prototype graph for each letter in their Latin script. For graph comparison Fischer et al. use the graph edit distance [BA83]. Another simplification is that the graphs used for modeling slices have no edges. Computing the graph edit distance between those graphs is reduced to the assignment problem that can be efficiently solved using the Hungarian algorithm [Mun57].

Leydier et al. [LLE07] use gradient features to identify zones of interest in historical document images. Their approach to word-spotting requires no prior learning. The query word is decomposed into zones of interest that are matched against zones of interest detected in the document. Three approaches to matching are presented. The native approach aims to position all zones of interest of the query using a single translation vector. This method suffers from its rigidity with respect to the variability of human hand-writing. Their elastic-naive method of matching aligns each pixel in the query independently, but suffers from a lack of cohesion and high computational complexity. Finally, a hybrid approach independently aligns zones of interest instead of individual pixels.

4.4 Implicit Wedge Similarity

In this chapter we present methods for computing the similarity of wedge constellations in implicit representation as introduced in Chapter 3, Section 3.3,

pp. 34. This assumption of modeling wedge constellations does not assume any underlaying structure and simply models a set of strokes. The set of strokes can be arbitrarily arranged, not necessarily representing wedge-shaped impressions, and still be compared and analyzed by the presented methods.

4.4.1 Projection Profile

A standard approach for word-spotting in historical handwriting is the projection profile introduced by Rath et al. [RM07]. It makes use of word profile feature descriptors by Chen et al. [Che95] which exploit the fact that written Latin script expresses its salient information in writing direction. Thus, if we collapse the columns of a word raster image by means of a summarizing function and express a word image as a linear signal, we do not lose differentiating information and remove noise. Further, comparing linear signals for similarity is computationally more tractable than images, e.g. using the DTW algorithm to align two signals has only quadratic runtime complexity. Figure 4.2 illustrates a rasterized cuneiform sign from a set of strokes and its projection and transition profiles.

Let I denote a raster image with pixel intensities $I_{ij} \in [0, 1]$. We assume that word images are binary where 1 is the foreground color and 0 is the background color. Then, each of these feature descriptors operate on columns of this image yielding a summarizing value $p(I)_i \in \mathbb{R}$. We compute four different word profile feature descriptors presented in [Che95] by Chen et al., which we denote as p_{proj}, p_{trans}, p_{top} and p_{bot}. The projection profile is a sum of foreground pixels.

$$p_{\text{proj}}(I)_i = \sum_j I_{ij} \tag{4.5}$$

The transition profile counts the amount of transitions between foreground and background pixels in each column.

$$p_{\text{trans}}(I)_i = \sum_j \| I_{ij+1} - I_{ij} \| \tag{4.6}$$

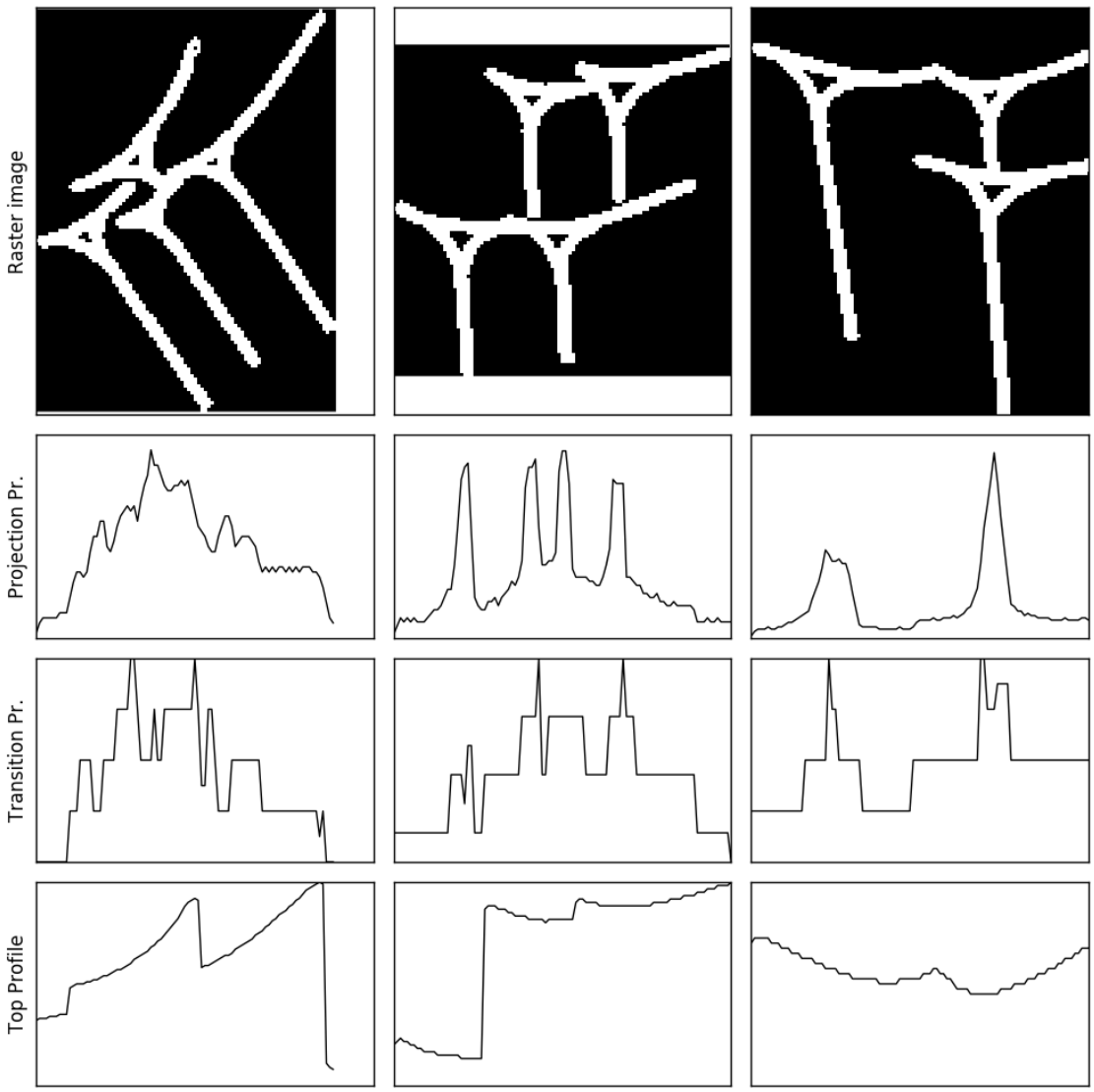


Figure 4.2: Three rasterized cuneiform signs with their projection profile, transition profile and top profile shown.

The top word profile and the bottom word profile counts the background pixels until reaching the first foreground pixel from the top or the bottom, respectively. It is an outline of the word as if looked upon from top or from below.

$$p_{\text{top}}(I)_i = \min\{j \mid I_{ij} = 1\} \tag{4.7}$$

$$p_{\text{bot}}(I)_i = \max\{j \mid I_{ij} = 1\} \tag{4.8}$$

From such a word image we derive a feature describing signal x by applying a profile feature descriptor for each column in turn. Applying more than one profile feature descriptor yields a multivariate signal where each element is a vector of the results of the applied profile feature descriptors.

$$x = \left[\begin{pmatrix} p_{\text{proj}}(I)_1 \\ p_{\text{trans}}(I)_1 \\ \vdots \end{pmatrix}, \begin{pmatrix} p_{\text{proj}}(I)_2 \\ p_{\text{trans}}(I)_2 \\ \vdots \end{pmatrix}, \cdots, \begin{pmatrix} p_{\text{proj}}(I)_i \\ p_{\text{trans}}(I)_i \\ \vdots \end{pmatrix} \right] \tag{4.9}$$

Two raster word images do not necessarily have the same width. The signal feature descriptors computed from these images are therefore of differing lengths. Signals of different lengths cannot be directly compared, e.g. by element-wise Euclidean distances. For this, we first align the signals from the two word raster images being compared so that their difference is minimized. The DTW algorithm aligns two sequences so that the sum of element-wise distances is minimized.

Dynamic Time Warping The DTW algorithm by Sakoe et al. [SC78] has seen first application in speech recognition where a signal of utterances has been aligned to a transcription. The algorithm uses a dynamic programming [Bel57] approach to find an alignment of two signals that minimizes their distance. Through the use of word profiles we model word images in documents as signals. Word similarity and word recognition is then transformed into a signal alignment challenge. Then, DTW is a similarity metric for word images.

For two discrete signals $x, y \subset \mathbb{R}$ we define a distance function $d(x_i, y_j) : \mathbb{R} \to \mathbb{R}$ with $d = \|x_i - y_j\|$ to measure the similarity between two points in the signals.

The alignment and cumulative distance $D(x, y)$ between these signals can then be recursively defined as follows.

$$\begin{aligned} D(1,1) &= d(x_1, y_1) \\ D(i,1) &= D(i-1,1) + d(x_i, y_1) \\ D(1,j) &= D(1,j-1) + d(x_1, y_j) \\ D(i,j) &= \min \begin{Bmatrix} D(i,j-1) \\ D(i-1,j) \\ D(i-1,j-1) \end{Bmatrix} + d(x_i, y_i) \end{aligned} \tag{4.10}$$

The cumulative distance D is a path of minimal cost in the similarity matrix of the signals being compared. Figure 4.3 shows two signals and the similarity matrix between those. The DTW distance function is efficiently computable with a $O(n^2)$ runtime complexity where n is the sum of lengths of the signals. Then, a distance function for two rasterized wedge constellations I_1 and I_2 are defined as follows.

$$d^{\mathrm{DTW}}(I_1, I_2) = D(p_{\text{proj, trans, top, bot}}(I_1), p_{\text{proj, trans, top, bot}}(I_2)) \tag{4.11}$$

To compare wedge constellations we rasterize the set of strokes representing each constellation into a raster image. We make use of the multivariate formulation of signal feature descriptor and compute all for profile feature descriptors for each column of each word raster image. This makes sure that all relevant vertical features are captured.

4.4.2 Graph of Keypoints

On basis of our implicit representation of wedge constellations we introduce three methods for comparing topological undirected graphs. We assume that changes in the graph connectivity indicate changes of meaning to the wedge constellation being described. Graph comparison can be achieved by methods, including the Weisfeiler-Lehman Graph Kernel framework [WL68; She+11], random-walk graph kernels [Vis+10], and the spectral decomposition [Chu97]. These methods exploit various properties of these graphs to approximate graph equality tests. Exact graph equality or similarity by means of the graph

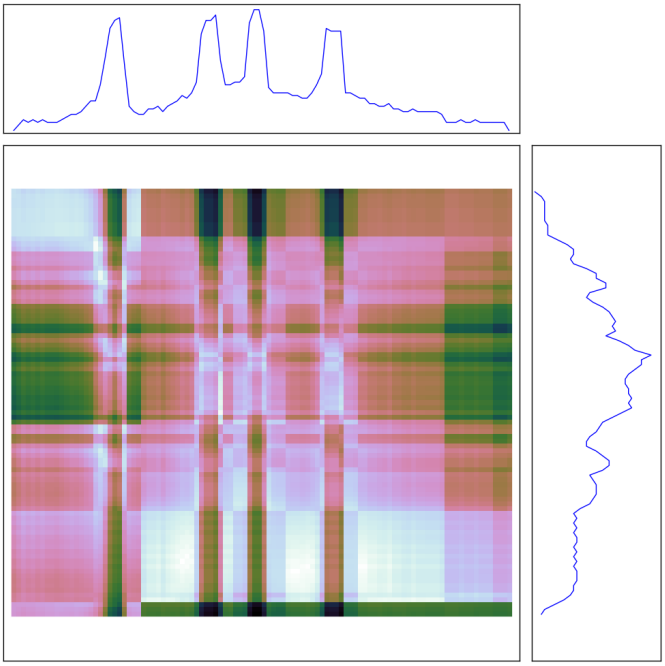


Figure 4.3: DTW matrix with exemplary projection profile of the previously shown cuneiform signs. Light color values denote high similarity in the respective areas and dark colors denote dissimilarity between the two signs.

edit distance [SF83; Gao+10] has combinatorial runtime complexity [GJ79]. The presented methods, however, require polynomial time to approximate similarity.

Weisfeiler-Lehman Graph Kernel Weisfeiler and Lehman [WL68] introduced a reduction of graphs to a canonical representation. Then, Shervashidze et al. expanded this approach to the concept of graph kernels [She+11]. The Weisfeiler-Lehman graph kernel framework provides us with a feature descriptors and a positive semi-definite kernel function.

The computation of graph kernels proceeds iteratively. Each vertex of a graph is assigned a, not necessarily unique, label. On every iteration of the algorithm each vertex label is expanded with labels of adjacent vertices. Those adjacent vertex labels have been, in turn, extended in an earlier iteration by their adjacent vertex labels. Thus, the label of each vertex is an enumeration of a subtree rooted at that vertex. Algorithm 6 gives a detailed procedure for the computation of the kernel and Figure 4.4 illustrates the procedure.

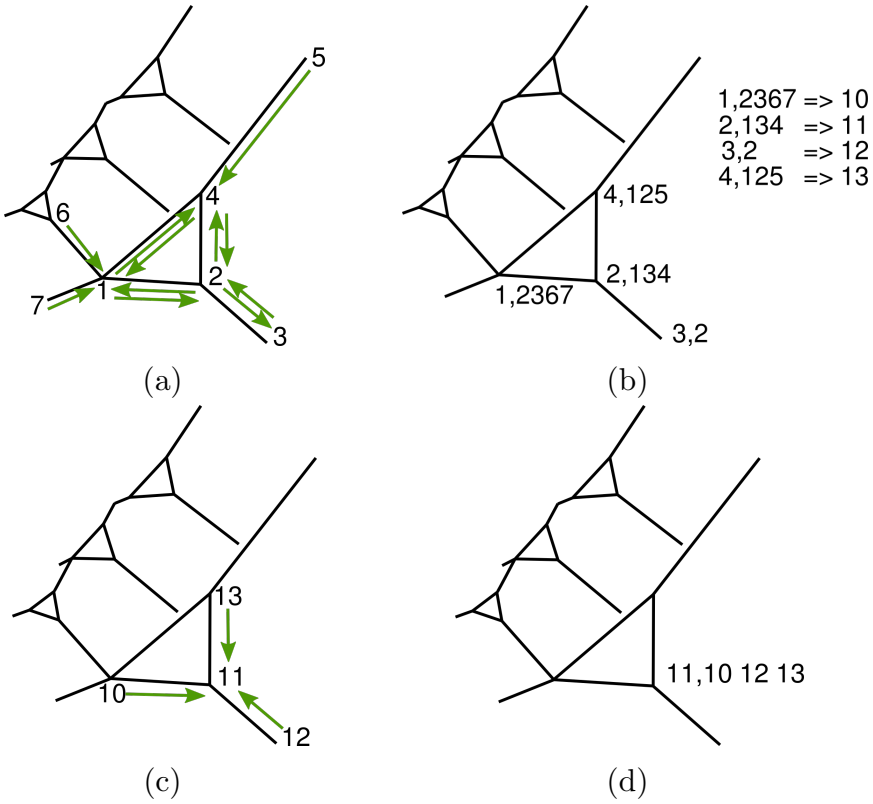


Figure 4.4: a, b) First iteration and b, c) second iteration of the Weisfeiler Lehman Kernel computation. a, c) Adjacent labels are first collected and then, b, d) vertices are relabeled. The arrows show the directions in which labels flow.

Let $n = |V|$ denote the count vertices and $m = |E|$ the count of edges in a graph. Finding adjacent vertices has a runtime complexity of $O(n)$, each vertex is checked for adjacency. This operation is performed for each vertex in the graph, k-times for the k re-labeling iterations. The runtime complexity of Algorithm 6 is $O(k * n * n) = O(k * n^2)$.

The subtree kernel is a direct application of the re-labeling iterations of the Weisfeiler Lehmann Graph Kernel Framework. In each iteration the re-labeling operation computes new labels. The subtree kernel compares the generated sets of two graphs and counts identical labels. This process is repeated for each iteration up to some iteration limit k and the counts are summed. Since the labels represent unique subtrees, the subtree kernel effectively counts identical subtrees between two graphs up to an depth of k. The kernel requires an a-priori labeling of the vertices. We label each vertex with a unique label.

Let L be the set of iteratively computed labels $L_0 \cdots L_k$ in k iterations. Then, the i-th component of the feature vector f of a graph is the count of times

Algorithm 6 Compute Weisfeiler-Lehman subtree labeling.

```
procedure LABELS(V, E, L, k)
    ▷ First iteration of labels is the initial
    ▷ labeling of vertices.
    L_0 ← L

    ▷ Expand labels k times
    for i ∈ {1 ··· k} do

        ▷ For each vertex find adjacent vertices.
        for v ∈ V do

            for w ∈ adjacent_E(v) do
                ▷ And expand the label of the current
                ▷ vertex with their labels from the
                ▷ last iteration.
                l_{i,v} ← l_{i,v} ∪ l_{i−1,w}
            end for

            ▷ Save hashes of labels for each iteration
            ▷ instead, otherwise they get exponentially
            ▷ long.
            l_{i,v} ← hash(l_{i,v})
        end for

    end for

    ▷ Return descriptor of graph as a set of labels
    ▷ generated from it. A similar graph will share
    ▷ many labels with this set. Graph comparison is
    ▷ then reduced to set comparison.
    return {L_0 ··· L_k}
end procedure
```

label l_i is present in the label set throughout the re-labeling iterations. That is, the count of times the subtree denoted by label l_i has been seen.

$$\begin{aligned} L &= \bigcup \{L_0 \cdots L_k\} \\ \{l_0, l_1, l_2, \cdots l_i, l_j, \cdots\} &\in L \\ f_i &= |\{l \in L \mid l = l_i\}| \\ f &= (\underbrace{|l_0|, |l_1|, |l_2|, \cdots}_{\in L_0} \underbrace{|l_i|, |l_j|, \cdots}_{\in L_{1 \cdots k}}) \end{aligned} \tag{4.12}$$

By fixing the iteration depth for two graphs G and G', we obtain equal length feature vectors that can be compared using the dot product.

$$k^{\text{Subtree}}(G, G') = \langle f_G, f_{G'} \rangle \tag{4.13}$$

The dot product in the kernel counts matching subtrees between two graphs as the vectors f_G and $f_{G'}$ are binary vectors denoting the presence or absence of a subtree.

Random Walk Graph Kernel The fundamental idea of this kernel is count shared walks, sequences of incident vertices starting and ending with the same vertex, between two graphs to compute their similarity. These walks are generated randomly on either of these graphs and both are checked for their presence. The similarity comparison becomes more accurate with increasing count of checked random walks. Figure 4.5 illustrates this principle on wedge constellations.

In their work [Vis+10] Vishwanathan et. al present an efficient means of computing the random walk graph kernel. They exploit a property of the direct graph product of two graphs. A random walk on the direct graph product of two graphs $G_\times = G \otimes G'$ is present in both of the original graphs. The direct graph product of two graphs can be computed by the Kronecker product of their adjacency matrices $A_{G_\times} = A_G \otimes A_{G'}$. Further, they make use of the property that the k-th power of an adjacency matrix $A^k_{G_\times}$ computes the count of walks of length k from vertex i to j in the matrix element A_{ij}.

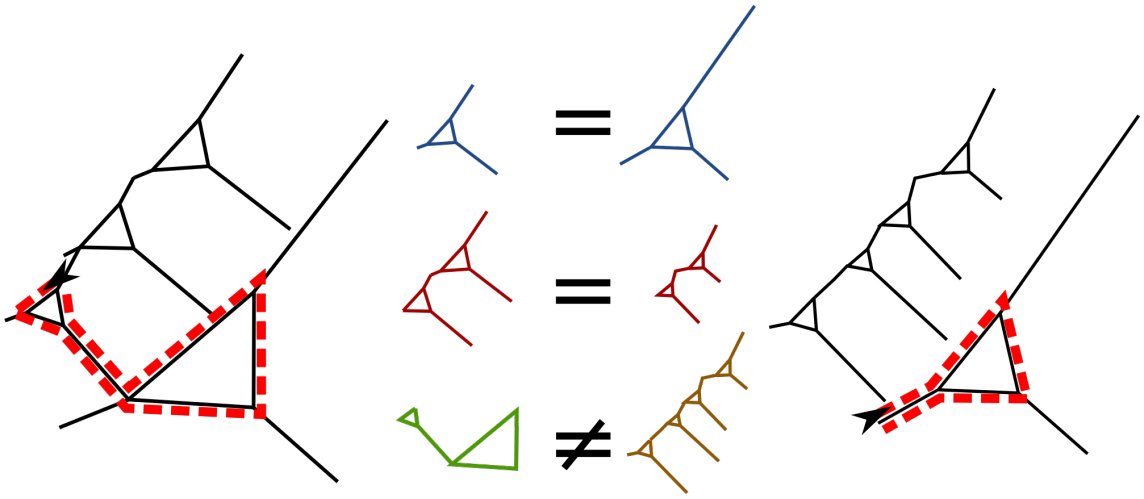


Figure 4.5: Two cuneiform signs represented as graphs. In each one an exemplary random walk has been illustrated by the red stippled path. The small graphs aside denote three random walks exemplary extracted from the respective sign. Two of those are topologically identical while the third ones are not. We consider graphs sharing a high count of topologically equal random walks to be similar.

Therefore, instead of naively generating random walks and checking their presence in both graphs, we make use of the properties of the adjacency matrices of undirected topological graphs, the kind which we use to represent wedge constellations. We iteratively compute the count of all shared walks up to a length of k. This approximation of the random walk graph kernel completely removes the component of randomness. We still call this approach the random walk graph kernel since we only approximate a random process by means of a deterministic method. Contrary to the naive random walk graph kernel, the approximation has a deterministic computational complexity. The Kronecker product of two matrices $A^{m\times m}$ and $B^{n\times n}$, representing two graphs with respectively m and n vertices, is $O(mn + mn) = O(mn)$ resulting in a matrix with $(m + n) \times (m + n)$ elements. Naive matrix multiplication has the complexity of $O(p^3)$ of two matrices with $p \times p$ elements. We compute lengths up to k by repetitive matrix multiplication. Therefore, the complexity of the random walk graph kernel of two cuneiform signs represented by two graphs with m and n vertices is $O(mn + k(m + n)^3) = O(km^3 + kn^3)$.

In our approach, we additionally scale those count by the path length k to weight longer paths more. Those are important for the graphs computed from cuneiform signs as short paths encode only wedges which are present in all signs. The approximation to the random walk graph kernel is computed as follows.

$$k^{\text{Randomwalk}}(G, G') = \sum_{k} (A_G \otimes A_{G'})^k * k \tag{4.14}$$

Contrary to the Weisfeiler Lehmann Graph Kernels, the Random Walk does not requires an a-priori labeling of graph nodes. We limit the count of iterations when computing random walks. More than 10 iterations did not improve the similarity metric.

Spectral Graph Kernel Here we introduce our third method for computing the similarity between graphs. Similar to the random walk graph kernel, it makes use of mathematical properties of the adjacency matrices of its graphs. The spectral decomposition into sequences of eigenvalues and eigenvectors of a graph has a variety of applications in the field of graph analysis [Chu97]. We make use of the property that the two similar graphs have similar sequences of eigenvectors and eigenvalues. Further, a small modification of the topology of a graph results in only a small change in its sequences of eigenvectors and eigenvalues. This property makes it well suited for a distance function.

