Ein unscharfer Suchalgorithmus für Transkriptionen von arabischen Ortsnamen

Scherl, Magdalena
magdalenascherl@gmail.com
Hochschule Mainz, Deutschland

Unold, Martin
martin.unold@gmail.com
Hochschule Mainz, Deutschland

Homburg, Timo
timo.homburg@hs-mainz.de
Hochschule Mainz, Deutschland

Einleitung

Motivation

Digitale Ortsverzeichnisse (Gazetteers) beinhalten Informationen über Orte sowie deren geographische Lage. Eine der grundlegendsten Aufgaben im Umgang mit solchen Ortsverzeichnissen ist die Suche nach Ortsnamen. Diese Suche kann sehr schwierig sein für Ortsnamen, die in verschiedenen Transliterations- oder Transkriptionsvarianten vorliegen, wie es oft bei arabischen Ortsnamen der Fall ist. In diesen Fällen reicht eine reine Volltextsuche nicht aus. Hier können unscharfe String-Matching-Algorithmen eine bessere Trefferquote für Suchen erreichen.

Zielsetzung

Unser Ziel war es, einen Suchalgorithmus zu entwickeln, der in der Lage ist, arabische Ortsnamen in verschiedenen Transliterationen und Transkriptionen zu identifizieren. Einerseits sollte der Algorithmus fehlertolerant sein, sodass er einen Suchbegriff findet, selbst wenn er etwas anders geschrieben wurde als im Ortsverzeichnis hinterlegt. Andererseits sollte er genau genug sein, um nur tatsächliche Transliterations- und Transkriptionsvarianten einzuschließen. Zum Beispiel sollte die Suche nach "Agaga" den Ort "Ajaja" finden, da es sich um verschiedene Transliterationen des selben arabischen Wortes handelt, aber nicht "Dagaga", da dies ein ganz anderer Ort ist. Um diese beiden Ziele zu erreichen, haben wir einen Algorithmus mit einer modifizierten gewichteten Levenshtein-Distanz (Levenshtein 1965) entwickelt. Eine weitere Eigenschaft unseres Suchalgorithmus ist, dass er für andere Anwendungsfälle als arabische Schrift leicht angepasst werden kann. Wir haben daher auch eine Version für Keilschriftsprachen implementiert und auf einem sumerischen Wörterbuch getestet.

Forschungsstand

Die gewichtete Levenshtein Distanz wurde bereits für Autokorrektur (Kukich 1992), für die Korrektur von Fehlern bei der Optical Character Recognition (OCR) (Lasko 2001, Mihov 2002) und für die automatische Spracherkennung (Ziolko 2010, Zgank 2012) genutzt. Um die Kosten für die Editieroperationen zu bestimmen, schlägt Weigel (1995) einen iterativen überwachten Lernalgorithmus vor. Lasko (2001) beschreibt die Verwendung einer probabilistischen Substitutionsmatrix und Schulz / Mihov (2002) schlagen die Implementierung eines endlichen Zustandsautomaten vor, um die Performanz des Levenshtein-Algorithmus zu verbessern.

Arabische Schrift

Variationen in der Schreibweise von arabischen Toponymen sind sehr häufig, da es mehrere Transliterationsstandards und verschiedene gebräuchliche Transkriptionsschemata gibt (Brockelmann 1953, Schlott-Kotschote 2004, UNGEGN 2016, Pedersen 2008). Insbesondere die Darstellung jener arabischen Buchstaben, die im lateinischen Alphabet keine direkte Entsprechung haben, variiert hier teilweise beträchtlich. Während einige Standards hierfür diakritische Zeichen verwenden, setzen andere Standards auf die Verwendung von Kombinationen aus zwei Buchstaben. Eine andere Quelle der Variation ist die fehlende Vokalisierung in der arabischen Schrift. Besonders regionale Variationen der Aussprache und Dialektdiversität führen dazu, dass arabische Vokale in der lateinischen Schreibweise unterschiedlich wiedergegeben werden. Zu Abweichungen führen auch unterschiedliche Traditionen der Transkription, die sich entweder eher an der