We compute the spectral decomposition from the normalized Laplacian matrix L with components l_{uv} from the adjacency matrix A of a graph G with vertices u, v and vertex degrees d_u, d_v. L has the eigenvectors ϕ_i and the sequence of eigenvalues λ_i.

$$\begin{aligned} L =& [l_{uv}] \\ l_{uv} =& \begin{cases} 1, & \text{if } u = v \text{ and } d_u \neq 0 \\ -\frac{1}{\sqrt{d_u d_v}}, & \text{if } u \neq v \text{ and } a_{uv} = 1 \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{4.15}$$

$$\begin{aligned} \phi_i^T L \phi_i =& \lambda_i \\ & \lambda_1 \leq \cdots \leq \lambda_i \leq \cdots \leq \lambda_n \end{aligned} \tag{4.16}$$

For a real symmetric matrix such as the adjacency matrix of our graphs, the Jacobi method [GV00] computes the set of eigenvalues in $O(n^3)$ time where n denotes the count of vertices in the graph. We illustrate the results of

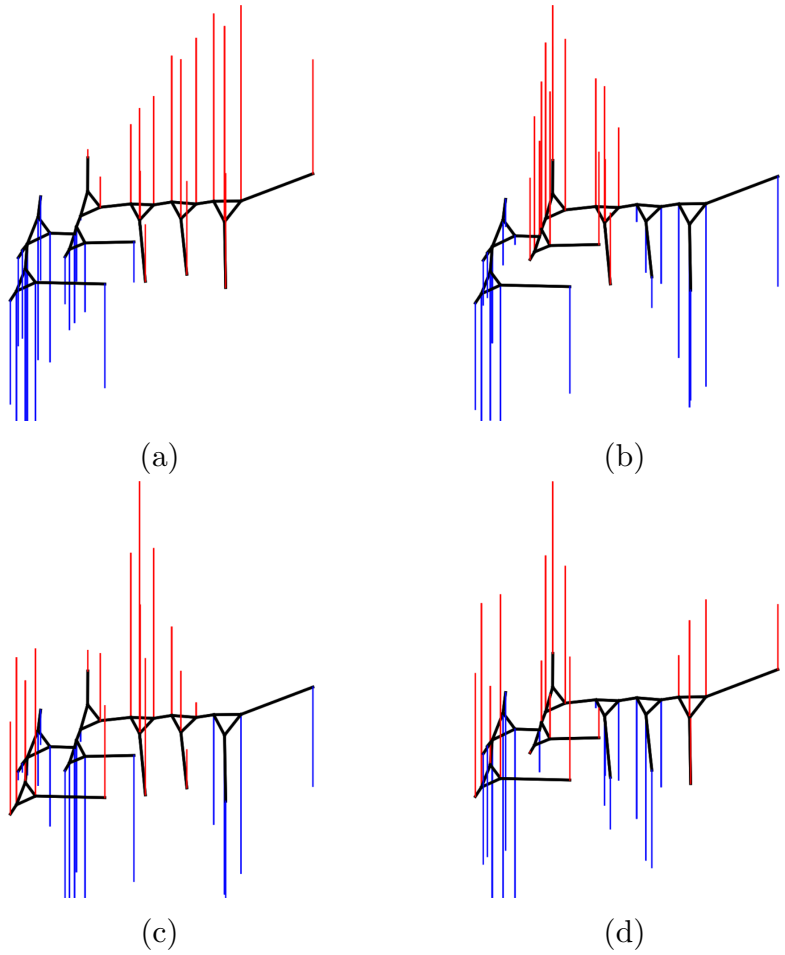


Figure 4.6: The first four eigenvectors of a graph representing a wedge constellation. Eigenvector components have visualized at their respective vertices. Line length visualizes component magnitude. Red lines visualize positive values and blue lines visualize negative values.

this computation here by mapping the eigenvector to the graph it has been computed from. The cardinality of the eigenvector matches the count of vertices. We attach each component of the eigenvector to its respective vertex in the graph and visualize its value in Figure 4.6.

The sequence of eigenvalues has the property that eigenvalues occurring at the beginning describe the topology globally, i.e. only significant changes in topology change their value, while eigenvalues at the end describe the topology locally, i.e. small changes in topology incur great changes in value. This allows us to compare graphs directly by their sequences of eigenvalues. The length of the sequence of eigenvalues is determined by the count of vertices in a graph. To ensure the sequences have the same length for comparison of two graphs, we shorten the longer sequence by dropping its values at end of its sequence.

As those only describe small and local modifications in topology we do so without loss of precision.

Typically, graphs are compared by embedding them into a feature space by their sequence of eigenvalues and measuring distances with the p-norm. However, we found that using the cosine distance [Sid+14], to measure the angle between the eigenvalues of both decomposed graphs, yields better classification performance. The similarity of two graphs G and G' can therefore be computed by measuring the angle between the feature vectors λ and λ', which are the sequences of eigenvalues of the respective graphs.

$$k^{\text{Spectral}} = \frac{\langle \lambda, \lambda' \rangle}{\|\lambda\| \|\lambda'\|} \tag{4.17}$$

Each of these three presented graph kernels compute distances purely based on the topology of the graphs. If the only difference between two wedge constellations represented by graphs are moved vertices, the presented methods are not able to detect such changes.

Delaunay triangulation To maintain the expressiveness of graph based representation while adding constraints to the movement of vertices, we perform a Delaunay triangulation [Del34]. We remove all edges from the graph and re-compute the topology by triangulating the vertices made of keypoints. Figure 4.7 illustrates a graph representing a wedge constellation and its triangulated counter-part. Then, the presented graph kernels are applied as-is for comparing wedge constellations. The process of comparison is then performed as follows.

1. Keypoints and edges are extracted from the initial set of stroke of the born-digital tracing of a cuneiform tablet.

2. When comparing two graphs of wedge constellations, edges are removed and re-computed by triangulation of the vertices.

3. One of the presented graph kernels is used on the triangulated vertices to compute a distance.

The triangulation of a set of n keypoints has a computational complexity of $O(n \log n)$ [Ber+08]. We evaluate all of the presented graph kernels with

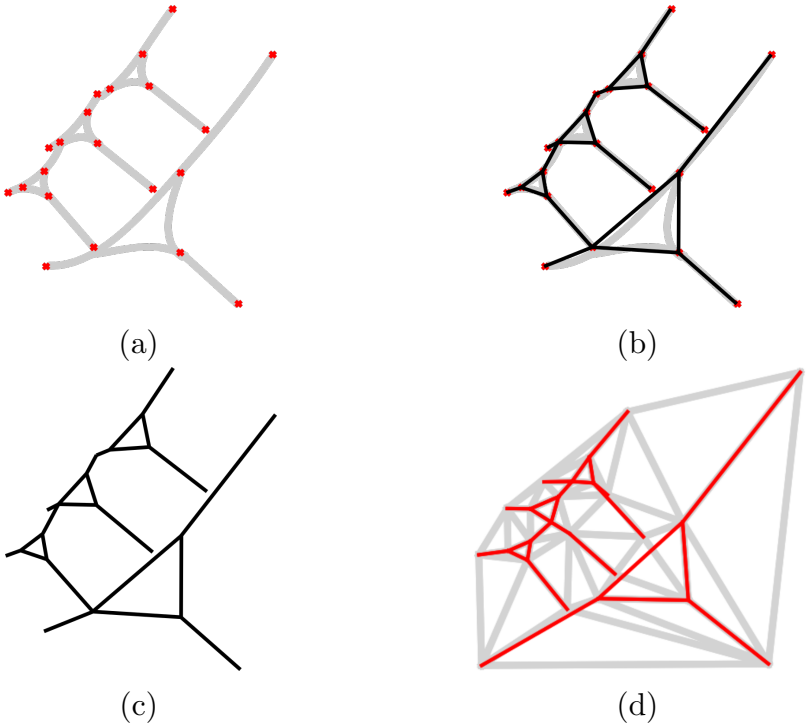


Figure 4.7: a) The keypoints of a set of strokes representing a wedge constellation, b) and c) the reconstructed graph and d) the Delaunay triangulation of the keypoints. The triangulation is shown in gray with the original recovered graph overlaid in red.

added triangulation to find the best performing combinations. The results are presented in Section 4.6, pp. 95 of this chapter.

4.4.3 Point Clouds from Splines

The presented graph representations model only the topology of graphs representing wedge constellations. Even the topology of the Delaunay triangulation does not change for small differences in keypoint position. For wedge constellations that are visually similar but carry different meaning, these small difference in position have to be modeled.

We differentiate three different sets of points in this work. Section 4.2.4, pp. 63 gives an overview. Here we extract points from the boundaries of strokes resulting in a point-cloud describing wedge constellations. While keypoints have a well-defined meaning for wedge constellations, points extracted from stroke boundaries are purely visual and are compared without any preference given to specific subsets. This approach discriminates between small changes

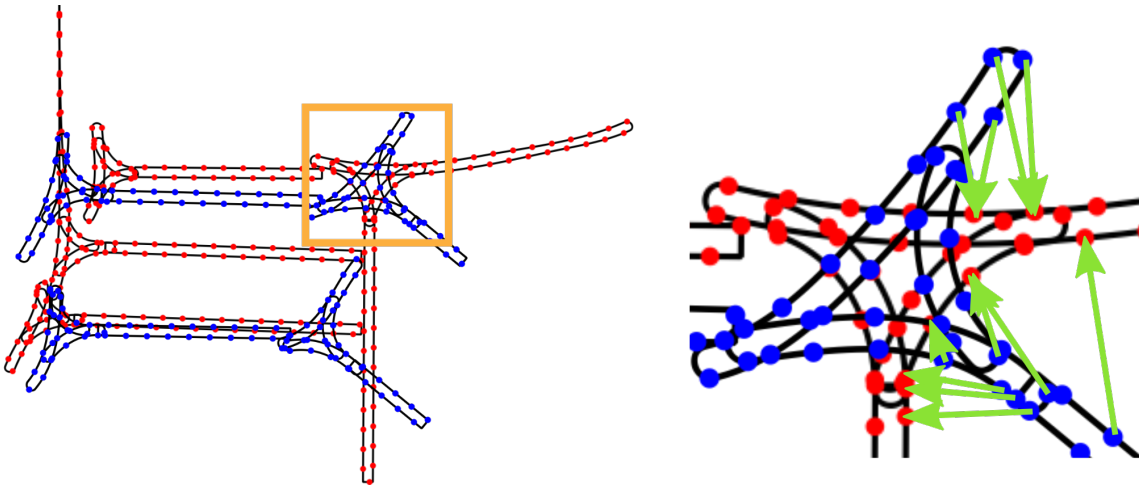


Figure 4.8: Two wedge constellations overlaid on top of their geometrical centers. Blue markers of the first point cloud are matched to the red markers of the second point cloud. The original strokes of the wedge constellations are shown but have no influence on the similarity computation. The Iterated Closest Points (ICP) algorithm performs this computation iteratively and adjusts each time the transformation to minimize distances.

in appearance and is unbiased by any definition of keypoints. Because of its simplicity and applicability to any set of stokes, this approach is a baseline in our evaluation that all other methods are compared to. Figure 4.8 visualizes the point-clouds extracted from two wedge constellations in two different poses and their closest points.

Let S be set of strokes of a wedge constellation and s a stroke in this set. Then, X is the point-cloud extracted from the boundaries of the strokes and x a two-dimensional point in the point-cloud.

$$X = \bigcup_{s \in S} \partial s \tag{4.18}$$

We introduce two methods to compare the resulting point-clouds. One computes the distances of the point-clouds as-is, and is in principle comparable to template matching of raster images, the other computes an affine transformation to optimize the matching.

Direct Distance If the two point-clouds being compared for similarity are in roughly the same pose, same rotation and same scaling, we can directly compute the distance between corresponding points. This is a reasonable assumption for modern cuneiform script, since line height and cuneiform sign size shows little variability across a single cuneiform tablet. For two point-clouds computed from two wedge constellations being compared, let X and X' denote their sets of points. We compute their distance by first centering the point-sets by aligning their centers of mass and then computing the minimal distances between points.

$$\begin{aligned} \tilde{X} &= \text{mean}(X) \\ \tilde{X}' &= \text{mean}(X') \\ d^{\text{Direct}}(X, X') &= \sum_{x \in \tilde{X}} \min_{x' \in \tilde{X}'} \|x - x'\|^2 \end{aligned} \tag{4.19}$$

Besl et al. [BM92] showed that computing the minimal distance between points approximates the distance computation corresponding points. We assume that for most points, the nearest point from the second point-cloud is also its corresponding point, in our case, a point on the boundary of the same stroke carrying the same meaning. Since the distance of every pair of points has to be computed, this approach has a computational complexity of $O(n^2)$ where n is the count of points.

Iterated Closest Points If the poses of the two point-clouds are not close, Besl et al. [BM92] presented the ICP method that iteratively converges on a transformation that positions point-clouds to minimize the squared distances of their corresponding points. In each iteration the currently closest points are matched and an error function between those is minimized [Hor87].

While an optimal solution is not guaranteed, it is possible the algorithm becomes stuck in a local optimum, the closer the initial poses are, the more likely a globally optimal solution is. Additionally, we restrict the transformation to translation, rotation and limited scaling, as shearing and scaling are not transformations which preserve meaning for cuneiform signs. Significantly smaller or bigger wedge-shaped impression carry different meaning, sheared wedges are not possible because of the constraints of impressing a rectangular stylus. RANSAC [FB80] is an another method for finding an alignment between two point-clouds that provides better stability than ICP. We choose

not to use RANSAC since the initial poses of two wedge constellations being compared are very close, i.e. rotation more than a few degrees implies a different cuneiform sign, and we perform only 10 iterations of ICP to correct slant present handwritten script.

The ICP algorithm (and its optimized variant FastICP by Jost et al. [JH02]) approximates an affine transformation matrix T of a point cloud X that minimizes the distance d^{ICP} to another point cloud X'.

$$d^{\mathrm{ICP}}(X, X') = \arg\min_{T} d^{P}(XT, X') \tag{4.20}$$

Similar to the direct distance computation, the ICP algorithm computes all distance pairs and iterates up to k times. Its runtime complexity is therefore $O(kn^2)$ with n points and a fixed count of iterations k. The three presented approaches of representing and comparing wedge constellations made no assumption about the structure. Rasterization of graphs and point-clouds does not require any specific arrangement of strokes to function. However, we know a-priori that wedge-shaped impressions are traced to look like triangles due to the usage of a rectangular stylus. Cutting an edge of a cube results in a triangular face. In the following section, we make use of this assumption and compare wedge constellations directly by their feature representation introduced in Chapter 3, Section 3.4, on pp. 43.

4.5 Structured Similarity

We extend the previously introduced unstructured approach to wedge constellation similarity by introducing hierarchical structure. The topology of keypoints in wedges-shaped impressions is directly modeled by six keypoints, three for the wedge-head and three for the endpoints of the wedge-arms. The wedge-head forms a triangle and the wedge-arms are vertices connected by an edge to the closest vertices of the wedge-head. The set of wedges is modeled as a set of wedge-head centers, no connectivity to neighboring wedges is preserved. Figure 4.9 illustrates the modeling hierarchy of keypoints describing wedges and wedges describing cuneiform signs.

We compare wedge constellations by first aligning wedges and summing the geometric distance of their centers, and then by computing the feature distances

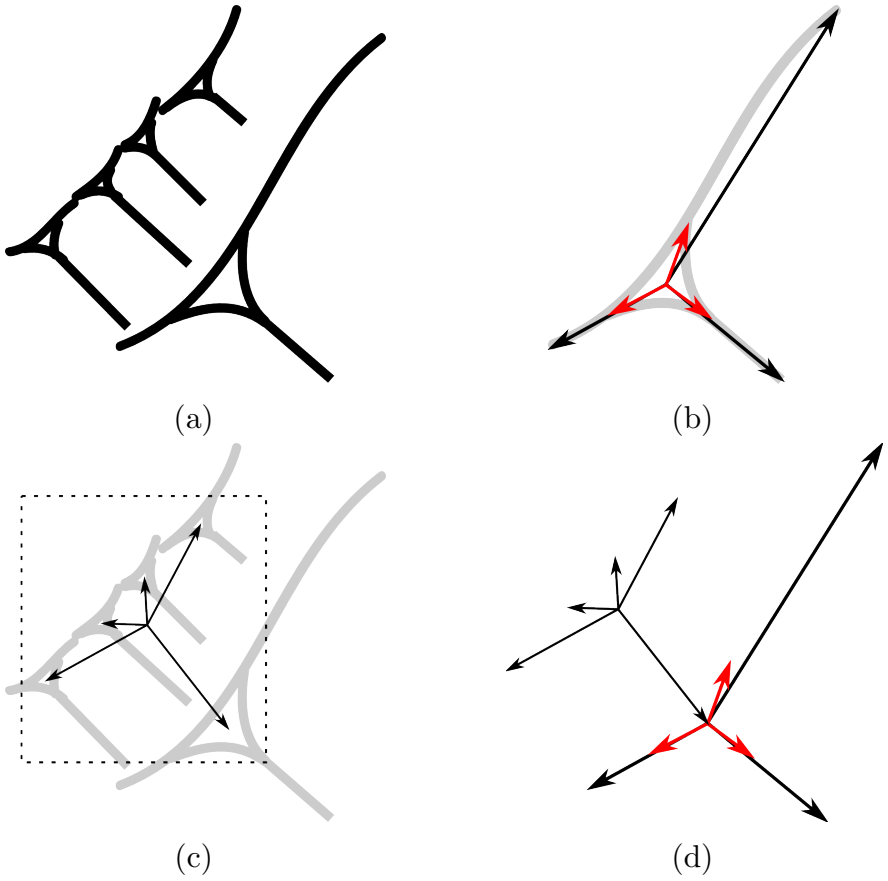


Figure 4.9: a) A wedge constellation as a set of strokes. b) Keypoint vectors from the center of the wedge-head. Vectors in red point toward wedge-head corners. Vectors in black point toward wedge-arm corners. c) The bounding box around all wedge-head centers. Black vectors from the center of the wedge constellation point toward wedge centers. d) The two stage hierarchical model. Vectors from the center of the wedge constellation to wedge centers with one of the wedges exemplary shown as vectors only.

between individual wedges. The two wedge constellations are first embedded into the same coordinate system. The centers of their bounding boxes, spanning the wedge-head vertices only, are overlaid on top of each other. Then the similarity of the wedges themselves is computed individually. While the distances of the coarse alignment of wedges is always computed using the Euclidean distance, the feature distance between individual wedges is generic. In the following we present three methods for comparing wedges by defining distance functions.

We assume no specific order in which wedges in a wedge constellation are written or read. It is therefore necessary to find an assignment of wedges between the two wedge constellations being compared. We do not assign wedges to their closest corresponding wedges, as used in the ICP method, as this may result in two wedges being assigned to one. As wedges are objects carrying complex meaning and the count of wedges in a wedge constellation is small (less than 30), we require each wedge to have at most one other corresponding wedge. That is, we require the assignment function to produce an injective mapping from one wedge constellation to another wedge constellation being compared. We introduce two methods for computing injective assignment functions. The first naive function assigns wedges by their order on the horizontal axis. While computationally very efficient, quasilinear due to sorting, it performs poorly in the presence of wedges stacked on the vertical axis. We solve this challenge in our second approach by solving an linear assignment problem that minimizes the cumulative offset induced by the assignment.

Assignment by horizontal order In our first approach we make use of the fact that cuneiform signs are slightly wider than tall. The difference in width and height is nonetheless much less pronounced than in Latin script. We order wedges by their position on the x-axis from left to right. The position of a wedge is the geometric center of the triangle representing the wedge. The edges of the arms do not influence the ordering of the wedges. Wedges of wedge constellations being compared are then compared individually by ordering. While computationally efficient, this approach fails at ordering wedges consistently between wedge constellations where wedges are vertically in line as shown in Figure 4.10. The illustration of ordered assignment in Figure 4.10 shows a wrong assignment of wedges between two wedge constellations that results in unnecessarily high assignment distances. The optimized assignment in Figure 4.10 is the expected assignment that minimizes distances between assigned wedges. The following method achieves the expected assignment.

Figure 4.10: Comparison of two different assignment approaches for computing the similarity distance of wedge constellations. First, ordered assignment, wedges are ordered by the position of their wedge-heads on the x-axis. Colored dots show the centers of the wedge-heads and the numbers denote their order on the x-axis. Incorrect assignments are denoted by red arrows. Second, optimizing assignment, wedges are assigned by minimizing distances of wedge pairs. The wedge-heads have no inherent ordering, minimal distances are shown with orange bars.

Assignment by optimal order In our second approach we formulate the challenge of matching wedges as an assignment problem with an assignment cost to be minimized. We compute all possible wedge assignments and choose the assignment that minimizes the distances. The advantage of this approach is that it allows wedges that are not in the same left-to-right order to still be assigned to each other, on the condition, however, that the difference in position and shape is minimal.

In theory, the Hungarian [Mun57] algorithm has better runtime complexity than a brute-force approach, i.e. polynomial complexity versus combinatorial complexity. In practice however, for small problem sizes of less than 6 wedges, constant costs like initialization of data-structures and function call costs, dominate the execution time. For comparisons where one wedge constellation has less than 6 wedges, we brute-force all possible permutations and compute assignment costs for all possible assignments. For comparisons with 6 wedges or more we use the Hungarian algorithm.

We evaluate the performance of our hybrid approach on a test dataset of increasing assignment matrix size. The matrix size is dependent on the wedge counts n, m of the two signs being compared $D^{n \times m}$. Figure 4.11 shows the runtime performance of both approaches with respect to the matrix size. We see that only for wedge counts greater than $D^{6 \times 6}$ the optimizing approach is faster, before that, the constant costs dominate its runtime. Since most wedges have less than 6 wedges, Figure 4.12 shows a histogram of wedge counts of manually segmented sign in our dataset, this optimization reduces the overall runtime of our pipeline.

The distance between two wedge constellations is computed as shown in Algorithm 7. We use the notation $[C]$ to denote all permutations of wedges in a wedge constellation C. For a set of all possible permutations P we denote a specific permutation $p \in P$ with $[C](p)$, e.g. $[\{1, 2, 3\}](1) = \{3, 1, 2\}$. If the count of wedges between two wedge constellations is unequal, for each missing wedge we add a penalty equal to highest assignment cost. Otherwise very small and zero length wedge constellation would always be very similar to any other wedge constellation.

For an asymptotic complexity analysis of Algorithm 7 the brute-force computation is effectively constant, it is only used for wedge counts $n = |C|$ and $m = |C'|$ less then $n < 6$ or $m < 6$. Solving an assignment problem with $n + m$ elements has a runtime complexity of $O(n^3 + m^3)$. Penalty computation depends on the difference of wedge counts $|n - m|$ thus at most $\max\{n, m\}$ and

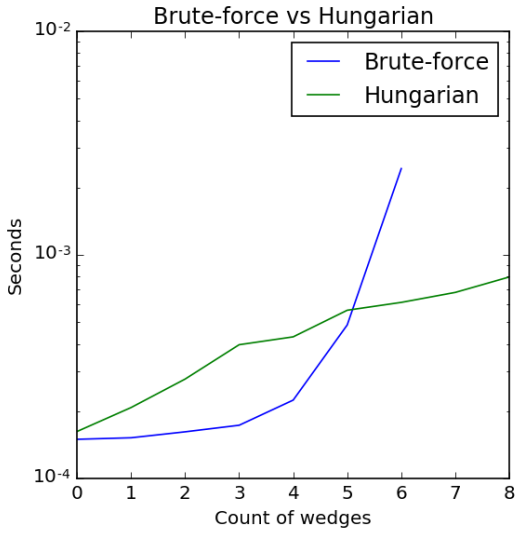


Figure 4.11: Runtime in seconds versus data size of two assignment approaches. A brute-force approach and linear optimization approach are compared. For very small dataset sizes the constant runtime costs dominate the performance of the linear optimization.

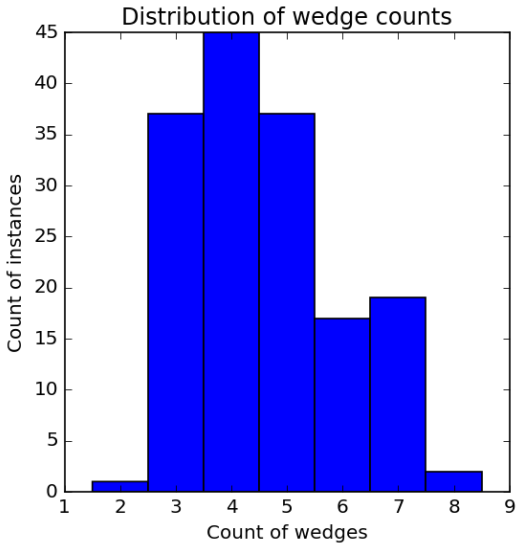


Figure 4.12: A histogram of wedge count sizes of our manually segmented dataset. Most cuneiform signs in our dataset have only few wedges.

the size of the distance matrix $n * m$. The algorithm has a runtime complexity of $O(n^3 + m^3 + \max\{n, m\} * n * m) = O(n^3 + m^3)$.

Algorithm 7 Compute the distance between two wedge constellations given a wedge distance function.

procedure DISTANCE(d, C, C')
  ▷ Compute distance matrix between each pair
  ▷ of wedges given a distance function d.
  $D \leftarrow \{d(c, c') \mid c \in C \wedge c' \in C'\}$

  ▷ Choose either optimal or brute-force
  ▷ approach depending on count of wedges.
  if $|C| > 5 \wedge |C'| > 5$ **then**
    $a \leftarrow \sum \text{solve\_assignment}(D)$
  else
    ▷ Iterate through all possible assignments
    ▷ of wedges between wedge constellations
    ▷ and choose assignment with lowest distance.
    $a \leftarrow \min_{p \in P} \sum D_{[C](p), C'}$
  end if

  ▷ Add a penalty equal to the highest assignment
  ▷ distance of the current distance matrix times
  ▷ missing wedges.
  $a \leftarrow a + ||C| - |C'|| \max D$

  ▷ Return cumulative distance and penalty of this
  ▷ assignment. This is the distance between these
  ▷ two wedge constellations.
  return a
end procedure

4.5.1 Wedge Distance by Keypoints

Our first structured approach to wedge similarity is based on the keypoints model derived in Chapter 3, Section 3.4.1, pp. 43. These keypoints are extracted from intersection points and endpoints of strokes and collected by pattern-matching in fixed-size feature vector.

For the keypoint model of wedge similarity we directly measure the Euclidean distance between the keypoints of two wedges. Each wedge is represented by a high-dimensional feature-vector. It consists of three intersection points forming the vertices of the wedge-head triangle and the three endpoints of the wedge-arms. Similar to the ordering challenge of wedges when comparing two wedge constellations, we require a unique assignment of keypoints between two wedges to minimize their distance. Since keypoints from wedge-heads and keypoints from wedge-arms carry different meaning, c.f the Winkelhaken described in Chapter 2, Section 2.1, pp. 19, we compare these individually. The relevant keypoints are shown illustratively in Figure 4.13.

Solving an optimal assignment problem using the Hungarian algorithm for each wedge has only polynomial complexity but is due practical considerations, e.g. setup costs and function call costs, not feasible. Nevertheless, we observe that in any comparison of wedges there are always three keypoints being compared to other three keypoints. The count of necessary comparisons for an optimal assignment is always six. Therefore, we brute-force the assignment possibilities and chose the optimal. Even though this approach is exponential in theoretical runtime complexity for an arbitrary count of keypoints, for our use-case, the real-world performance is significantly better than an optimal algorithm and the theoretical complexity with fixed input size is constant, i.e. $O(1)$.

Let $f^K, f'^K \in \mathbb{R}^{12}$ be the wedge feature descriptors of two wedges and let $(h, a) = f^K$ and $(h', a') = f^K$ denote the wedge-head and wedge-arms parts. We use $h_{\{1,2,3\}}$ to denote the set of vertices of the triangular wedge-head and $a_{\{1,2,3\}}$ the vertices of the endpoints of the wedge-arms. The notation $[(1, 2, 3)]$ denotes all permutation of the tuple $(1, 2, 3)$:

$$[(1, 2, 3)] = \{(1, 2, 3), (3, 1, 2), (3, 2, 1), \cdots\} \tag{4.21}$$

We use an index p to inspect a specific permutation $[(1, 2, 3)](1) = (3, 1, 2)$ and an index i to inspect a specific element of that permutation $[(1, 2, 3)](1)_0 = 3$. We combine this notation to iterate all orderings of wedge-head vertices $h_{[(1,2,3)]}$ all orderings of endpoints in wedge-arms $a_{[(1,2,3)]}$.

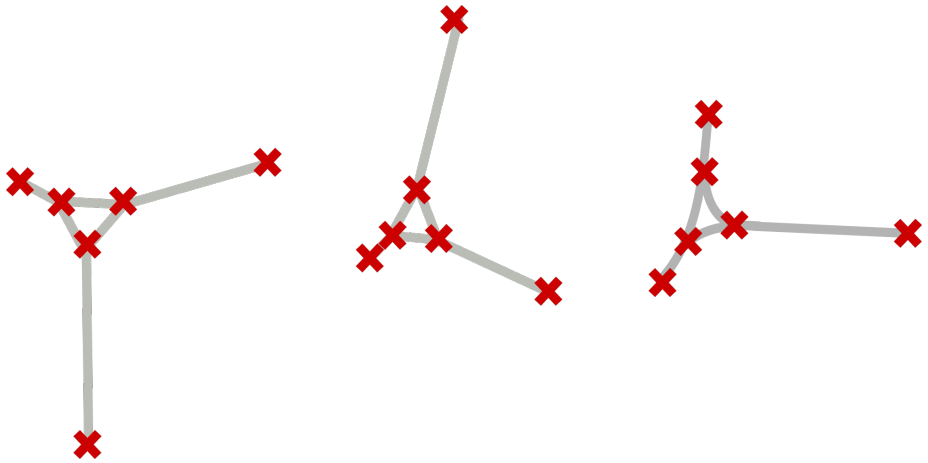


Figure 4.13: Two cuneiform wedges in Keypoint Model representation. Connection between the wedge-head vertices and the endpoints of the wedge-arms are shown as gray lines. The keypoints are shown as red markers.

$$
\begin{aligned}
d^K(\{h,a\},\{h,a\}) = \min_k \sum_{i\in\{1,2,3\}} \|h_{[(1,2,3)](k)_i} - h_i\| \\
+ \min_k \sum_{i\in\{1,2,3\}} \|a_{[(1,2,3)](k)_i} - a_i\|
\end{aligned}
\tag{4.22}
$$

We compare all orderings of vertices of a wedge to the vertices of another wedge and choose the ordering which results in the smallest distances, effectively brute-forcing an optimal assignment. With this method we are independent of the ordering of vertices in feature descriptor of a wedge. And with the optimal assignment of wedge constellations shown in the previous section, we are also independent of the ordering of wedges in wedge constellations. The comparison thus proceeds in a structured fashion. 1) Wedges between two wedge constellations being compared are assigned to minimize distance between centers, 2) vertices of the wedge-heads are assigned to minimize distance, 3) vertices of the wedge-arms are assigned to minimize distance. This comparison scheme ensures that the results are stable and semantically meaningful measures of similarity.

4.5.2 Bag-of-properties Model

Here, we introduce a feature description of wedges that models the presence or absence of specific properties of a wedge. While the previous structured and

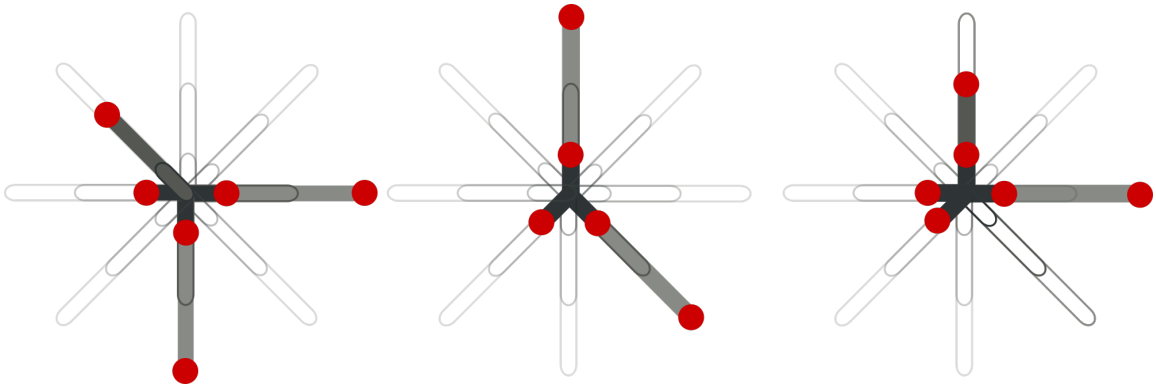


Figure 4.14: Original wedges as sets of strokes and properties set for bag-of-properties model. Dark shades with red dots denote properties set the wedge-head strokes. Lighter shades with red dots denote properties set by each of the wedge-arm strokes. If wedge-head and wedge-arm strokes match the same property, it is set only once.

unstructured representations directly and accurately described the geometric shape of a wedge-shaped impression, the exact angle and length of a wedge is not necessary to decipher its meaning. We simplify the range of possible property expressions (angles and lengths) by discretization. We discretize lengths into three ranges (short, medium, long) and angles into 8 directions. Then, the presence or absence of a property, e.g. a long wedge-arm down-left, is marked with a bit in a feature-vector. The advantage of this representation is that the ordering of properties is fixed for all wedges and no additional step aligning, as shown in the previous section, is necessary. Figure 4.14 illustrates the bits in the property vector that are set for a wedge-shaped impression.

Let $\hat{t}^P$ be a property vector and $\hat{t}^P_k$ the k-th component of that vector. We compare vectors of the keypoint feature descriptor to the property vectors to determine which features a wedge-shaped impression posses. The respective bit for the aforementioned property is then set in the property feature descriptor. The properties are enumerated by the set of angles $\{\alpha = 45°n \mid 0 \leq n < 8 \wedge n \in \mathbb{N}\}$ and by empirically set property vector lengths $\{l \in 1, 5, 10\}$. Using this discretization our property feature descriptors are 24-dimensional.

$$\hat{t}^P_{1\cdots24} = \left\{ \begin{pmatrix} \sin\alpha \\ \cos\alpha \end{pmatrix} l \right\} \tag{4.23}$$

Then, f^P is a binary vector defined as follows. Let W and W' denote the sets of paths representing two wedges. We base this feature representation on the keypoint feature descriptor. That is, we compare each of the six vectors in the keypoint feature to the property vectors $\hat{t}^P$ and choose the template vector $\hat{t}^P$ that is closest to the respective vector. Let $v \in f^K$ be keypoint vector from the center of the wedge-head in the keypoint feature descriptor of a wedge W.

$$
\begin{aligned}
f_k^P &= \begin{cases} 1 & \text{if } k \in \{\arg\max_i \|\hat{t}_i^P - v\| \mid v \in f^K\}, \\ 0 & \text{otherwise} \end{cases} \\
f^P &= \begin{pmatrix} f_1^P \\ f_2^P \\ \vdots \\ f_2^P 4 \end{pmatrix}
\end{aligned}
\tag{4.24}
$$

Given a binary feature descriptor of properties, we define a distance function between two wedges W and W'.

$$
d^P(W, W') = \|f^P - f'^P\| \tag{4.25}
$$

The resulting binary feature vectors are compared using the Euclidean distance. Testing with the cosine distance yielded worse results. The properties of this model have been empirically constructed by discretization of the space of possible expressions of wedge-heads and wedge-arms. The actual space of possible wedge expression is smaller, in the following section we infer typical wedge shapes and arrive at a significantly shorter, 5 dimensional instead of 24 dimensional, feature vector representation.

4.5.3 Gaussian Mixture Model

Cuneiform is written using a limited set of prototypical wedges. Experts actually only recognize three different types [Bor04]. We imagine written wedges as imperfect manifestations of these known types. To account for the natural variance of human handwriting and the damage of the cuneiform

tablets, we model the keypoints of wedge constellation with a mixture of high-dimensional Gaussian distributions.

If we want to model the imperfect manifestations of some value and assume that it follows a Gaussian distribution, we denote this as a random variable $\tilde{X}$ following the Gaussian distribution $\tilde{X} \sim \mathcal{N}(\mu, \sigma^2)$ with mean μ and variance σ^2. We can fit the parameters μ, σ^2 by means of maximum likelihood estimation by computing the mean and the standard deviation of a dataset X. To align our notation with literature in probability theory, we denote a dataset as X with samples $x \in X$. A single Gaussian distribution can be used to model a unimodal variable. If the underlaying process is inherently multi-modal, a mixture of many Gaussian distributions is used for modeling. In that case, we assume that a dataset is best approximated by a sum of multiple Gaussian distributions with the means μ_i, variances σ_i^2 and additionally the weights for each Gaussian θ_i. This modeling approach is then extended by modeling high-dimensional multi-model features. We imagine these visually as ellipses encircling dense clusters of datapoints in feature-space as illustrated by Figure 4.15.

Let μ_i be randomly initialized prior means and σ_i^2 prior variances of the mixture component i of k Gaussian mixtures. We denote the parameters θ as follows.

$$\theta_{i=1\ldots k} = (\mu_{i=1\ldots k}, \sigma^2_{i=1\ldots k}) \tag{4.26}$$

We use covariance matrices Σ_i for each component in a multi-variate setting to model dependencies between the dimensions of the samples. Then, the posterior distribution $p(\theta \mid x)$, given the feature vectors $X \subset \mathbb{R}^n$ and covariance matrices Σ_i, is formulated as follows.

$$p(\theta \mid X) = \sum_{i=1}^{k} \mathcal{N}(\mu_i, \Sigma_i) \tag{4.27}$$

While a single distribution can be estimated by computing the mean and standard deviation, for multi-model many-dimensional data we have to employ an iterative approach of estimation. The posterior distribution is estimated using the Expectation Maximization (EM) method, introduced by Dempster et al. [DLR77]. The EM algorithm finds a Maximum Likelihood Estimation

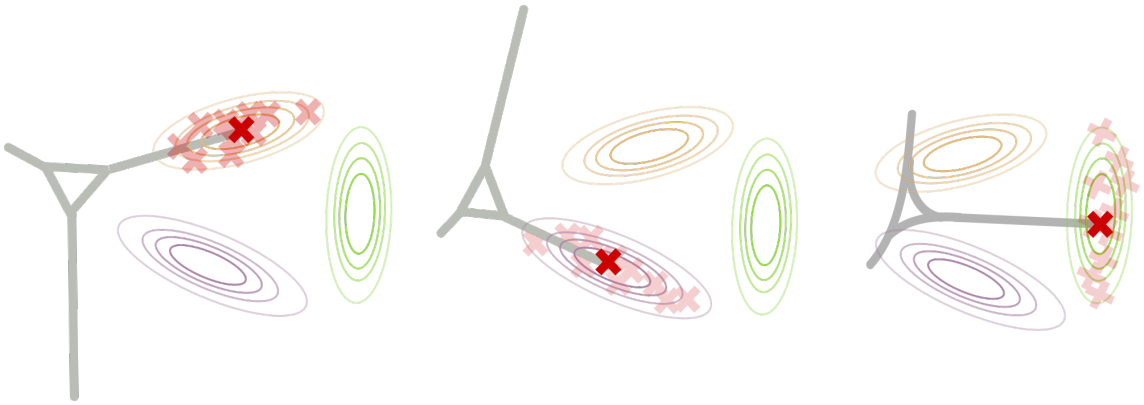


Figure 4.15: Gray lines denotes keypoint model from which distributions are derived. A mixture of Gaussian distributions (illustrated as orange and purple contour lines) is estimated for all keypoints (red crosses) of the keypoint model.

(MLE) of the parameters of a statistical model where there are unobserved, latent, variables. It proceed in two steps, the expectation step and the maximization step. The initial estimates for the two unknown sets, the latent variables and the parameters, can be randomly initialized. Then, the likelihood of the two sets are maximized separately in turn, once in the expectation step and once in the maximization step, given the recent inaccurate estimate of the other set. This procedure continues until convergence. The latent variables in our case are the assignments of feature vectors to the respective components denoted by $z_{i=1\ldots k} \in Z$. We approximate the component parameters best explaining our data.

$$\hat{\theta} = \arg\max_{\theta} p(\theta \mid X, Z) \tag{4.28}$$

Then, the probability of a feature vector $x \in X$ being part of component i is computed as follows.

$$p(x \mid \theta, z_i) = \mathcal{N}(x; \mu_i, \sigma_i^2) \tag{4.29}$$

After introducing the basics of modeling with Gaussian mixtures, we now derive wedge feature descriptors from this model. To align our notation with literature in probability theory, we denoted a dataset as X with samples $x \in X$. Now, we revert to notation used in previous chapters, i.e. F^k is the set of keypoint feature descriptors, the dataset X in previous notation, with

f^K being a keypoint feature descriptor for a single wedge, a sample x in the previous notation. We denote the Gaussian feature descriptors based on our GMM with F^T for the set of all features and with $f^T \in F^T$ for one feature descriptor of a specific wedge. Similar to the bag-of-properties model, we derive the Gaussian feature descriptors from a set of keypoint feature descriptors. Therefore, we write $f^T(f^K)$ to denote that f^T is computed on basis of f^K. To estimate the types of wedges used to represent the Gaussian wedge feature vectors, we maximize the parameters θ for 5 components $\{z_1, \cdots, z_5\} \in Z$ given the feature descriptors F^T.

$$\hat{\theta} = \arg\max_\theta p(\theta \mid F^K, Z) \tag{4.30}$$

We construct a Gaussian wedge feature vector f^T by estimating the posteriori probability of it being a manifestation of one of the learned prototypical wedges, the mixture components. We use this vector of probabilities as a feature-vector and measure similarity between wedge manifestations f^T and f^T with the euclidean distance. This comparison scheme ensures a smooth similarity function, as opposed to using the class affinity as a binary classifier.

$$\begin{aligned} f^T(f^K) &= \begin{pmatrix} p(f^K \mid \theta, z_1) \\ \vdots \\ p(f^K \mid \theta, z_k) \end{pmatrix} \\ d^G(f^K, f^K) &= \|f^T(f^K) - f^T(f^K)\| \end{aligned} \tag{4.31}$$

Contrary to all previous wedge modeling approach presented in this chapter, this approach requires a priori estimation of parameters $\theta_{i=1\ldots k}$ before feature vectors can be constructed. It cannot be used directly if no cuneiform wedges have been seen before. The parameter estimation is performed on the whole dataset F^K, i.e. all wedges extracted and represented by means of the keypoint feature vectors. Then, Gaussian feature vectors are computed for the same set of wedges. Thus, we are performing a clustering of the keypoint feature vectors of vectors to discretize their possible expressions, i.e. wedge types. Cuneiform wedge constellations are then described by the positions and type affiliations of its wedges instead of the exact geometric shape. The process of computing Gaussian wedge feature vectors and representing wedge constellations proceeds as follows.

1. Estimate Z by maximizing $p(\theta \mid F^K, Z)$ with respect to θ given the keypoint feature descriptors.

2. Derive new Gaussian feature vectors f^T from f^K by computing the affiliations to the estimated wedge types.

3. Represent wedge constellations by means of f^T and measure distances between wedges with the Euclidean distance between Gaussian feature vectors with the distance function d^G.

This method concludes our approaches to modeling distance functions for wedges and wedge constellations. In the following we evaluate the presented approaches to wedge and cuneiform similarity. Unstructured and structured approaches are compared and evaluated to the current state-of-the-art in handwritten Latin text retrieval.

4.6 Evaluation of Wedge Distance Metrics

At the beginning of this chapter we introduced the concept of a usable distance metric, one that is robust and discriminatory, c.f. Section 4.2.2 pp. 61. Here, we evaluate the representations and distance metrics presented in this chapter with respect to a ground truth. We manually segmented a dataset of cuneiform signs and labeled each cuneiform sign with its respective meaning in the Assyrian language. We expect a good distance metric to compute a lower distance for cuneiform signs that are labeled with the same label and a higher distance for cuneiform signs with different labels. The exact numerical values are not important as long as they are clearly separable for equally labeled and differently labeled cuneiform signs.

The evaluation task proceeds by comparing a query cuneiform sign against all other segmented cuneiform signs in our dataset. We expect the most similar cuneiform signs to have the same label as the query. That is, the optimal retrieval result is sequence of cuneiform signs of descending similarity where the first part are only cuneiform signs with the same label as the query followed by cuneiform signs having other labels. We say that such a result has a high precision. If the labels in this sequence are mixed, the distance function is not able to discriminate labels well, we say that the precision is low.

The task to test the classification performance of the presented methods was performed by hiding a query instance of the segmented cuneiform signs and comparing the remaining cuneiform signs instances against the query instance. The retrieved candidates were ranked by similarity from most similar to least similar. The classification performance of the presented methods was then compared using a precision recall graph. We evaluate our approach on a set of vectorized cuneiform tablets manually segmented into cuneiform signs. There are 160 cuneiform sign instances and 32 cuneiform sign classes in our test dataset.

We evaluate two of the three explicit wedge models, i.e. the bag-of-properties model and the Gaussian mixture model, in Chapter 5, Section 5.5, pp. 120. These models have been developed in unison with our extension of Howe's part-structured method. The testing methodology employed for these methods differs from the testing methodology presented here, it does not necessitate an a-priori manual segmentation of the documents, and it is therefore not directly comparable with the methods presented here. It does, however, allow us compare the performance of the explicit methods to state-of-the-art word-spotting methods in Latin script. In opposition, the keypoint feature model is part of the following evaluation.

4.6.1 Precision Recall Graphs

A precision recall graph visualizes the performance of a retrieval system, its precision, at different recall values. Precision P is the fraction of relevant retrieved items. True positives (TP) are relevant items that are in the result set. False positives (FP) are irrelevant items in the result set.

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{4.32}$$

Recall R is the fraction of retrieved relevant items of all relevant items. False negatives (FN) are relevant items that are not in the result set.

$$R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{4.33}$$

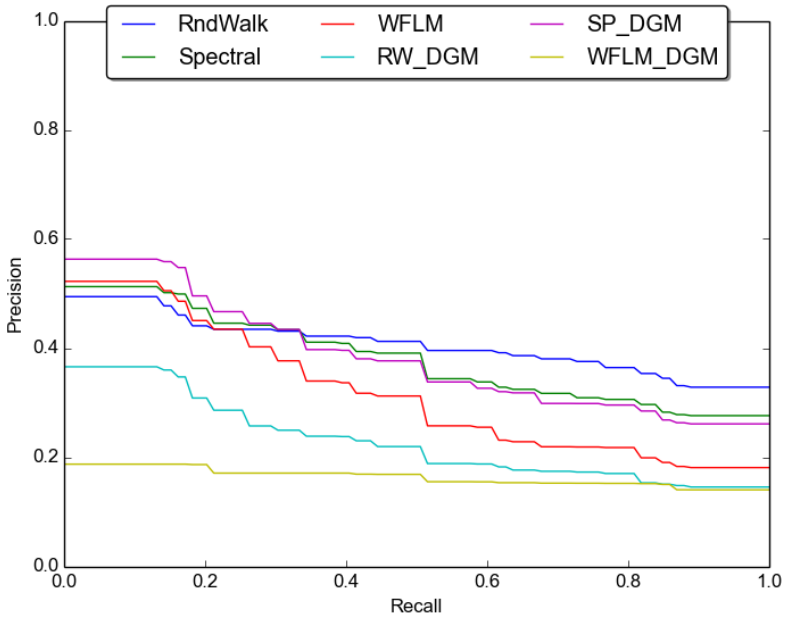


Figure 4.16: Precision Recall graph for three graph based methods. The Weisfeiler-Lehman Graph Kernel (*WFLM*), the spectral decomposition (*Spectral*) and the random walk graph kernel (*RndWalk*). Then, the methods extended with the delaunay triangulation are *WFLM_DGM*, *SP_DGM* (for the spectral decomposition) and *RW_DGM* (for the random walk kernel), respectively.

Then, the precision recall visualizes the precision of the result set as the recall is increased. Intuitively, the count of relevant results increases as the whole result set increases in size but the count of relevant items gets smaller. In the trivial case, returning all results, the recall is maximal, all relevant items have been returned, but the precision is minimal, most returned items are irrelevant. An optimal retrieval system has a high precision for most recall values, that is, the curve is close to the top of the graph.

4.6.2 Discussion on the Impact of Triangulation

Figure 4.16 shows the classification performance of the graph based methods with and without triangulating keypoints. The Delaunay transformation reduces precision greatly for the Weisfeiler-Lehman graph kernel. This kernel counts the number identical subtrees in both graphs. Since many vertices in a Delaunay triangulated graph have the same degree, two geometrically dissimilar triangulated graphs will share a high number of subtrees rendering them indistinguishable for the Weisfeiler-Lehman graph kernel.

The decrease in performance for the random walk kernel can be attributed to the same problem. Dissimilar triangulated graphs share a lot of random walks since most vertices are reachable by a high number of different walks. The random walk method and the Weisfeiler-Lehman graph kernel achieve better classification performance when the untransformed cuneiform graphs are used. Variable node degrees and unique walks in the untransformed graphs enable those methods to differentiate graphs representing wedge constellations better.

The spectral decomposition, on the other hand, has better precision when extended with Delaunay transformed graphs. The spectral decomposition can be seen as a series of minimal cuts [Chu97] of a graph where the edge density is lowest. Translation and rotation of wedges are therefore detectable by changes in connectivity of the graph partition leading to a better classification performance than just using the graph topology. An exemplary cuneiform sign retrieval is shown in Figure 4.17 using the best performing graph based similarity method, the spectral graph decomposition for Delauney triangulated graphs.

4.6.3 Comparison with the State of the Art

The classification performance of our method is compared to three standard methods used in handwritten text recognition. We compare to the DTW method [RM07], the Word Warping method [KBS11] and a HMM classifier. None of these three standard methods work on vectorized data. We rasterize wedge constellations to make these methods applicable to our dataset. All wedge constellations are rasterized to a uniform line height and a variable width. The conversion takes place by filling and rasterizing the closed spline paths describing the strokes of the wedge constellations. The result is no different than rendering the vector data in a vector graphics editor. We tested 50 pixels and 300 pixels line heights and found that a 150 pixels line height yields the best classification performance for the three methods. The configuration parameters of the four methods are as follows:

Dynamic Time Warping We limit the DTW matrix to a band of 15 pixels in either direction. Neither a higher nor a lower limit improve classification performance.

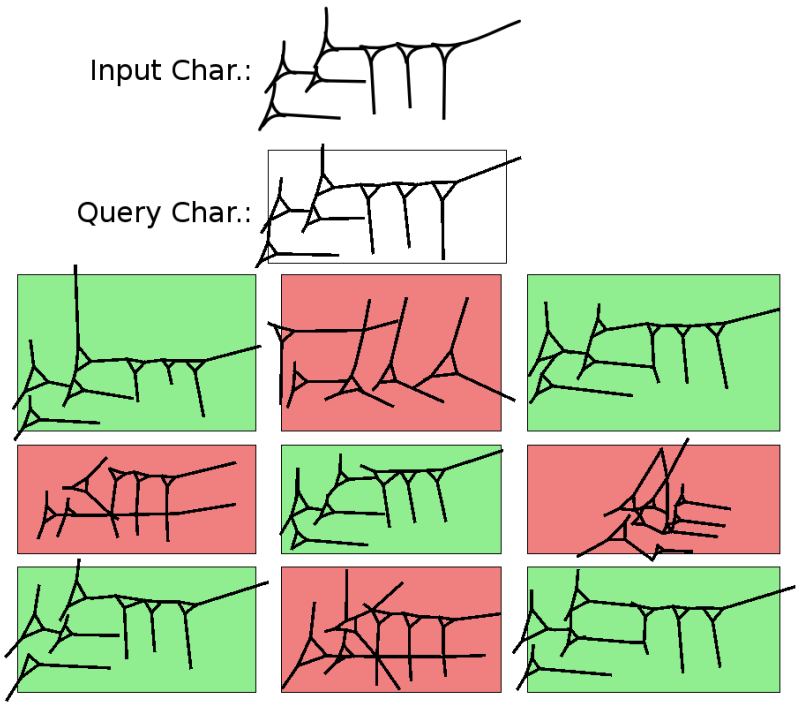


Figure 4.17: The top 9 results for the task to retrieve the query sign are displayed. The query sign is represented as a set of strokes. This set is then transformed into the graph representation. The kernel used for similarity is the spectrum kernel extended with Delaunay triangulation. The results are ordered from best (top left) to worst (bottom right). Cuneiform signs with a green background have been correctly classified, signs with a red background have been incorrectly classified.

Word Warping We choose a grid spacing of 10 pixels and perform ten refinement iterations. A lower grid spacing gives only a negligible improvement on classification performance but a severe impact on runtime. More refinement steps do not increase classification performance.

Hidden Markov Model Wedge constellation images are split 10 pixel wide slices. We tested 5 pixels and 20 pixels wide slices and found that 10 pixel wide slices yield the best classification performance. The image slices are transformed into a Histogram of oriented Gradients (HoG) [DT05] with 4 possible orientations and 5 histograms per slice. Our HMM state machine has a left-to-right topology with 10 states and uses 20 Gaussian mixtures to represent the feature vectors. The state machine is trained on the query word using the Baum-Welch algorithm. Candidate words are decoded using the Viterbi algorithm [Rab89].

Iterated Closest Points Wedge constellations are converted into point clouds by sampling the closed spline paths equidistantly. We choose a sampling rate of 1 point per 1 millimeter of spline path. This leads to approximately 40 sampling points per wedge. Lower sampling at 0.3 pixels or higher sampling at 10 pixels does not improve classification performance.

An exemplary retrieval result using the keypoint feature descriptor is shown in Figure 4.18 and the precision recall results for all evaluated methods are shown in Figure 4.19.

The naïve ICP approach proves surprisingly successful at low recall. Matching point clouds is highly discriminative and provides good results if wedge constellations are highly similar. The method, however, is not able to deal with the high variability of wedge constellations. It fails to classify cuneiform signs written differently, which results in bad classification performance at high recall.

DTW struggles with high visual complexity of cuneiform signs on the vertical axis. Although Word Warping performs better than DTW, by allowing deformation in both directions, it can only match the performance of the ICP method at most.

The HMM classifier performs at the level of the DTW method. The bad classification performance can be attributed to having only one training sample and the inability to efficiently model vertical complexity of cuneiform signs.

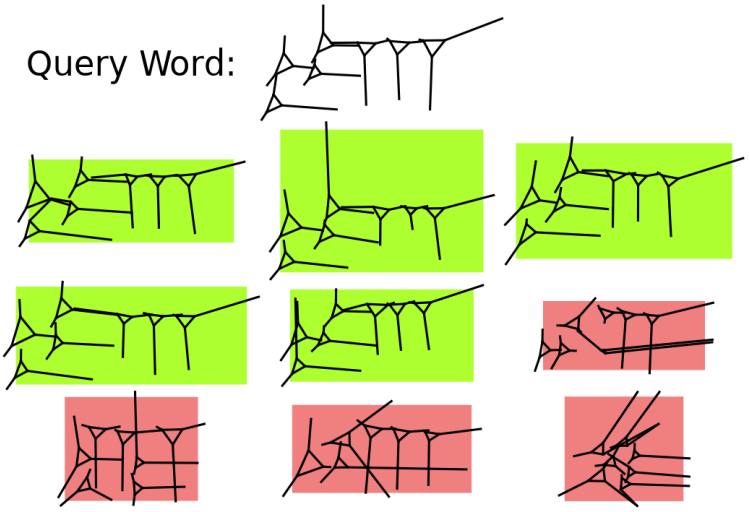


Figure 4.18: Given a query cuneiform sign the keypoint model for describing wedges retrieves 9 signs most similar to the query. There are 5 True Positives (green background) and 4 False Positives (red background). The cuneiform signs in this figure have been reconstructed and visualized using only their keypoint feature descriptors.

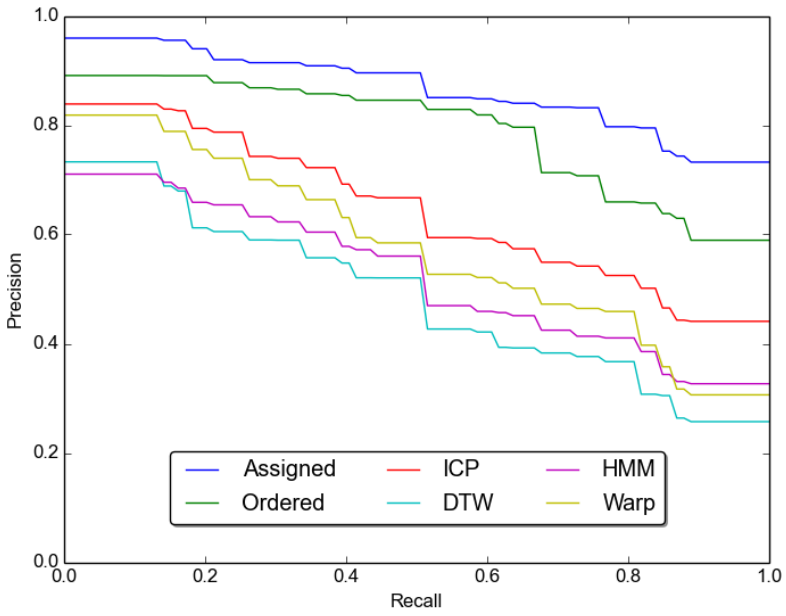


Figure 4.19: Precision-recall curves for our approach with optimized wedge assignment (Assigned) and naive assignment (Ordered) compared against three approaches commonly used for handwritten text recognition.

Our method outperforms all other methods on our dataset. Matching wedges by optimizing a linear assignment problem yields significantly better results at very high and very low recall. In the case of very high recall values it is important to correctly recognize damaged cuneiform signs or cuneiform signs written with reordered wedges. For very low recall values the cuneiform signs are mostly identical, and small differences in arm length and wedge orientation decide between correctly and incorrectly classified cuneiform signs.

4.7 Summary

In this section we introduced the notion of distance functions to measure a similarity between two objects. The methods presented in this chapter have been successively developed and published together in [BGM15b; BGM15a] and [BHM16]. For comparing cuneiform wedge constellations we developed two approaches differing in their assumptions on the structure of the wedge constellations being compared. The distance function of the first approach assumes no underlaying structure and model wedge constellations holistically. From this assumption we derive five distance functions.

We rasterize wedge constellations and compare their projection profiles with the DTW method. When representing wedge constellations as mathematical graphs, we compute random walks, subtrees, and spectral decompositions to measure the similarity of two graphs. By discretizing the set of strokes of a wedge constellation into a point cloud, we can measure the similarity directly by embedding both wedge constellations into a similar pose using the ICP method and then computing the sum of distances between corresponding points.

Our other approach assumes that the most basic constituent of cuneiform script is the wedge-shaped impression created by the writing tool, a rectangular stylus. We model each wedge by six keypoints, three for the triangular wedge-head and three for the endpoints of the wedge-arms. Then, we derive three feature vector representations. The Keypoint Model is a direct representation of the six keypoints in a feature vector, the Bag-of-Properties model is a binary vector of properties present in a wedge and the Gaussian Mixture Model learns prototypical wedges that serve as templates actual wedges are compared against.

We evaluate the presented distance functions against each other and the state of the art classifiers, such as a HMM classifier. The results are compared in a precision recall graph. We find that the Keypoint Model outperforms all of our other distance functions and all of the state of the art classifiers.

5 Symbol Spotting

Literary analysis, literary edition and other forms, such as interpretation and criticism of text, always require the cross-reference of works for the comparison and discovery of sources. For modern Latin text, such work is aided by linear time complexity [KMP77] word search facilities, allowing scientists analyzing literary text to seek out references easily.

Transcription and transliteration of cuneiform script requires intensive study of cuneiform signs and reading of related cuneiform tablets for cross-reference to understand the underlaying context in which signs are used. Symbol spotting, similar to word-spotting for Latin text, enables automated searching and exploration of both transcribed and not yet transcribed cuneiform tablets to discover the meaning of a sign by its usage.

In this chapter, we develop and evaluate means of searching wedge constellations without an a-priori segmentation of the cuneiform tablets being searched. In contrast to the previous chapter, where we developed methods for computing the similarity of two given wedge constellations, here we introduce approaches that additionally locate wedge constellations on a tablet. Such a retrieval task is called segmentation-free word-spotting. Since the following approaches do not require the queries to be known cuneiform signs, we therefore perform segmentation-free wedge constellation spotting.

5.1 Challenges and Objectives

This section provides an overview of the main objectives and contributions of this chapter.

Motivation Segmentation of cuneiform script is challenging due to its complex grammatic structure, meaning dependent upon grammatical case and lack of whitespace. For computational analysis of cuneiform script word-spotting algorithms cannot relay on an a-priori segmentation.

Challenges Similar to approach for handwritten Latin script, word-spotting methods assume that most complexity is expressed in the horizontal axis. Further, approaches for Latin assume a left-to-right reading order, a constraint that also is not supported by texts in cuneiform.

Objectives Develop methods for discovering free-form arrangements of wedges on basis of a query wedge constellation. Additionally, this approach has to support our shared keypoint feature descriptor.

Related Work We review state-of-the art segmentation-free spotting approaches to historical Latin script. The presented approaches are based on sliding a window across the document and computing a distance between those two. Different a-priori feature transformations accelerate this process.

Input The presented cuneiform sign spotting methods work on either 1) the set of wedges extracted in Chapter 3, Section 4.5, pp. 81 represented explicitly by keypoint feature descriptors 2) or when evaluating to the state-of-the-art, as a rasterized set of strokes. Further, query wedge constellation is given that is being searched.

Output The results are a set of bounding boxes that indicate areas in the searched documents that contain the query wedge constellation.

Methods We adapt our data for a state-of-the-art word-spotting approach based on HMM. Additionally, we employ the similarity functions developed in Chapter 4 and extend a part-structured spotting approach that can directly work with the our set of extracted wedge feature vectors.

Evaluation Based on ground truth created for the evaluation in Chapter 4, Section 4.6, pp. 95 of labeled cuneiform signs, we compute expected bounding boxes for each query sign. Methods are evaluated using a precision-recall graph. We find that the explicit modeling of similarity of wedge-shaped impressions and the free-form arrangement allowed by part-structured models outperforms the implicit and linear modeling of HMM based approaches for cuneiform script.

Discussion We discuss the impact and failure behavior of the different explicit wedge models introduced in Chapter 3, Section 3.4, pp. 43.

Publications The work presented in this chapter has been published in [BHM16].

5.2 Related Work in Handwriting Recognition

The common approach to automated reading of written text is Optical Character Recognition (OCR) which aims to detect written letters and reconstruct their layout, i.e. words and sentences, in documents. Historical text is often damaged, hand-written, used unknown fonts or is written in a no longer used script, c.f. Chinese Sutra [MHK09] or cuneiform. In such cases, automated reading is limited to word-spotting. Given one (one-shot) or more (multi-shot) examples of a query word this word is then pattern matched against a document collection.

The concept of word-spotting was first proposed by Manmatha et al. [RMC96]. It makes the challenge of computational analysis of written text, e.g. searching or counting n-grams, in an unknown and hard to read script, e.g. medieval Latin, tractable. Instead of optical recognition of the written characters, a comparison on basis of the word image regions is performed. While this task is similar to image recognition tasks, several assumptions can be made about written script that are not possible for image recognition. Most written script has a writing direction, is written in lines of similar length and uses only one color. These assumptions allow for recognition tasks that model most information in the writing direction, i.e. sequential models, and where the text can be segmented into lines using the projection profile.

Rodriguez et al. [RSP09] introduce in their work the use of HMMs to model and query handwritten script. The authors evaluate their approach on handwritten

letters. Line segmentation is performed by computing the projection profile and word segmentation by computing connected components. In a first step of pruning, a linear classifier [Kri+05] rejects unlikely word candidates. The remaining word images are transformed using a sliding window into a sequence of feature vectors computed from HoG. The authors explore the use of two different HMM types, continuous and semi-continuous. The semi-continuous HMM uses shared weights for the GMM modeling the feature vectors. This restriction significantly increases the performance of single-shot word-spotting since the amount of parameters to be learned is greatly reduced.

Rothacker et al. [Rot+15] employ their word-spotting framework based on bag-of-features HMM [RRF13] for segmentation-free spotting of cuneiform signs. A 3D scan of a cuneiform tablet is transformed into a 2D representation using the curvature of the tablet surface. Then, the authors learn a HMM on a single example query word and spot cuneiform signs by decoding the learned HMM on a grid of possible positions on the cuneiform tablet. Their approach is based on their previous work in segmentation-free spotting in historical Latin script. We compare our approach to spotting to a modified variant of their method which is suited more favorably for our data.

In their work Rusiñol et al. [Rus+14] aim to model the spatial complexity of handwritten script by using Spatial Pyramid Matching (SPM) [LSP06] with Bag-of-Visual-Words (BoVW) populated by dense Scale Invariant Feature Transform (SIFT) [FVS08] descriptors. The authors divide the document image into patches of four different lengths to accommodate different query lengths. Each document patch is first partitioned horizontally into two regions, used for the SPM part of the descriptor. The dense SIFT sampling then populates the SIFT feature-vector, of the two half regions and the whole region, for the resulting image patch descriptor. These vectors are then re-weighted using the TF-IDF model [SB88] to emphasize visual words frequent in a local patch but infrequent in the corpus. Since the variability of written script results in words being represented by different visual words, the authors transform the feature descriptor using Latent Semantic Analysis (LSA) [Dee+90]. The underlying assumptions is that while words may not have the same visual words representing them, they will have the same topics and lie close in LSA space. Queries are retrieved by performing the same transformation on the query image as on the document image and then computing the cosine similarity between the query descriptor and all document descriptors. Regions in the document that have gained the most votes are likely candidates for the results.

Almazan et al. [Alm+14] argue that since most approaches implement a sliding window approach, costly feature descriptors are best avoided, in favor of computationally efficient ones as HoG. In their approach the author propose two stage word-spotting where the first stage provide fast but inaccurate results and the second stage refines those. For the first stage the authors divide the document into equal sized HoG cells. The query word is also divided into identically sized HoG cells. A sliding window of the query computes the convolution of the query HoGs over the document HoGs for all cells. In a second step, the authors use the Exemplar Support Vector Machine (E-SVM) framework [MGE11] to learn a more accurate representation of the query. In this framework a SVM is trained on slightly shifted windows of the same query as positive samples versus many random windows from the document as negative samples. The candidates generated from the first step are then re-ranked using the learned classifier. To increase the reach of the word-spotting, new queries on basis of the spotted words are run to improve the retrieval results.

5.3 Sequential Modeling

Rothacker et al. [RRF13; Rot+15] approach the task of word-spotting by means of combining a dense SIFT sampling of the document for BoVW feature descriptors with sequential model based on a HMM. For handwriting where spatial information is typically laid out horizontally, modeling with a HMM provides the necessary sequential expressiveness while ignoring horizontal spatial relationships by means of BoVW. Since their approach achieves state-of-the-art performance in segmentation-free handwritten word-spotting tasks, albeit on Latin script, we evaluate their method on our born-digital cuneiform tracings.

5.3.1 Feature Extraction

Our dataset is derived from different sources in heterogeneous representations. To make this data available for the sequential modeling approach, we rasterize the cuneiform tablet set of strokes represented by splines. The resolution of the rasterized documents is chosen to reflect the script size in the dataset used in the approach by Rothacker et al. [RRF13]. Therefore, the typical height of our rasterized cuneiform signs is 60 pixels. Then, we employ their feature extraction

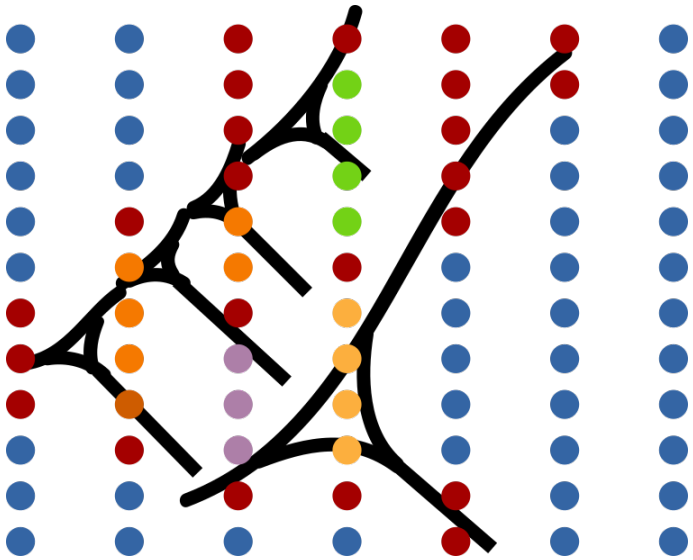


Figure 5.1: Dense SIFT features on a rasterized cuneiform tracing. The color of the dots represents the class of the detected SIFT feature in the learned dictionary of features.

pipeline and densely sample SIFT features from rasterized document images. The extracted features are clustered using the k-means [Mac67] algorithm yielding reduced codebook of 1024 visual words. Figure 5.1 illustrates the dense SIFT sampling of cuneiform sign. Features are extracted by sliding a thin window (5 pixels width and 60 pixels height) over the query words, the words being searched, and extracting the BoVW feature vectors. We denote these features as the observed emissions $y_{t=1...T}$.

5.3.2 Hidden Markov Model Topology

The HMM introduced by Baum et al. in [BP66] is an unsupervised generative technique for describing a sequence of events by means of an underlying unobserved state machine. The state machine is a graph of transition probabilities between each of the states. Each state has a set of emission probabilities of observed events. The Model assumes that the future states of a system is only determined by its current state without regard to previous states, this is the Markov assumption. Let the following definitions describe a HMM.

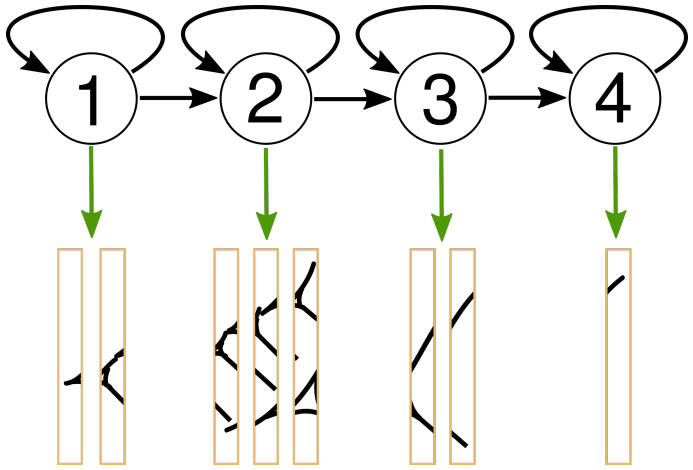


Figure 5.2: The topology of a HMM state machine in chain topology. Only loops and steps forward are allowed. The assignment of probability mass is heavily biased towards loops.

$$
\begin{aligned}
\phi_{i=1\ldots N, j=1\ldots N} &: \text{Probability of transition from state } i \text{ to state } j \\
\mu_i \in \mathbb{R}^K &: \text{Means vector of observations } x \text{ associated with state } i \\
\Sigma_i^2 \in \mathbb{R}^{K\times K} &: \text{Covariance matrix of observations } x \text{ associated with state } i \\
x_{t=1\ldots T} &: \text{State of observation at window position } t \\
y_{t=1\ldots T} &: \text{Observation at window position } t
\end{aligned}
\tag{5.1}
$$

The observation y_t is a histogram of visual words occurring at sliding window position t of the query words. The unobserved state of the HMM for an observation y_t is denoted by x_t. While the graph for the unobserved state machine, the transitions ϕ_{ij}, can be arbitrary, learning a high amount of transition probabilities, given only few observations, is not robust. In handwriting recognition topologies such as the chain topology are used to exploit the sequential way in which Latin script is written. The chain topology only allows state transitions from left-to-right and loops. Figure 5.2 illustrates a HMM state-machine in chain topology configuration. The sliding window positions are linearly mapped onto the HMM states sharing many successive window positions. In one of their approaches, Rothacker et al. the number of HMM states is 17% of the count of window slice of the query. Let N_i be the count of observations x mapped onto state i. Then, the transitions probabilities inducing a chain topology are defined as follows.

$$\begin{aligned} \phi_{ii} &= \frac{N_i - 1}{N_i} \\ \phi_{ij} &= \frac{1}{N_i} \end{aligned} \tag{5.2}$$

The parameters of a HMM can be estimated with Baum-Welch training. It is an instance of the EM algorithm [Bau+70]. Given a sequence of observed emissions, Baum-Welch training executes three inference steps, the forward procedure, the backward procedure, and an update step until the convergence of the estimated parameters. The training maximizes the probability of an observed sequence $y \in Y$ with respect to the parameters $\theta = \{\phi, \mu, \Sigma\}$ of the HMM.

$$\theta^* = \arg\max_\theta P(Y|\theta) \tag{5.3}$$

This concludes the learning phase. While the HMM can be trained using one single query example, more examples increase its classification performance.

5.3.3 Result Retrieval

Results are retrieved by sliding a query sized window over the document image and decoding the learned HMM on the document image BoVW feature vectors. The algorithm named after Andrew Viterbi [Vit67] is used to find a maximum probability decoding of a HMM. Given a sequence of observations and model parameters of hidden states of a HMM, Viterbi decoding finds a most likely path through the HMM states. This path through the hidden states is the path that most likely caused the sequence of observed states. Let $y_1 \dots y_T$ be a sequence of observed visual words slices in the document image, then the Viterbi algorithm determines the probability of the most probable state transition sequence $x_1 \dots x_T$ with respect to the learned HMM configuration θ^*.

$$\max P(x_1, \dots, x_T, y_1, \dots, y_T \mid \theta^*) \tag{5.4}$$

For retrieval, we are not interested in the actual sequence of HMM states, but the probability of an observed sequence. That is, we compute the probability

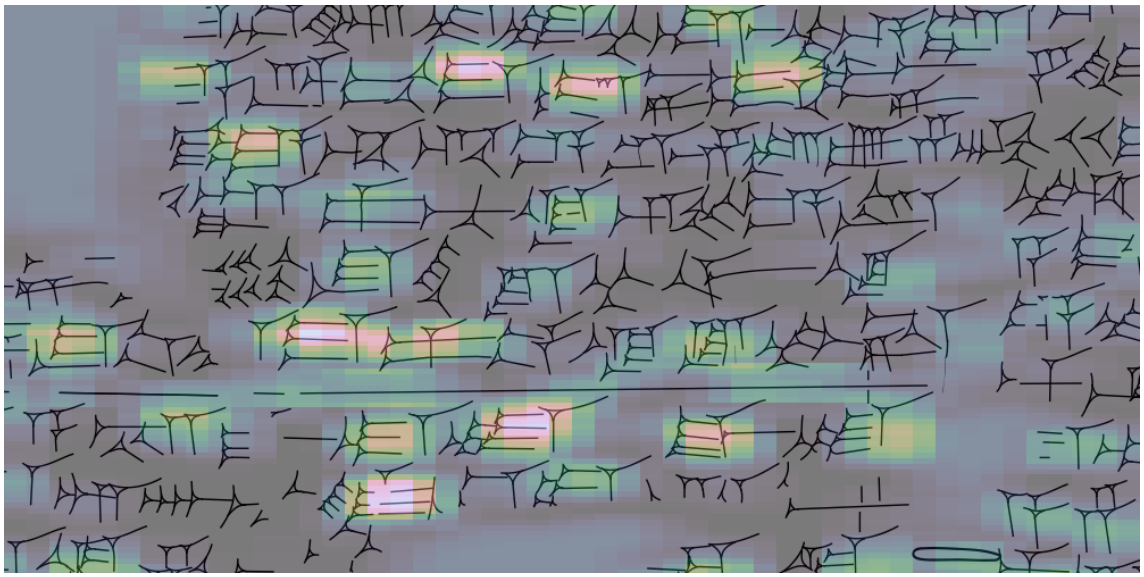


Figure 5.3: A search window is moved across the raster image of a cuneiform tracing. For each position the Viterbi decoding for a HMM trained on a query is computed. A higher probability is denoted by a lighter color.

of the observed visual word sequence being generated from the learned query word visual word sequence. A high probability indicates that the observed visual word sequence is similar to the learned sequence, we spotted a similar word. Figure 5.3 shows the computed probabilities of a search window sliding over the document image. Results are than retrieved using non-maximum suppression [Can86] of the probability field.

5.4 Part-Structured Modeling

In this section, we adapt an approach developed by Howe [How13] for our wedge feature representation. In his work on word-spotting, Howe [How13] introduced the usage of part-structured models. These are generative models of word appearance.

Part-structured spotting by Howe [How13] provides a framework for spotting words modeled as ink-balls. In cooperation with Howe, we adapted the part-structured model to allow spotting of arbitrary high-dimensional point clouds, albeit, still constrained to transposition on two-dimensions. Since all our wedge models construct a feature-vector, they all are applicable as underlaying representations for the part-structured model.

Compared to Howe's original approach, query wedge constellations are directly represented as trees of feature-vector. No thinning and ink-ball placement is performed. We use a similar greedy approach to tree construction where the nearest feature-vector, on the plane induced by the two unconstrained dimensions for spotting, is attached with an edge to the existing tree.

Then, instead of computing the distance transform of the medial-axis of the document, we compute the euclidean distance between all query constellations wedge feature vectors and all document wedge feature vectors. From this, a distance transformed image is computed that is then summed according to scheme presented by Howe [How13].

5.4.1 Query and Document Feature Representation

In the approached presented by Howe, a medial axis transform of the query is computed and points are placed equidistantly on the skeleton of the query. In our modified approach, we use the semantically rich feature description introduced in Chapter 3, Section 3.4.1, pp. 43 of cuneiform wedges and represent a query wedge constellation by a collection of these high-dimensional points. We connect these points into a tree structure by starting at the point closest to the center and greedily add points while avoiding loops. Figure 5.4 illustrates the tree structure on a cuneiform sign.

Algorithm 8 describes the construction of a tree from a set of vertices V. Let $n = |V|$ be the count of vertices in this set. The minimum operations used have a time complexity of $O(n)$. Checking a vertex for presence in the set V' has an amortized time complexity of $O(1)$ due to the use of an efficient hashing data structure. Since for every vertex a minimum must be computed inside the loop, the algorithm has a time complexity of $O(n + n * n) = O(n^2)$.

In similar fashion, the document to be queried is represented as a collection of points, the positions of wedges on the document and their high-dimensional wedge feature vectors. These have been extracted by the methods presented in Chapter 3. Figure 5.5 on page 117 provides an illustrated overview of the word-spotting framework and our extensions to it. A rigorous description follows in the next section.

Algorithm 8 Greedily connect a collection of points to a tree structure.

```
procedure GREEDY_TREE(V)
    ▷ Initialize set of vertices in tree structure.
    ▷ It is empty at beginning, vertices are successively
    ▷ added.
    V' ← ∅

    ▷ Initialize empty set of vertices which will hold
    ▷ the structure of the the constructed tree.
    E ← ∅

    ▷ The root vertex is the most central vertex.
    V' ← V' ∪ {min_{p∈V} ‖ mean(V) − p‖}

    ▷ For each vertex not yet in the structure
    ▷ find the closest vertex add an edge between those.
    for v ∈ V do
        if v ∈ V' then
            ▷ The current vertex already is part of the
            ▷ tree structure.
            Continue
        end if

        ▷ Find closest vertex in tree for current
        ▷ vertex which is outside.
        v' ← min_{v'∈V'} ‖v − v'‖

        ▷ Add newly discovered edge to the set of edges
        ▷ and the set of visited vertices.
        E ← E ∪ {(v, v')}
        V' ← V' ∪ {v'}
    end for

    ▷ Return edges defining the structure of the tree.
    return E
end procedure
```

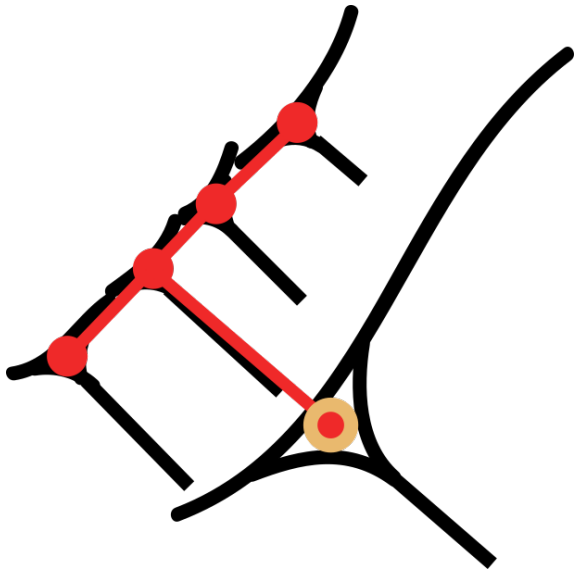


Figure 5.4: Tree structure computed by greedily connecting points, centers of wedge-heads, of a cuneiform sign.

5.4.2 Displacement Energy Computation

Word spotting is performed by finding the configurations of the query sign model with minimal energy that match the observations, the wedges, in the document. Let $Q = \{q_1, ..., q_n\}$ represent a set of wedges that form some wedge constellation or an other unit of interest, and let $\{v_1, ..., v_n\}$ represent their positions, taken as the mean of the three wedge head vertices. We assemble the parts into a tree structure by greedily forming pairwise links between disconnected units. Without loss of generality let q_1 be the wedge closest to the group's center of mass; this wedge becomes the root of the tree. Letting $q_{i\uparrow}$ indicate the parent of q_i in the tree structure, we define the default offset m_i for each child node. Note that m_1 and $q_{1\uparrow}$ are undefined, since q_1 has no parent.

$$m_i = v_i - v_{i\uparrow} \tag{5.5}$$

Having identified the model's tree structure and default offsets, we define a deformation energy E_ξ for any proposed configuration of the model, $Z = \{z_1, ..., z_n\}$. This energy is a quadratic function of the difference between the observed offsets and the model default, and is invariant under rigid translations.

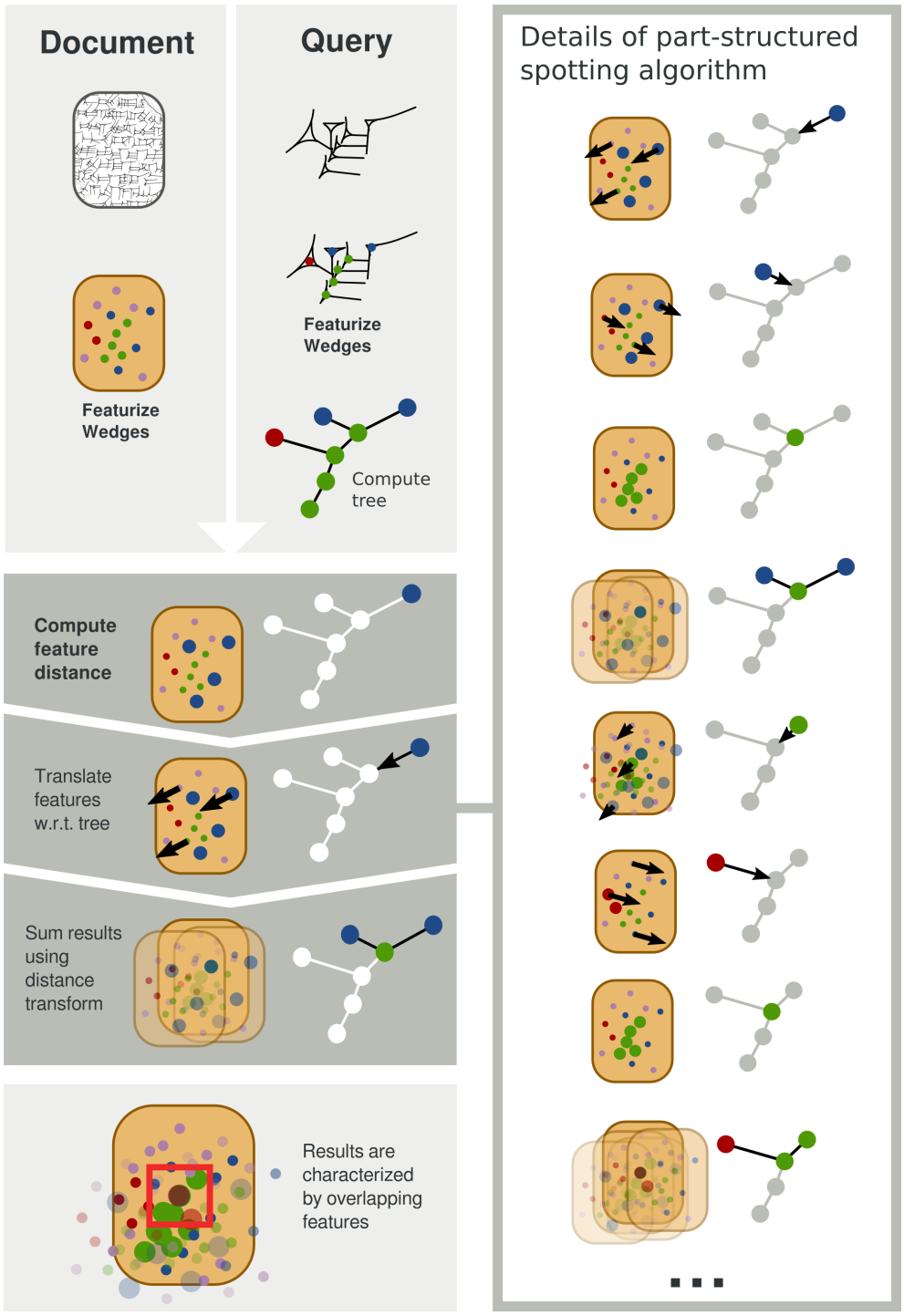


Figure 5.5: Illustration of the word-spotting framework introduced by Howe and our extensions based on methods developed in previous chapters emphasized in bold text. The word-spotting pipeline proceeds in three phases. First, using our extraction methods, document and query features (color dots) are computed. Then, using our distance metrics for these features, we compute document displacement energies (tablets with enlarged dots). The displacement energies are summed according to Howe's framework (overlaid tablets). Finally, the query is found at local minima of the resultant document.

$$E_{\xi}(Z) = \sum_{2}^{n} \|(z_i - z_{i\uparrow}) - m_i\|^2 \tag{5.6}$$

Given a cuneiform document, word-spotting attempts to identify locations that are likely matches to the query model. More precisely, if $T = \{t_1, ..., t_N\}$ is the set of wedges in the target, with positions $U = \{u_1, ..., u_N\}$, then we seek locations x where the following energy function reaches a minimum and lies below some target threshold.

$$\mathcal{E}(x) = \min_{Z|z_1=x} [E_{\xi}(Z) + E_{\omega}(Z, Q, T, U)] \tag{5.7}$$

The second term in this expression measures the proximity of model wedges to suitable target wedges under the proposed configuration. A parameter α trades off between spatial proximity and wedge match quality D, as computed using one of the methods from the previous section.

$$E_{\omega}(Z, Q, T, U) = \sum_{i=1}^{n} E_{\omega}(z_i, q_i, T, U) \tag{5.8}$$

$$E_{\omega}(z_i, q_i, T, U) = \min_{j=1}^{N} \left[\|z_i - u_j\|^2 + \alpha D(q_i, t_j)\right] \tag{5.9}$$

The energy function in Equation 5.7 can be minimized via an efficient dynamic programming algorithm, as in prior work on part-structured models [How15]. We first write an expression for the minimum energy of arbitrary subtrees of the model, where $Z_{i\downarrow}$ represents the positions of wedge q_i and all its descendants.

$$\mathcal{E}_i(x) = \min_{Z_{i\downarrow}|z_i=x} [E_{\xi}(Z_{i\downarrow}) + E_{\omega}(Z_{i\downarrow}, Q_{i\downarrow}, T, U)] \tag{5.10}$$

This in turn can be rewritten recursively as the wedge match at the root plus the minimum energy over all possible child configurations, as adjusted by the model offsets.

$$\mathcal{E}_i(x) = E_{\omega}(x, q_i, T, U) + \min_{Z_{i\downarrow}|z_i=x} \mathcal{E}_{\downarrow i}(x) \tag{5.11}$$

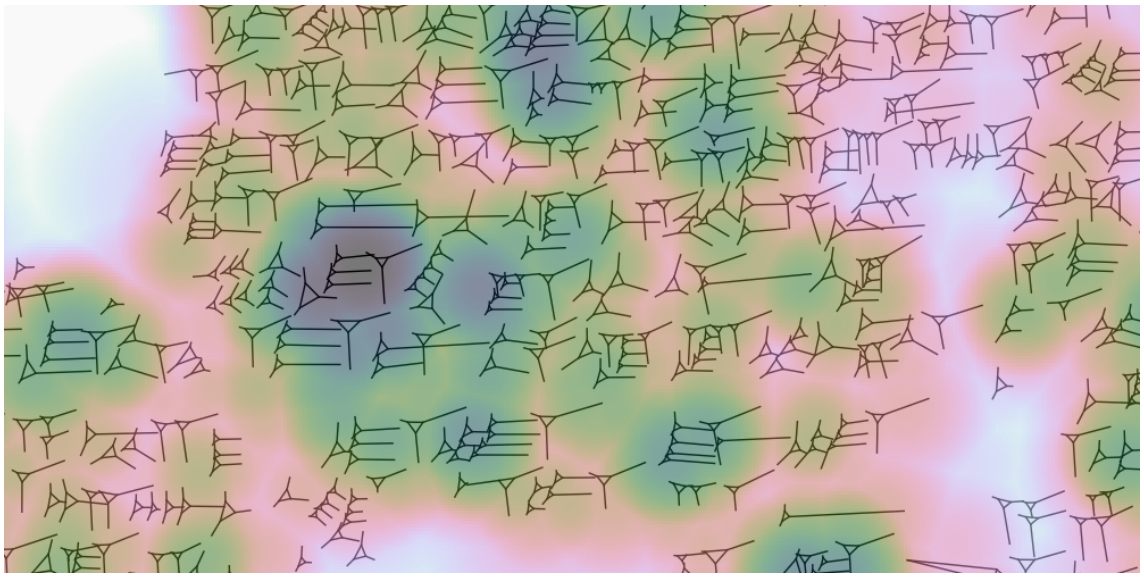


Figure 5.6: The deformation energy necessary to fit the query model to the document image. Low values indicate a particularly good fit for the query sign as it has to be only minimally deformed. These minima are the results of the retrieval process.

$$\min_{Z_{i\downarrow}|z_i=x} \mathcal{E}_{\downarrow i}(x) = \sum_{j|j\uparrow=i} \Gamma\left(\mathcal{E}_j(x - m_j)\right) \tag{5.12}$$

Here Γ denotes the generalized distance transform (GDT) [FH05; FH12], which performs the minimization over the deformation term. At the leaves of the model, the child energy contribution $\mathcal{E}_{\downarrow i}$ is zero and the energy function $\mathcal{E}_i$ can be computed simply via a GDT, with the function values with respect to x represented on a discrete grid. Moving up the tree, parent node energies $\mathcal{E}_i$ can be computed using the results at the leaves, translated by the offset m_j and passed through another generalized distance transform before being added up. This process eventually yields the energy of the entire model over the grid of possible root positions. A visualization of this energy field is shown in Figure 5.6. Strong local minima on this grid are the locations where the model matches well. We place bounding boxes with the extents of the query around these minima and perform a non-minimum [Can86] suppression of the results to remove close duplicates.

5.5 Evaluation of the Modeling Approaches

We evaluate our methods on a dataset of two cuneiform tablets line traced by professional Assyriologists using a graphics editor. These tablets contain around 500 identifiable cuneiform signs. The tablets are only incompletely manually labeled and segmented. This precludes a precise evaluation of the performance of the presented methods on the given dataset and offers us no possibility to exactly analyze the methods on their recall performance.

However, our evaluation scheme makes the relative differences in retrieval performance still valid and allows for a comparison of the methods. We perform retrieval queries by example using the set of segmented cuneiform signs. Given the size of our dataset, we assume that each cuneiform signs class has in average 30 instances. Perfect recall is achieved when 30 instances are retrieved. An expert then decides for each returned result whether it belongs to the class of the query and tags it with either true positive or false positive.

We evaluate our method against the work of Rothacker et al. on word-spotting on Latin script [RRF13]. Since we use vectorized transcriptions of cuneiform tablets as data, we cannot employ their work on word-spotting on cuneiform tablets for comparison [Rot+15]. We have no 3D data or curvature data available for our dataset.

To make our data available to their bag-of-features word-spotting framework, we first rasterize the vectorized dataset to raster images. The size of the raster images is chosen so that their choice of slice parameters and sliding windows advancement is optimal for our data. That is, we chose 40 pixel sized structuring elements for the dense SIFT [Bic+06] transformation with a 5 pixel wide regular grid. The query examples are taken from the incompletely segmented document and sliced with 5 pixel wide horizontal slices advancing 2 pixels. We use the same HMM chin-topology as presented in their work.

The results of our evaluation are shown in Figure 5.7. The Gaussian mixture model yielded the least satisfactory results. Closer inspection of the inferred classes showed that they did not represent the space of different wedges well.

Binary template vectors offer more flexibility, nonetheless, they do not perform as well as the keypoint model. We attribute this to the rigidity of the defined templates. If many semantically different wedges fall into the same template, they no longer can be differentiated. A more flexible template model, more

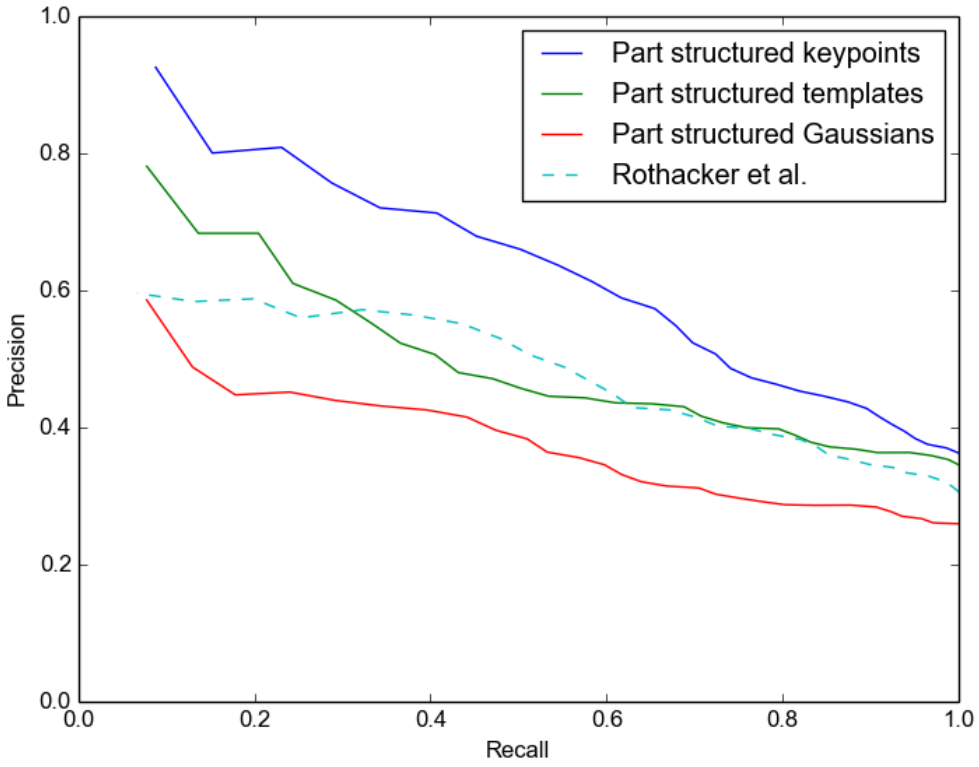


Figure 5.7: Precision recall graph for the sequential and part-structured model. The part-structured model has been evaluated with three different wedge feature descriptors.

angles and more sizes, would lead to high-dimensional feature vectors that are hard to compare.

The keypoint model performs best and outperforms the approach presented by Rothacker et al. The keypoint model uses an advantageous description of wedges when used in combination with the adapted part structured spotting. It too models two dimensional points that are compared using the Euclidean distance.

5.6 Summary

In this chapter we introduced two approaches to spotting wedge constellations. The state-of-the-art in segmentation-free word-spotting Latin script is achieved by sequential modeling approaches such as the BoVW HMM as introduced by Rothacker et al. [RRF13]. We adapted our dataset to the pipeline presented in their work by rasterizing the born-digital tracings of cuneiform tablets. Evaluation of the pipeline on our dataset showed that the significant vertical complexity expressed in cuneiform signs is insufficiently modeled resulting in bad retrieval performance.

Then, we presented a part-structured modeling approach that relies on a tree-structure of ink-balls connected by spring-like potentials. We employed our semantically descriptive wedge feature vectors introduced in Chapter 4, Section 3.4.1, pp. 43 to replace the ink-ball model.

We performed an evaluation of both models on our dataset. Our adapted part-structured approach outperformed the word-spotting framework of Rothacker et al. on our test dataset. Therefore, we achieve state-of-the-art segmentation-free constellation spotting on cuneiform. As a result of our common feature representation, our adapted part-structured wedge constellation spotting pipeline is applicable to heterogeneous datasets of cuneiform tracings. The results of this work have been published in [BHM16].

6 Applications and Concepts

With the development of a computational representation of cuneiform script, similarity metrics for wedge constellations and segmentation-free spotting of wedge constellations, we introduce in this chapter two applications of these methods to gain new insights into cuneiform script. While qualitative aspects of cuneiform script and its history is being researched by Assyriologists, we present quantitative analyses common to computational linguistics in Latin script.

6.1 Challenges and Objectives

We make use of our computational tools to perform analyses, such as pattern-mining and spatial n-gram mining, which were hitherto not feasible to do manually. We also present an approach towards automatic transliteration of transcriptions by mining common pairs of wedge constellations and respective transcriptions.

Motivation Computational analysis of written language is based on robust methods for similarity and search of signs. Further, understanding and explicit representation of the basic constituents of cuneiform, its wedge-shaped impressions, enables analysis that generates new insights into its structure, e.g. frequent patterns in signs.

Challenges Pattern mining and sequence alignment is commonly discrete, e.g. text, or sequential, e.g. voice. Cuneiform is a two-dimensional freely arrangeable script with continuous expressions of wedge-shaped impressions and a very complex structure.

Objectives A precise sign spotting method and a robust similarity measure enable the discovery of constellations of wedges. Without prior knowledge

of known cuneiform signs, wedge constellations are found that are part of the list of discovered cuneiform signs. Therefore, showing that the developed methods enable the automation of cuneiform script analysis.

Related Work Since until now no computational analyses were possible, we review work in pattern mining and clustering. For automatically inferred transcriptions of cuneiform tablets we refer to work in language learning with sequential and structural models.

Input We compute spatial n-grams on basis of the set of extracted keypoint feature descriptors from cuneiform tablets. Additionally, we make use of our novel similarity functions cluster patterns and discover n-grams.

Output The result is a set of constellation of wedges that occurs frequently in the analyzed dataset. For our transliteration approach we expect a learning method to infer Latin transliteration tokens given a raster image of cuneiform wedge.

Methods A distance matrix is computed on basis of the similarity function for wedge feature vectors. Then, we perform a TSNE embedding of the matrix and extract clusters with the k-means algorithm. We learn transliterations by mining co-occurring raster image slices and tokens which are used to train sequential and structural models.

Discussion We find that our developed methods enable hitherto unachievable computational and automated linguistic analysis of cuneiform script. We discuss the future impact and possibilities our methods enable for computational analysis.

Publications We published the work in pattern mining spatial n-grams in [BM16] and the work towards automated transliterations in [BKM17].

6.2 Pattern Mining Spatial n-Grams

We build upon our previous work, i.e. extracting features from cuneiform, to cluster constellations of wedges. We describe cuneiform tablets in terms of spatial n-grams to efficiently query which locations in a tablet contain all n-grams of a query. Additionally, we describe the automatic discovery of

these patterns by mining cuneiform tablets for repeating constellations of wedges. These locations are used to perform an exact matching. We provide preliminary results in form of exemplary query results to show the viability of our method. Our main contribution in this work is a framework to decompose cuneiform tablets and identify repeating geometric patterns that are usable for a large-scale and segmentation-free search.

6.2.1 Background on Linguistic Patterns

Syllables, words and some sentence markers are expressed as unique constellations of wedges in cuneiform script and form the radicals — the most basic, semantically meaningful constituents — of the various languages. Signs and word lists as the Borger list [Bor04] enumerate known cuneiform signs and their meaning. We introduce the concept of spatial n-grams, similar to n-grams in computer linguistics, a small collection of two-dimensional points with associated high-dimensional feature-vectors. Similar to their counterparts in computer linguistics, a cuneiform sign is composed of many spatial n-grams. Unlike their counterparts, spatial n-grams have no ordering but a position in 2D of their atoms, the wedges. While pattern-mining [AH14] in sequential data and associated algorithms is a well researched topic, pattern discovery in point clouds is novel and requires more generalized algorithms [AS94]. We represent wedge constellations as a finite set of search-able patterns, spatial n-grams, which in turn are sets of wedge feature vectors as introduced in Chapter 3, Section 3.4.1 pp. 43. The feature representation and feature similarity metric of wedges forming a spatial n-gram is general and is used to derive different properties of similarity.

6.2.2 Clustering to Extract Common Spatial n-Grams

We enumerate all possible spatial n-grams by finding all constellations of wedges that are closer than some threshold value. Wedge-heads centers are stored in a kd-tree that is then repeatedly queried for neighbors for each wedge-head center. Some interesting constellations, spatial n-grams, may be obscured by other overlapping spatial n-grams or by stray points. We can remove those obscuring points by enumerating all possible k-subsets of each spatial n-gram. While this approach greatly expands the set of spatial n-grams, many of which have no semantic importance, we reduce this set by pruning

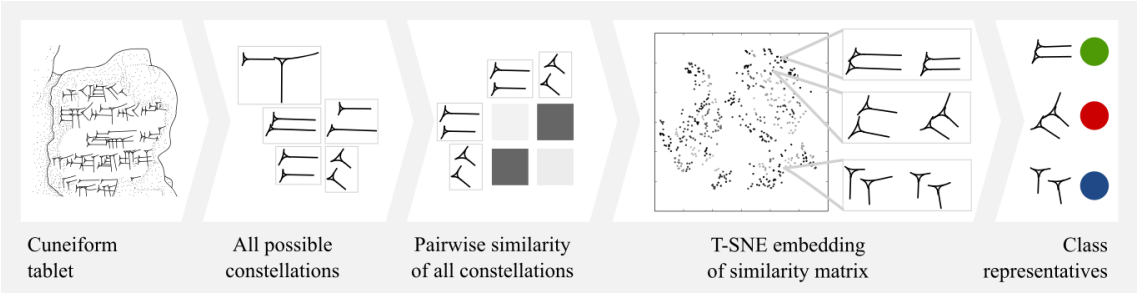


Figure 6.1: Data transformations of the learning stage. The colored points of the class representatives symbolize the spatial n-grams used to decompose queries.

spatial n-grams that are not common. The commonness of a spatial n-gram is determined by it being an inlier in the subsequent clustering.

Given the similarity metrics defined in Chapter 4, Section 4.5, pp. 81 we have a notion of distance between spatial n-grams. This allows us to employ hierarchical clustering algorithms [RM05] that only require a distance matrix between the samples. Hierarchical clustering uses various linkage criteria. Since the extracted spatial n-grams come from a noisy source, damaged cuneiform tablets, we use a very robust linkage criterion, complete linkage, for clustering.

Other clustering methods, such as k-means [Mac67] and DBSCAN [Est+96], require the samples to be embedded into metric space. We use embedding methods such as Multi-Dimensional Scaling (MDS) [BG05] and TSNE [MH08]. Then, we cluster the resulting space with DBSCAN to make sure that definite outliers, like the generated meaningless spatial n-grams, are not part of the resulting clusters. From the resulting cluster we choose spatial n-grams that are most central, their mean distance to all other spatial n-grams in that cluster is minimal. Those are then the common spatial n-grams, we denote these as class representatives. The pipeline of this learning stage is illustrated in Figure 6.1.

We search a query constellation by decomposing it into known class representatives. All possible constellations are extracted from the search query and compared to the representatives. Then, locations on the tablet are computed where constellations from all classes are present that are contained in the query. A cuneiform tablet is represented as a point cloud where each point is labeled with cluster of constellations it belongs to. The query is located by performing a nearest neighbor search so that all classes of the query are present in the

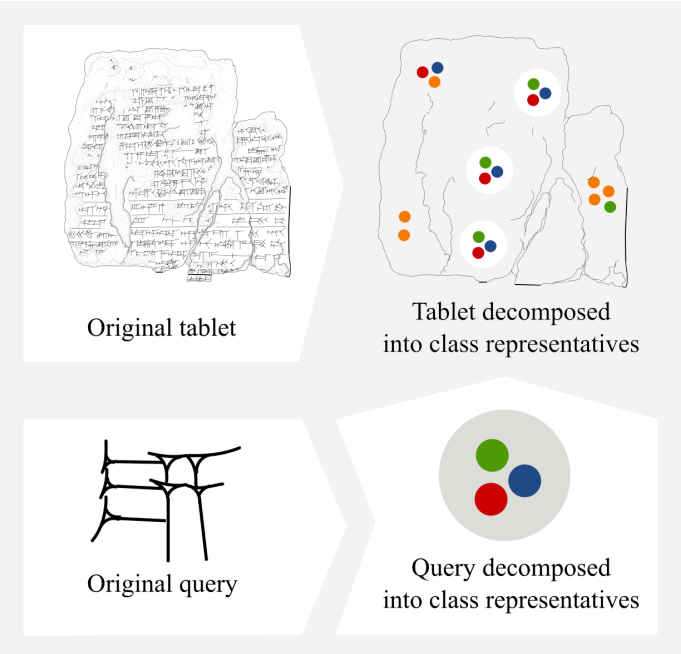


Figure 6.2: Data transformations in the retrieval process. Spatial n-grams are extracted from tablet and query. Only location on tablet containing all spatial n-grams are matched exactly. The colored points symbolize these n-grams.

retrieved search. Then, for each of the computed locations an exact matching using the similarity metrics presented in Chapter 4, Section 4.5, pp. 81 is performed. This process is illustrated in Figure 6.2.

6.2.3 Evaluation of the Mined Spatial n-Grams

We preliminarily evaluate our method on our real-word dataset of traced cuneiform tablets. First, the embedding and clustering methods are trained on the dataset to find an embedding and cluster centers. Then, the query is transformed according to our pipeline and located on the tablet. The query is a random wedge constellation, not correlated to grammatical structures, that is spotted on tablet without the need of prior segmentation. The repeating patterns in Figure 6.3 were autocratically marked by our framework. In future work we will validate of our method. Currently, we manually segment and tag tablets to create ground truth. We also evaluate different embedding and clustering methods for the pipeline.

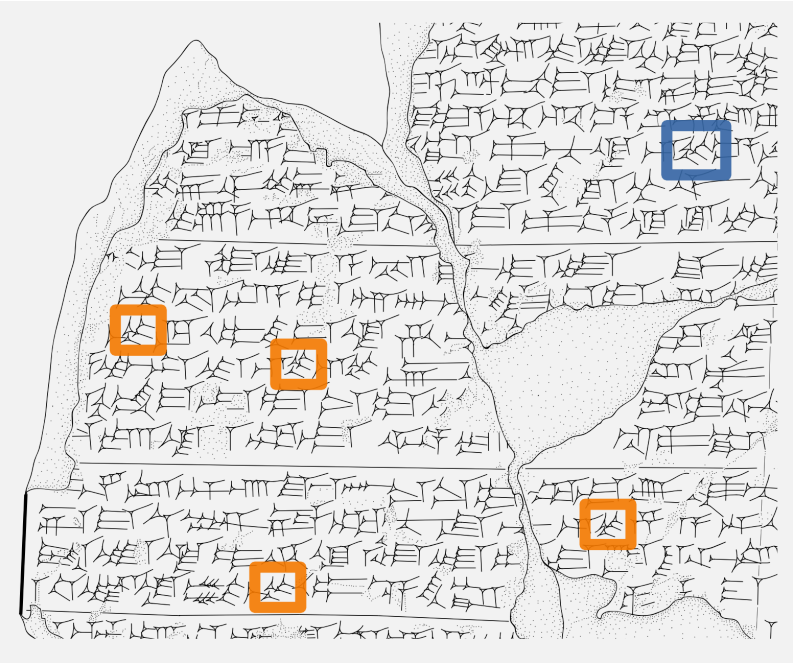


Figure 6.3: Retrieval results of an example query. The query has a blue bounding box, the spotted cuneiform signs have an orange bounding box.

6.3 Automating Transliteration from Parallel Sentences

Currently, cuneiform databases like the Cuneiform Commentaries Project (CCP, `http://ccp.yale.edu/`) provide raster-images of cuneiform tablet tracings and associated transliterations. The library cannot be searched using wedge constellations as queries. Only the available transliterations can be searched using Latin query words. An automatical generation of transcription would greatly expand the corpus of data that can be searched, even if it was done only partially. Additionally, automatically generated transcriptions could be vetted by expert Assyriologists to expand the ground truth.

Therefore, we present a first approach towards automated learning of transliterations of cuneiform tablets based on a corpus of parallel sentences. To our knowledge there is no prior work on the automatic transliteration of cuneiform script. There is little work on automatic transliteration in general and the challenge is usually approached by learning a mapping from a sequence of discrete tokens to another set of discrete tokens, a form of restricted machine translation.

Kang et al. [KC00] present a method for automatically transliterating English words into Korean phonemes. Since their method is unsupervised, they modify Covingtonś algorithm [Cov96] to learn an alignment of tuples of English and Korean words. Their method works on a character level and learns decision trees for the individual characters to transliterate and back-transliterate unobserved words.

Al-Badrashiny et al. [AB+14] approach the challenge by aligning a corpus of parallel words with GIZA++ [ON03]. The alignments are then used to build a Finite State Transducer from sequences of Arabizi characters to sequences of Arabic characters. Sajjad et al. [Saj+11] uses rule-based equivalence and the Levenshtein distance [Lev65] to align a parallel corpus of Urdu and Hindi. Then, a most likely transliteration is found by deriving character to character probabilities from the alignments.

Our transliteration task is significantly more complex, as we work on parallel lines where many words are present and with raster data, where features are continuous (instead of discrete tokens of language) and noise introduces false positives.

We derive our dataset by automatically scraping the CCP website for data. Since cuneiform script is three dimensional, a single photograph with a single direction of lighting is not sufficient for reading the tablet. A line-tracing is a manually created drawing on the basis of the original artifact, usually accompanying cuneiform tablets to ease deciphering.

6.3.1 Background on Ground Truth Extraction

For around 130 tablets of 1000 tablets, the CCP provides both a line-tracing and the corresponding transliteration. To avoid introducing false positives, tablets where the count of line-tracings and transliteration pages mismatch, where transliteration references have dead links, and where line-tracings or transliterations lack meta-data clearly identifying a correspondence (no data on which side is obverse) are all discarded. This leaves us with 30 tablets for analysis and learning.

Transliterations and translations provided by the CCP are heavily annotated with additional cross-references to word-stems and alternate word-forms. Figure 6.4 visualizes a cuneiform tracing and the accompanying transliteration.

While, at first sight, such a range of annotations seems favorable to a learning task, the amount of relationships between forms of denoting cuneiform signs create ambiguities. Not all transliterations are annotated or correctly annotated, some lead to non-existing documents. Some relationships are only implied by the authors while others are overly expansive. We choose to use the directly annotated and longer written forms instead of following a chain of relationships.

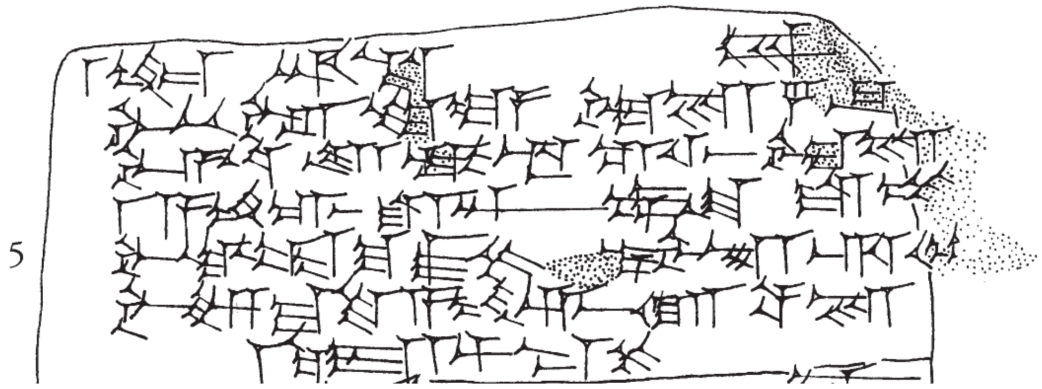
The Assyrian language has a complicated grammar and numerous derivations where suffixes, prefixes and even infixes are added to verbs. This results in many different readings of single sets of wedges. Experts may decide to transliterate bigger cuneiform signs or their constituents, both are valid decisions. We make use of the annotations to split all transliteration tokens to its most basic and shortest radicals.

There is no whitespace between cuneiform signs. It is therefore very hard to infer word boundaries, since any sequence of words may look very similar to any other sequence of differently stemmed words. Only from understanding the topic of the document and the grammatical case of a sentence can word boundaries be inferred. Word segmentation is thus not tractable without a language model.

6.3.2 Feature Extraction

Our learning task requires sequences of transliteration tokens paired with sequences of feature-vectors from the raster-image data. We extract fixed-size feature-vectors by segmenting lines and moving a sliding window over the segmented lines to extract HoG (Histogram of oriented Gradients [DT05]) feature descriptors.

Line Segmentation The cuneiform tablets in our dataset are historically young and therefore written in well separated lines and from left-to-right. The common approach of computing a projection profile [San+09] and segmenting lines centered around local peaks provides good results. We normalize line heights to 40 pixels height, typical heights in our dataset range between 20 and 38 pixels.



Vs. [I]NE-[d]*Iš*$_8$*-tár* BAN
[f]*Be-el-tu-ia* DAM-*su* *šá* KI[N]
[f]*A-na-ru-qi-al-si-ši* DAM-*su* [*šá* KIN]
[I]*A-am-mar-ša*-DINGIR DUMU-*šu šá* GABA
5 [f]*La-ka-ma-áš-še* ⌜DUMU⌝.MUNUS-*su tal-mi-*[*tu*]
[f]*Im-ma-te-a-am-mar* KI.MIN *pír-su*
1 ANŠE 2 1/2 SÌLA ŠE

Figure 6.4: Excerpt from a cuneiform tablet (TCH 92.G.127) (top) and its transliteration [Jak09] (bottom). Areas with ink points denote tablet damage. Transliteration contains both radicals (lowercase) and fully identified letters (uppercase).

The line-tracings are often annotated in-band, that is, Latin text inside the raster-images. Those annotation do not disturb the line segmentation but add lines from the top or at the bottom of the cuneiform tablet. We manually remove tablets from the dataset with annotation at the top and automatically discard lines that exceed the count of transliteration lines.

Descriptors We extract HoG feature descriptors by sliding a 40-by-8 pixels window with 5 pixel increments horizontally over the segmented lines. Each window contains 16 4-by-4 pixel HoG cells with 9 orientations each. Different parameters were tried, 2-by-2 or 8-by-8 HoG cells and 4 or 10 pixel wide windows, the chosen parameters performed best on our dataset. This results in a 144 dimensional feature-vector for each slice of a segmented line. We denote a feature descriptor on line k at slice position l with $x_{k,l}$.

Outlier Removal Cuneiform tablets are between 1000 and 3000 years old weathered stone artifacts that are damaged, broken or misshapen. Assyriologists annotate outline, damage, white-space and unreadable signs by dense crosshatching. Those areas are have to be removed from the extracted set of slices.

Whitespace is removed by thresholding the sum of pixels in a slice. Slices with more than 95% (experiments with 80% or 90% performed worse) white background color are rejected. We assume that crosshatching has an unusual count of edges compared to written cuneiform. We extract edges from a slice with the Canny Edge Detector [Can86] and sum those per column and per row. The resulting set of features is modeled using a Gaussian distribution. Slices that have a log-probability of being part of the estimated distribution lower than -50 are marked as outliers and removed from the dataset. We additionally experimented with lower (-30, -40) and higher values (-60, -70). The result of outlier removal is shown in Figure 6.5.

6.3.3 Sequence Learning

Given parallel sequences of features and labels, HoG descriptors and transliteration tokens, the challenge of learning automated transliteration is transformed into a sequence labeling task. Learning a reordering between features and labels is not necessary since transliterations proceed in the same sequence

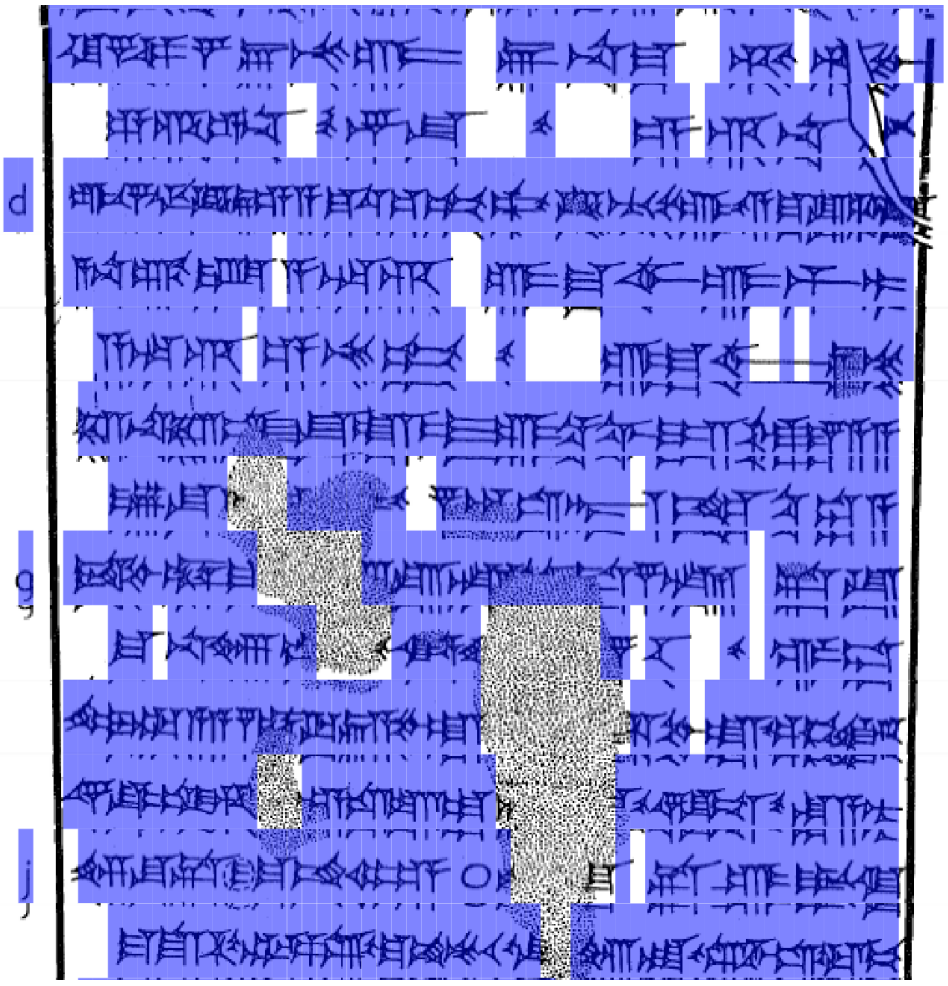


Figure 6.5: An excerpt of a cuneiform tablet. Areas highlighted in blue have been accepted after passing background thresholding and outlier detection. Whitespace, tablet shape markings and tablet damage has been successfully left out. Annotations in Latin to the right have been erroneously accepted.

Figure 6.6: Visualization of the data alignment process. An extracted line from a cuneiform tablet, the associated HoG (Histogram of oriented Gradients) feature descriptors, the profile feature descriptors and the aligned and normalized transliteration tokens are shown. Some slices are not annotated since those have been rejected as outliers.

as their source, unlike translations where a reordering has to be estimated. Therefore, we evaluate various methods used for sequence classification.

Generating Training Data The classifiers evaluated in this work require the sequence of labels and the sequence of feature descriptors to be of equal lengths, each feature descriptor is paired with a label. In our case, the count of emissions (feature descriptors of slices) per line is significantly higher than the count of labels (transliteration tokens). We stretch the label sequence to the length of the emission sequence by repeating labels.

Original word labels are denoted with $y_{k,m}$ on line k and word position m where $|y_{k,\cdot}|$ is the count of labels on a line k. Stretched labels are denoted as $\hat{y}_{k,l}$ and $|\hat{y}_{k,\cdot}|$ is the count of stretched labels on a line k.

$$\hat{y}_{k,l} = y_{k,m} \text{ where } k = l * \frac{|\hat{y}_{k,\cdot}|}{|y_{k,\cdot}|}$$

We assume that the lengths of the transliteration tokens correlate with the lengths of cuneiform signs. This assumption is valid as we split the written forms of the signs into short radicals (comparable to consonants in Latin). Wrong transliterations, damaged areas in the tablet, and annotations complicate this assumption. Figure 6.6 shows this repetition of labels.

Point-wise Classification With point-wise classification of sequences we learn and predict labels independent of surrounding context. While the

modeling power of this approach for sequence labeling is limited, we can make use of common classification methods. We evaluate a k-Nearest Neighbor (KNN) [Alt92] approach and a multi-class C-Support Vector Machine (SVC) [Sch+00]. First, we reduce the dimensionality of the feature descriptor using Principal Component Analysis (PCA) [Pea01]. We experimented with different values for the target dimensionality $(2, 5, 10, 50, 100)$ and reached an optimal value of 5.

Sequence Classification In sequence classification the label of a feature descriptor is dependent on its context and complete sequences are predicted at once. We evaluate a generative approach modeling with a Hidden Markov Model (HMM) [Bau+70] and a discriminative approach with the SVM-HMM framework by Joachims et al. [JFY09].

For the HMM approach, our a-priori model consists of starting probabilities s_i, transition probabilities $\phi_{i,j}$ and means μ_i and covariances σ_i^2 with a set of labels $i, j \in L$. The set $\{x_{k,l}|y_{k,l} = i\}$ denotes the feature vectors x that were assigned the label i. The emissions features of the HMM are modeled as a Gaussian distribution.

$$
\begin{aligned}
s_i &= \frac{|\hat{y}_{\cdot,0} = i|}{|\hat{y}_{\cdot,0}|} \\
\phi_{i,j} &= \frac{|\hat{y}_{k,n} = i \wedge \hat{y}_{k,n+1} = j|}{|\hat{y}_{\cdot,\cdot}|} \\
\mu_i &= \text{mean}(X_i) \\
\sigma_i^2 &= \text{cov}(X_i, X_i) \\
X_i &:= \{x_{k,l}|y_{k,l} = i\}
\end{aligned}
$$

Then, for a sequence of feature descriptors we find a path with highest probability through the modeled states using Viterbi [Vit67] decoding. Smoothing, by reserving a probability mass for unobserved transitions, did not improve results. No a-priori model for the SVM-HMM framework is specified. The penalty parameter for the structured SVM is set to 10 where $(0.1, 1, 10, 100)$ have been tried. Both of these methods did not benefit from prior dimensionality reduction.

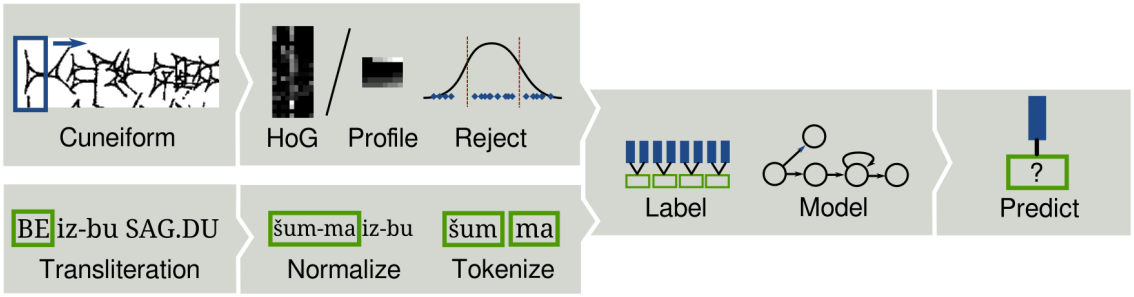


Figure 6.7: The pipeline of our presented automatic transliteration approach consists of four stages. From left to right, data extraction (web scraping, line segmentation and line slicing), Feature pre-processing (normalization, tokenization, descriptor computation and outlier detection), training (data alignment and model fitting) and prediction.

6.3.4 Discussion of the Generated Transliterations

Our transliteration pipeline is thus as follows. First, scraped cuneiform tablet tracings are segmented into lines and sliced into fixed size windows. Then, outliers are removed from this dataset. Transliteration tokens are normalized and tokenized into radicals. These two sets of parallel data are aligned by repeating tokens. One of the four presented models are trained on the aligned data. Finally, a transliteration can be predicted from unseen data either point-wise (each feature descriptor individually) or by whole sequence (all feature descriptors of a line at once). Figure 6.7 illustrates this process.

Experiment setup To evaluate our transliteration pipeline, we employ a 10-fold sequence aware cross-validation scheme. The training and test folds are randomly populated with continuous sequences (lines of cuneiform) so that the count of samples (HoG feature descriptors), not the count of sequences, is proportional to the split of train and test data.

We use two different sets of metrics to measure the classification performance of our methods: (i) Point-wise metrics consider only slices without context and evaluate the zero-one loss for each sample in the test set individually. (ii) Sequence metrics evaluate the zero-one loss for complete sequences, that is, each sample in a sequence must be correctly classified to consider such a sentence correct as well. Good results are hard to achieve on this metric if we take into account that sequences have usually more than 100 samples and all

Method (Parameters)	Point-wise	Sequence
Baseline	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.00 \pm 0.00}$
KNN ($k = 100$)	0.04 ± 0.01	0.00 ± 0.00
SVC ($C = 0.1$)	0.05 ± 0.01	0.00 ± 0.00
HMM	0.06 ± 0.03	0.01 ± 0.02
SVM-HMM	0.02 ± 0.01	0.00 ± 0.00

Figure 6.8: Accuracy scores for point-wise prediction and whole sequence prediction after 10-fold cross validation. Best performing parameters for each method are given.

of these need to be correctly identified. We also include a baseline estimator that provides a point-wise stratified random guess for each sample.

Evaluation Figure 6.8 summarizes the performance of the evaluated methods. An evaluation of the parameters of the classifiers is given in Figure 6.9. The best point-wise classifier, SVC, is five times better than the random baseline. Given the amount of noise in our dataset, a lenient mis-classification penalty, low value of the parameter C, provides best results. For the same reason, the KNN classifiers perform best with a high count of neighbors used for classification providing smooth results. The sequence classification methods do not significantly outperform the point-wise classification methods. Nevertheless, the HMM model is the only classifier which was capable of correctly transliterating at least some of the sentences in the test data and performs six times better than the random baseline. An a-priori language model of the transliteration allowed the HMM to outperform the structured SVM on our dataset.

The biggest challenge in our transliteration task proved to be alignment of cuneiform signs to the respective tokens in the transliteration. Figure 6.10 shows that the classes (transliteration tokens) in our dataset have only few data-points assigned to them and the assigned data-points do not repeat. Too many false-positives are present in the ground truth for efficient learning. Due to the performance of the HMM approach on whole sequences, we conclude that cuneiform transliteration is tractable but requires significantly cleaner data. The alignment process has to be either performed manually by experts or learned, e.g. using the common Expectation Maximization (EM) [DLR77] method.

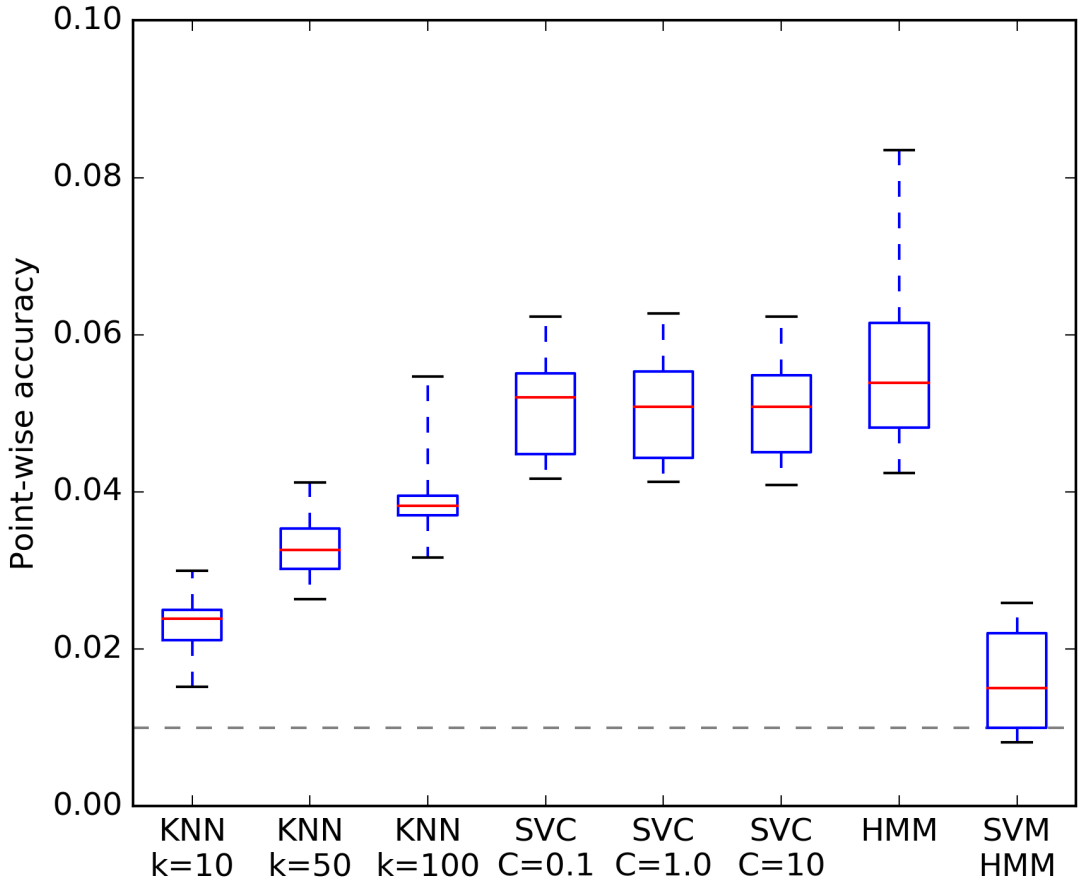


Figure 6.9: Accuracy score for point-wise prediction over various parameters. The gray stippled line indicates mean accuracy of the the baseline estimator.

Figure 6.10: Transliteration tokens and assigned slices. Each token has only few assigned slices and those often constitute only one example. This is not enough training data to learn many of the tokens used to describe cuneiform sign.

6.4 Summary

In this work, published in [BKM17], we presented a first approach for automatic transliteration of cuneiform script by learning sequence classifiers on a dataset of parallel lines. We employed a pipeline of feature extraction, tokenization, outlier detection, and subsequent classification of written cuneiform to predict labels.

The main challenge is the sparsity of data to reliably train a classifier. Only 30 tablets were annotated with identifiable transliterations, tablets were damaged, misshapen and contained in-band annotations that need to be reliably segmented before aligning. Additionally, the transliteration did not use a uniform notation to denote identically written cuneiform script. The Machine Translation and Automated Analysis of Cuneiform (MTAAC, `https://cdli-gh.github.io/mtaac/about/`) project has been initiated to standardize the format of annotations and cuneiform tablet data and provide structured ground-truth for high-level computational analysis. Homogenizing cuneiform tracings [BMM15] into a unified description of wedges would also alleviate the challenge of multiple annotation and tracing styles.

Nevertheless, we successfully segmented outliers and we were able to demonstrate that some structure has been learned by showing that a sequence classifier outperformed point-wise classifiers. In our future work we aim to obtain a reliable segmentation of the cuneiform into words, not lines, performed manually by experts and a uniform style of transcription also performed by experts.

7 Outlook and Conclusion

In this work, we cross-linked research in Assyriology to computer science and computer linguistics by laying the groundwork for computational analysis of cuneiform script. We developed novel foundational tools for linguistic processing of cuneiform. To prove the applicability of our framework we performed linguistic pattern analysis of cuneiform which was thus far infeasible.

Our work paves the way for quantitative research on cuneiform and consists of four major contributions. Each step builds upon the last and establishes a pipeline of increasing power of abstraction and reasoning capabilities.

Uniform inter-operable feature vector description Cuneiform tablets and transcriptions are acquired from disparate sources in heterogeneous representations. We unify these to a common and shared representation suitable as a foundation for computational research.

Semantically robust similarity measures Cuneiform signs are highly variable, but not all variations are uniquely meaningful. We developed distance functions modeling semantic similarity for cuneiform script. Thus, we were the first to introduce a computational notion of semantic equality and inequality of cuneiform signs.

Segmentation-free sign spotting The fluid concept of words and lack of whitespace in cuneiform script makes segmentation very challenging. We circumvent brittle grammatical analysis for word separation by adapting segmentation-free part-structured spotting with our wedge feature representations and distance functions. We enable free-form cuneiform sign search without any assumptions to tablet layout. With this development we provide a crucial tool and the groundwork for researchers in Computer Linguistics and Assyriology to analyze and gain new insights into statistical properties of cuneiform script.

Application of our foundational tools for analysis For the lack of any computational tools for cuneiform script analysis, quantitative research was so far not possible. We applied our framework for computational analysis and gained new insights into patterns of wedge constellations which were previously unattainable.

Cuneiform script is an especially challenging handwriting due to its highly variable three dimensional signs necessitating the use of manual transcriptions which, in turn, introduce their own set of idiosyncrasies. We tackled these challenges by applying and evaluating a multitude of methods, in particular, the following:

CVWW2015 We analyze the suitability of graph-based representations of cuneiform scripts with **Graph Kernels**, the **Spectral Decomposition**, **Random Walks** and **Delaunay triangulation**.

HIP2015 We present a unified shared description of cuneiform script based on an explicit representation of wedge-shaped impressions and methods for extracting these by means of **Over-segmentation** and **Heuristic Conflict Resolution**.

ICDAR2015 We introduce distance functions for our shared descriptions of cuneiform script by evaluating **Projection Profiles** with **DTW**, **ICP**, **HMM**.

DAS2016 We prove the applicability of our shared description by mining frequent patterns of wedge constellations by applying **TSNE** and **k-Means clustering**.

ICFHR2016 We adapted part-structured spotting together with Howe [How13] with our shared description and new distance functions, **GMM** and **Bag-of-properties**, and evaluate to a HMM model.

ICDAR2017 We develop an approach towards automatic transcription of retro-digitized tracings by evaluating models using **HMM**, **Structural Support Vector Machine (S-SVM)** and **KNN**.

Additionally, we summarized our findings with and analyzed their applicability for the Digital Humanities community in the following publications: [MB15] and [BM17]. The multitude of different mathematical tools and machine

learning methods we were able to apply to analyze cuneiform script further underscores the generality and practical usefulness of our methods.

7.1 New Research Questions

During our research we discovered further directions and new areas in which our methods are capable of being applied. Especially, our tools allow the formulation of new research question.

Which novel research questions can now be posed?

Apply our foundational tools for advanced linguistic analysis. By developing computational models and tools for cuneiform script, we enabled the first quantitative research and development of new statistical insights. The wealth of computer linguistic methods now made applicable to cuneiform enables for the formulation of research questions in the, up to now, unreachable frontier of Digital Assyriology.

Which further complex scripts are prime for computational research?

Transfer our approach to modeling of cuneiform to the next, just as complex and computationally inaccessible, historic writing script. Our approach to understanding and modeling cuneiform by decomposing it into a hierarchy of basic constituents based on tool-usage, i.e. wedge-shaped impressions and cuneiform signs, is applicable to other highly complex languages, such as Maya script [Fel+17]. The methods we developed for cuneiform script analysis are also capable providing novel insights into different historic scripts.

Which new insights can be gained by cross-referencing disparate sources of cuneiform?

Extract our common shared wedge feature description from more different cuneiform sources, such as photographs of original tablets. Cuneiform tablets and tracings are available in plentiful, disparate and mutually incompatible formats. With our work, we lifted born-digital tracings into a computationally analyzable representation. With each additional source of cuneiform made available to our shared wedge feature description, new opportunities arise for cross-source analyses.

7.2 Summary

Cuneiform tablets appertain to the oldest textual artifacts, in extent comparable to texts written in Latin or ancient Greek. Since those tablets were used in all of the ancient Near East for over three thousand years [Sod94], interesting research questions can be answered regarding the development of religion, politics, science, trade and climate change [Kan+13]. These tablets were formed by clay and written on by impressing a rectangular stylus [Bor04] into a wet clay tablet, leaving a wedge-shaped impression.

The digitization of cuneiform tablets and the development of databases provides us with open access corpora of photographs, line tracings, transliterations and translations. Major examples are the Cuneiform Digital Library Initiative (CDLI, `http://cdli.ucla.edu/`) and the Open Richly Annotated Cuneiform Corpus (ORACC, `http://oracc.museum.upenn.edu/`).

Cuneiform tablets are acquired by different means, including 3D-scanning original artifacts, flatbed scanning tracings and digital tracing of originals. We extracted wedges from born-digital tracings of cuneiform tablets by splitting documents into independent components and over-segmenting triangular shapes. Then, we presented a novel approach to extract our wedge model, from densely written and self-intersecting cuneiform script, by representing the pattern matching process by an optimal assignment task that can be efficiently solved by the Hungarian algorithm.

From the extracted wedges, we derived two unstructured, using graphs, and three structured, using feature vectors, wedge representations and developed

associated similarity metrics. We evaluated these methods on a spotting task with a-priori segmented cuneiform signs. The results have been compared using a precision-recall graph. We found that a six keypoint representation with a Euclidean distance similarity metric gave the most accurate results.

On the basis of our wedge descriptors we adapted part-structured models to enable segmentation-free spotting of wedge constellations. Additionally, we implemented a sequential HMM, the state-of-the-art in Latin word-spotting, for use on our dataset and evaluated both on a test dataset. Comparing results with a precision-recall graph, we find that our approach of combining part-structured spotting with our six keypoint model and its associated similarity metric achieved superior performance on the spotting task.

Finally, we presented two different applications of our methods to prove the applicability of our computational analysis tools for cuneiform. We extracted frequent constellation of cuneiform from a set of cuneiform tablets that match the basic constituents, radicals or word-stems, of cuneiform as defined by experts. We also present a first approach to automatic transliteration of cuneiform script. Lines of cuneiform script are extracted from tracings by layout analysis and line segmentation. Then a sequential HMM is trained on parallel sentences of slices of HoG transformed signs and the associated transliterations.

* * *

In conclusion, we developed and evaluated several ways of processing and manipulating cuneiform script computationally resulting in a set of tools that serves as a basis for near future computational research in Assyriology. Our contributions to the frontier of research in the Digital Humanities open a completely new research focus where cuneiform script can by analyzed quantitatively.

Notation

S	Set of strokes
$S_1 \cdots S_k$	Sets of strokes partitioned from S
$s_i, s_j \in S$	Strokes in the set S
$I_S \in S^2$	Set of intersecting strokes in S
$\|S\|$	Cardinality (size) of a set
$\text{center}(s_i, s_j) \in \mathbb{R}^2$	Center of intersection area
$\text{endpoints}(s) \in (\mathbb{R}^2, \mathbb{R}^2)$	Endpoint of a stroke
$\vec{ab}$	Vector from a to b
$\|\|a\|\|$	Euclidean norm
V	Set of vertices
E	Set of edges
L	Set of labels
$G = (V, E)$	An undirected topological graph
$P(S)$	Power-set of a set
$W \subset P(S)$	Set of all wedges
$w \in W$	Wedge consisting of a set of strokes
$w^h \subset w$	Set of strokes for the wedge-head
$w^a \subset w$	Set of strokes for the wedge-arms
$F^K \subset \mathbb{R}^{12}$	Set of all keypoint feature vectors
$f^K \in F^K$	A keypoint feature vector
$(h, a) = f^K$	Feature subsets for the wedge-head and wedge-arms
$v \in f^k, v \in \mathbb{R}^2$	2D-Vector projections of keypoints
$F^P \subset \{0, 1\}^{24}$	Set of all bag-of-properties feature vectors
$f^P \in F^P$	A bag-of-properties feature vector
$F^T \subset \mathbb{R}^5$	Set of all GMM feature vectors
$f^T \in F^T$	A GMM feature vector

$A = \{0,1\}^{S \times W}$	Assignment matrix from strokes to wedges
$C = \mathbb{R}^{S \times W}$	Cost matrix of assigning strokes to wedges
$d(x, y) \in \mathbb{R}$	Distance between two objects
$k(x, y) \in \mathbb{R}$	Kernel (dot-product) between two objects
$d^{\mathrm{DTW}}(I, I')$	DTW distance for profile features
$d^{\mathrm{Direct}}(X, X')$	Nearest neighbor distance between point-clouds
$d^{\mathrm{ICP}}(X, X')$	ICP distance between point clouds
$k^{\mathrm{Subtree}}(G, G')$	Sub-tree graph kernel
$k^{\mathrm{Randomwalk}}(G, G')$	Random-walk graph kernel
$k^{\mathrm{Spectral}}(G, G')$	Spectral graph kernel
$D = \mathbb{R}^{m \times n}$	Assignment costs between two sets of wedges
$C \subset W$	Constellation of wedges
$[C] \subset P(W)$	Set of all permutations of C
$[C](p) \in P(W)$	p-th permutation of C
$\hat{t}^P$	Property vectors of the property feature descriptor
$\theta_{1 \dots k}$	Gaussian mixture parameters for k components
$\hat{\theta}$	Maximum likelihood estimation of parameters
$\mu_{1 \cdots k}$	Means of k Gaussians
$\sigma^2_{1 \cdots k}$	Squared variances of k Gaussians
$\Sigma \in \mathbb{R}^{k \times k}$	Covariance matrix of aforementioned Gaussians
$\mathcal{N}(\mu_i, \Sigma_i)$	i-th multi-variate Gaussian distribution
$p(\theta \mid X)$	Probability of parameters θ given data X
$d^K(f^K, f'^K) \in \mathbb{R}$	Keypoint feature distance
$d^P(f^P, f'^P) \in \mathbb{R}$	Bag-of-properties feature distance
$d^T(f^T, f'^T) \in \mathbb{R}$	Template feature distance

List of Acronyms

BoVW Bag-of-Visual-Words 108–110, 112, 122, 148

DTW Dynamic Time Warping 64–66, 68–70, 98, 100, 102, 142, 148

E-SVM Exemplar Support Vector Machine 109, 148

EM Expectation Maximization 92, 112, 148

GDT Generalized Distance Transform 148

GMM Gaussian Mixture Model 61, 94, 108, 142, 147, 148

HMM Hidden Markov Model .. 61, 65, 98, 103, 106–113, 122, 142, 145, 148

HoG Histogram of oriented Gradients 100, 108, 109, 145, 148

ICP Iterated Closest Points 79–81, 83, 102, 142, 148

KNN K-Nearest Neighbors 61, 142, 148

LSA Latent Semantic Analysis 108, 148

MDS Multi-Dimensional Scaling 126, 148

MLE Maximum Likelihood Estimation 92, 148

OCR Optical Character Recognition 107, 148

PSS Part-structured Spotting 148

S-SVM Structural Support Vector Machine 142, 148

SIFT Scale Invariant Feature Transform 108–110, 148

SPM Spatial Pyramid Matching 108, 148

SVG Scalable Vector Graphics 14, 26, 27, 32, 148

SVM Support Vector Machine 59, 61, 62, 109, 148

TSNE t-stochastic neighborhood embedding 59, 124, 126, 142, 148

XML eXtensible Markup Language 14, 148

List of Terms

3D-acquired tablet A cuneiform tablet acquired using a structured light scanner. The result is a polygonal mesh describing its surface.. 34, 43, 148

arm Rifts extending from the wedge-head created by the impressed corners of a rectangular stylus used for writing. See also wedge-arm. 148

born-digital tracing Tracing of a cuneiform tablet created digitally by means of a vector graphics editor resulting in vector graphic, e.g. the Scalable Vector Graphics format. 24–27, 31, 32, 34, 37, 39, 41, 43, 44, 46, 54, 56, 77, 144, 148

constellation A set of wedge-shaped impressions. See also wedge constellation. 69, 114, 122, 148

cuneiform Historic writing system used in ancient near east. Writing was performed by impressing wedge-shaped impressions in wet clay tablets. See also cuneiform script. 20, 33, 44, 148

cuneiform script Historic writing system used in ancient near east. Writing was performed by impressing wedge-shaped impressions in wet clay tablets. 20, 32–34, 36, 43, 54, 55, 59, 60, 105, 123, 148

cuneiform sign A named and known set of wedge-shaped impressions that carries some meaning.. 19–22, 25, 31–34, 36, 38, 43, 56, 59, 60, 66, 67, 70, 74, 80, 81, 83, 86, 95, 96, 98–102, 105, 107–110, 114, 116, 120, 122, 124, 125, 128, 130, 134, 137, 139, 141, 143, 145, 148

cuneiform tablet Wet clay tablet used as writing medium in the ancient near east. 23, 33, 54, 105, 108, 109, 124, 148

head Inner triangular part of a wedge. See also wedge-head. 148

keypoint Endpoints of stroke in a cuneiform tracing or center of the intersection area of two strokes. . . . 10, 32, 36, 38, 41, 43, 44, 53–56, 59, 62–64, 77–79, 81, 87, 88, 91–94, 96, 122, 148

keypoint feature Twelve-dimensional feature vector used to describe wedge-shaped impressions. See also keypoint feature descriptor. 91, 148

keypoint feature descriptor Twelve-dimensional feature vector used to describe wedge-shaped impressions. 43, 44, 53, 56, 63, 64, 90, 91, 95, 100, 101, 106, 124, 148

point-cloud Here, a set of two-dimensional points representing a constellation of wedges. 79, 148

property feature descriptor Describes wedge-shaped impressions by means of a twenty four dimensional bit-vector of properties present or absent in a particular wedge. 44, 90, 148

retro-digitized tracing Flat-bed scans of cuneiform tablet tracings that have been created with pen and paper. 24, 31, 32, 43, 142, 148

sign A named and known set of wedge-shaped impressions that carries some meaning. See also cuneiform sign. 33, 34, 36–38, 44, 54, 56, 59, 60, 64, 70, 74, 85, 99, 101, 105, 107, 116, 119, 123, 125, 132, 134, 141, 145, 148

spatial n-gram Small collection of low-dimensional points with associated high-dimensional feature vectors. 125, 126, 148

spline A parametric curve description based on polynomial equations. . 109, 148

stroke A hand drawn line on a tracing of cuneiform tablet. Many strokes are used to draw wedge-shaped impression. On born-digital tracings strokes are represented by closed spline paths. 28, 34, 36, 39, 43, 45, 47, 53, 54, 109, 148

tablet Wet clay tablet used as writing medium in the ancient near east. See also cuneiform tablet. 33, 43, 105, 108, 148

template feature descriptor Five dimensional feature vector describing wedge-shaped impressions by their affiliations to a set of learned classes of wedge expressions. 44, 148

wedge An impression left on clay after impressing a rectangular stylus used to write cuneiform. See also wedge-shaped impression. . 20, 33, 34, 36, 43–45, 47, 48, 53–55, 64, 74, 81, 83–86, 96, 101, 107, 124, 125, 148

wedge constellation An arbitrary set of wedge-shaped impressions that is meaningless. 21, 33, 34, 36, 44, 45, 49, 56, 59, 61, 63, 65, 66, 69, 73, 74, 76–85, 87–89, 92, 94, 95, 98, 100, 102, 105, 106, 114, 116, 122–125, 127, 128, 142, 145, 148

wedge-arm Rifts extending from the wedge-head created by the impressed corners of a rectangular stylus used for writing. 19–21, 26, 38, 43–45, 47, 48, 50, 51, 53, 56, 57, 63, 64, 81, 82, 88–91, 102, 148

wedge-head Inner triangular part of a wedge.. 19–21, 23, 26, 38, 43–48, 50, 51, 53, 54, 56, 57, 63, 64, 81–84, 88–91, 102, 116, 125, 148

wedge-shaped impression An impression left on clay after impressing a rectangular stylus used to write cuneiform. 19–21, 31–34, 36, 43, 44, 53, 54, 56, 63, 81, 90, 102, 142, 143, 148

Bibliography

[AB+14] M. Al-Badrashiny, R. Eskander, N. Habash, and O. Rambow. "Automatic Transliteration of Romanized Dialectal Arabic". In: *Conference on Computational Language Learning* (2014).

[AH14] C. C. Aggarwal and J. Han, eds. *Frequent Pattern Mining*. Springer, 2014.

[Alm+14] J. Almazán, A. Gordo, A. Fornés, and E. Valveny. "Segmentation-free word spotting with exemplar SVMs". In: *Pattern Recognition* (2014).

[Alt92] N. S. Altman. "An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression". In: *The American Statistician* (1992).

[AS94] R. Agrawal and R. Srikant. "Fast Algorithms for Mining Association Rules". In: *International Conference on Very Large Data Bases* (1994).

[BA83] H. Bunke and G. Allermann. "Inexact Graph Matching for Structural Pattern Recognition". In: *Pattern Recognition Letters* (1983).

[Bau+70] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains". In: *The Annals of Mathematical Statistics* (1970).

[Bel57] R. Bellman. "Dynamic Programming". In: *Princeton University Press* (1957).

[Ber+08] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008.

[BG05] I. Borg and P. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer, 2005.

[BGM15a] B. Bogacz, M. Gertz, and H. Mara. "Character Retrieval of Vectorized Cuneiform Script". In: *International Conference on Document Analysis and Recognition* (2015).

[BGM15b] B. Bogacz, M. Gertz, and H. Mara. "Cuneiform Character Similarity Using Graph Representations". In: *Computer Vision Winter Workshop* (2015).

[BHM16] B. Bogacz, N. Howe, and H. Mara. "Segmentation Free Spotting of Cuneiform using Part Structured Models". In: *International Conference on Frontiers in Handwriting Recognition* (2016).

[Bic+06] M. Bicego, A. Lagorio, E. Grosso, and M. Tistarelli. "On the Use of SIFT Features for Face Authentication". In: *Computer Vision and Pattern Recognition Workshops* (2006).

[BKM17] B. Bogacz, M. Klingmann, and H. Mara. "Automating Transliteration of Cuneiform from Parallel Lines with Sparse Data". In: *International Conference on Document Analysis and Recognition* (2017).

[Blu67] H. Blum. "A Transformation for Extracting New Descriptors of Shape". In: *Air Force Cambridge Research Laboratories* (1967).

[BM16] B. Bogacz and H. Mara. "Clustering Fundamental Spatial n-Grams for Large Scale Cuneiform Search". In: *Document Analysis Systems* (2016).

[BM17] B. Bogacz and H. Mara. "Codicology and Palaeography in the Digital Age". In: Books on Demand, 2017. Chap. Automatable Annotations - Image Processing and Machine Learning for Script in 3D and 2D with GigaMesh.

[BM92] P. J. Besl and N. D. McKay. "A Method for Registration of 3-D Shapes". In: *Pattern Analysis and Machine Intelligence* (1992).

[BMM15] B. Bogacz, J. Massa, and H. Mara. "Homogenization of 2D & 3D Document Formats for Cuneiform Script Analysis". In: *Workshop on Historical Document Imaging and Processing* (2015).

[Boo01] C. de Boor. *A Practical Guide to Splines*. Springer, 2001.

[Bor04] R. Borger. *Mesopotamisches Zeichenlexikon*. Ugarit-Verlag, 2004.

[Bor86] G. Borgefors. "Distance Transformations in Digital Images". In: *Computer Vision, Graphics, and Image Processing* (1986).

[BP66] L. E. Baum and T. Petrie. "Statistical Inference for Probabilistic Functions of Finite State Markov Chains". In: *The Annals of Mathematical Statistics* (1966).

[BS61] F. L. Bauer and J. Stoer. "Absolute and Monotonic Norms". In: *Numerische Mathematik* (1961).

[Cam14] M. Cammarosano. "The Cuneiform Stylus". In: *Mesopotamia* 49 (2014).

[Can86] J. Canny. "A Computational Approach to Edge Detection". In: *Pattern Analysis and Machine Intelligence* (1986).

[Che95] C.-H. Chen. "Lexicon-Driven Word Recognition". In: *International Conference on Document Analysis and Recognition* (1995).

[Chn+14] E. Chng, S. I. Woolley, E. Gehlken, A. Lewis, L. H. Munoz, and T. Collins. "A Collaborative Environment for Assisted 3D Reconstruction of Cuneiform Tablets". In: *Internationl Conference on Virtual Systems and Multimedia* (2014).

[Chu97] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[Cov96] M. A. Covington. "An Algorithm to Align Words for Historical Comparison". In: *Computational Linguistics* (1996).

[CV95] C. Cortes and V. Vapnik. "Support-Vector Networks". In: *Machine Learning* (1995).

[Dee+90] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. "Indexing by Latent Semantic Analysis". In: *Association for Information Science and Technology* (1990).

[Del34] B. Delaunay. "Sur la sphère vide". In: *Bulletin de l'Académie des Sciences de l'URSS, Classe des sciences mathématiques et naturelles* (1934).

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society* (1977).

[DT05] N. Dalal and B. Triggs. "Histograms of Oriented Gradients for Human Detection". In: *Conference on Computer Vision and Pattern Recognition* (2005).

[Est+96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases With Noise". In: *International Conference on Knowledge Discovery and Data Mining* (1996).

[FB80] M. A. Fischler and R. C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *SRI International* (1980).

[Fel+17] F. Feldmann, B. Bogacz, C. Prager, and H. Mara. "Histogram of Oriented Gradients for Maya Glyph Retrieval". In: *Eurographics Workshop on Graphics and Cultural Heritage* (2017).

[FH05] P. Felzenszwalb and D. Huttenlocher. "Pictorial Structures for Object Recognition". In: *International Journal on Computer Vision* (2005).

[FH12] P. Felzenszwalb and D. Huttenlocher. "Distance Transforms of Sampled Functions". In: *Theory of Computing* (2012).

[Fis+14] D. Fisseler, F. Weichert, G. Müller, and M. Cammarosano. "Extending Philological Research with Methods of 3D Computer Graphics Applied to Analysis of Cultural Heritage". In: *Eurographics Workshop on Graphics and Cultural Heritage* (2014).

[FJF17] E. Frahm, E. Jiménez, and M. Frazer. "Commentary on Marduk's Address, Muššu'u, and Udughul". In: *Cuneiform Commentaries Project* (2017).

[FRB10] A. Fischer, K. Riesen, and H. Bunke. "Graph Similarity Features for HMM-Based Handwriting Recognition in Historical Documents". In: *International Conference on Frontiers in Handwriting Recognition* (2010).

[FVS08] B. Fulkerson, A. Vedaldi, and S. Soatto. "Localizing Objects with Smart Dictionaries". In: *European Conference on Computer Vision* (2008).

[Gao+10] X. Gao, B. Xiao, D. Tao, and X. Li. "A Survey of Graph Edit Distance". In: *Pattern Analysis and Applications* (2010).

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[GV00] G. H. Golub and H. A. van der Vorst. "Eigenvalue Computation in the 20th Century". In: *Journal of Computational and Applied Mathematics* (2000).

[Hor87] B. K. P. Horn. "Closed-Form Solution of Absolute Orientation using Unit Quarternions". In: *Optical Society of America* (1987).

[How13] N. Howe. "Part-Structured Inkball Models for One-Shot Handwritten Word Spotting". In: *International Conference on Document Analysis and Recognition* (2013).

[How15] N. Howe. "Inkball Models for Character Localization and Out-of-Vocabulary Word Spotting". In: *International Conference on Document Analysis and Recognition* (2015).

[HT73] J. Hopcroft and R. Tarjan. "Efficient Algorithms for Graph Manipulation". In: *Communications of the ACM* (1973).

[HW11] H. Hameeuw and G. Willems. "New Visualization Techniques for Cuneiform Texts and Sealings". In: *Akkadica* (2011).

[Jak09] S. Jakob. *Die Mittelassyrischen Texte Aus Tell Chuera in Nordostsyrien*. Otto Harrassowitz, 2009.

[JFY09] T. Joachims, T. Finley, and C.-N. Yu. "Cutting-Plane Training of Structural SVMs". In: *Machine Learning Journal* (2009).

[JH02] T. Jost and H. Hügli. "Fast ICP Algorithms for Shape Registration". In: *Joint Pattern Recognition Symposium* (2002).

[Kan+13] D. Kaniewski, E. van Campo, J. Guiot, S. Le Burel, T. Otto, and C. Baeteman. "Environmental Roots of the Late Bronze Age Crisis". In: *PLoS ONE* (2013).

[Kar72] R. M. Karp. "Reducibility Among Combinatorial Problems". In: *Complexity of Computer Computations* (1972).

[KBS11] D. J. Kennard, W. A. Barrett, and T. W. Sederberg. "Word Warping for Offline Handwriting Recognition". In: *International Conference on Document Analysis and Recognition* (2011).

[KC00] B.-J. Kang and K.-S. Choi. "Automatic Transliteration and Back-Transliteration by Decision Tree Learning". In: *International Conference on Language Resources and Evaluation* (2000).

[KMP77] D. E. Knuth, J. H. Morris, and V. R. Pratt. "Fast Pattern Matching in Strings". In: *SIAM Journal of Computing* (1977).

[Kri+05] B. Krishnapuram, L. Carin, M. A. T. Figueiredo, and A. J. Hartemink. "Sparse Multinomial Logistic Regression: Fast Algorithms and Generalization Bounds". In: *Pattern Analysis And Machine Intelligence* (2005).

[Lev65] V. I. Levenshtein. "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals". In: *Soviet Physics Doklady* (1965).

[LLE07] Y. Leydier, F. Lebourgeois, and H. Emptoz. "Text Search for Medieval Manuscript Images". In: *Pattern Recognition* (2007).

[LSP06] S. Lazebnik, C. Schmid, and J. Ponce. "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories". In: *Computer Society Conference on Computer Vision and Pattern Recognition* (2006).

[Mac67] J. B. MacQueen. "Some Methods for Classification and Analysis of Multivariate Observations". In: *Berkeley Symposium on Mathematical Statistics and Probability* (1967).

[Mar12] H. Mara. "Multi-Scale Integral Invariants for Robust Character Extraction from Irregular Polygon Mesh Data". PhD thesis. University Heidelberg, 2012.

[Mar+10] H. Mara, S. Krömker, S. Jakob, and B. Breuckmann. "GigaMesh and Gilgamesh - 3D Multiscale Integral Invariant Cuneiform Character Extraction". In: *International Symposium on Virtual Reality, Archaeology and Cultural Heritage* (2010).

[Mas+16] J. Massa, B. Bogacz, S. Krömker, and H. Mara. "Cuneiform Detection in Vectorized Raster Images". In: *Computer Vision Winter Workshop* (2016).

[Mau17] S. M. Maul. *Das Gilgamesch-Epos neu übersetzt und kommentiert*. C. H. Beck, 2017.

[MB15] H. Mara and B. Bogacz. "A Bridge to Digital Humanities: From Geometric Methods to Machine Learning for Analyzing Ancient Script in 3D". In: *Computer Applications and Quantitative Methods in Archaeology* (2015).

[MGE11] T. Malisiewicz, A. Gupta, and A. Efros. "Ensemble of Exemplar-SVMs for Object Detection and Beyond". In: *International Conference on Computer Visions* (2011).

[MH08] L. J. P. van der Maaten and G. E. Hinton. "Visualizing High-Dimensional Data Using t-SNE". In: *Journal of Machine Learning Research* (2008).

[MHK09] H. Mara, J. Hering, and S. Krömker. "GPU based Optical Character Transcription for Ancient Inscription Recognition". In: *International Conference on Virtual Systems and Multimedia* (2009).

[MK13] H. Mara and S. Krömker. "Vectorization of 3D-Characters by Integral Invariant Filtering of High-Resolution Triangular Meshes". In: *International Conference on Document Analysis and Recognition* (2013).

[MMS08] H. Mara, A. Monitzer, and J. Stöttinger. "Introducing 3D Vision and Computer Graphics to Archaeological Workflow - an Applicable Framework". In: *International Conference on Computer Vision Theory and Applications* (2008).

[MR05] R. Manmatha and J. Rothfeder. "A Scale Space Approach for Automatically Segmenting Words from Historical Handwritten Documents". In: *Transactions On Pattern Analysis And Machine Intelligence* (2005).

[Mun57] J. Munkres. "Algorithms for the Assignment and Transportation Problems". In: *Society for Industrial and Applied Mathematics* (1957).

[ON03] F. J. Och and H. Ney. "A Systematic Comparison of Various Statistical Alignment Models". In: *Computational Linguistics* (2003).

[Pea01] K. Pearson. "On Line and Planes of Closest Fit to Systems of Points in Space". In: *Philosophical Magazine* (1901).

[Rab89] L. Rabiner. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". In: *Institute of Electrical and Electronics Engineers* (1989).

[RB10] K. Riesen and H. Bunke. "Graph Classification and Clustering Based on Vector Space Embedding". In: *Machine Perception and Artificial Intelligence* (2010).

[RM03] T.M. Rath and R. Manmatha. "Word Image Matching using Dynamic Time Warping". In: *Computer Vision and Pattern Recognition* (2003).

[RM05] L. Rokach and O. Maimon. *Data Mining and Knowledge Discovery Handbook*. Springer, 2005.

[RM07] T. M. Rath and R. Manmatha. "Word Spotting for Historical Documents". In: *International Conference on Document Analysis and Recognition* (2007).

[RMC96] E. M. Riseman R. Manmatha C. Han and W. B. Croft. "Indexing Handwriting Using Word Matching". In: *International Conference on Digital Libraries* (1996).

[Rot+15] L. Rothacker, D. Fisseler, G. G. W. Müller, F. Weichert, and G. A. Fink. "Retrieving Cuneiform Structures in a Segmentation-free Word Spotting Framework". In: *Workshop on Historical Document Imaging and Processing* (2015).

[RRF13] L. Rothacker, M. Rusiñol, and G. A. Fink. "Bag-of-Features HMMs for Segmentation-Free Word Spotting in Handwritten Documents". In: *International Conference on Document Analysis and Recognition* (2013).

[RSP09] J. A. Rodríguez-Serrano and F. Perronnin. "Handwritten Word-Spotting using Hidden Markov Models and Universal Vocabularies". In: *Pattern Recognition* (2009).

[Rus+14] M. Rusiñol, D. Aldavert, R. Toledo, and J. Lladós. "Efficient Segmentation-free Keyword Spotting in Historical Document Collections". In: *Pattern Recognition* (2014).

[Saj+11] H. Sajjad, N. Durrani, H. Schmid, and A. Fraser. "Comparing Two Techniques for Learning Transliteration Models Using a Parallel Corpus". In: *International Joint Conference on Natural Language Processing* (2011).

[San+09] R. P. dos Santos, G. S. Clemente, T. I. Ren, and G. D. C. Calvalcanti. "Text Line Segmentation Based on Morphology and Histogram Projection". In: *International Conference on Document Analysis and Recognition* (2009).

[SB88] G. Salton and C. Buckley. "Term-Weighting Approaches in Automatic Text Retrieval". In: *Information Processing & Management* (1988).

[SC78] H. Sakoe and S. Chiba. "Dynamic Programming Algorithm Optimization for Spoken Word Recognition". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* (1978).

[Sch00] B. Schölkopf. "The Kernel Trick for Distances". In: *Advances in Neural Information Processing Systems* (2000).

[Sch+00] B. Schölkopf, A. Smola, R. Williamson, and P. L. Bartlett. "New Support Vector Algorithms". In: *Neural Computation* (2000).

[SF83] A. Sanfeliu and K.-S. Fu. "A Distance Measure Between Attributed Relational Graphs for Pattern Recognition". In: *IEEE Transactions on Systems, Man and Cybernetics* (1983).

[She+11] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. "Weisfeiler-Lehman Graph Kernels". In: *Journal of Machine Learning Research* (2011).

[Sid+14] G. Sidorov, A. Gelbukh, H. Gomez-Adorno, and D. Pinto. "Soft Similarity and Soft Cosine Measure: Similarity of Features in Vector Space Model". In: *Computatción y Sistemas* (2014).

[SM92] R. Sablatnig and C. Menard. "Stereo and Structured Light as Acquisition Methods in the Field of Archaeology". In: *Deutsche Arbeitsgemeinschaft für Mustererkennung* (1992).

[Sod94] W. von Soden. *The Ancient Orient: An Introduction to the Study of the Ancient Near East*. Wm. B. Eerdmans Publishing Co., 1994.

[Spe81] Gerhard Sperl. "Erkennen von Keilschriftszeichen mit Hilfe elektronischer Rechenanlagen". PhD thesis. Innsbruck, 1981.

[TSK05] P. N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[Vis+10] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. "Graph Kernels". In: *Journal of Machine Learning Research* (2010).

[Vit67] A. J. Viterbi. "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm". In: *Transactions on Information Theory* (1967).

[WL68] B. Weisfeiler and A. A. Lehman. "A Reduction of a Graph to a Canonical Form and an Algebra Arising During This Reduction". In: *Nauchno-Technicheskaya Informatsia* (1968).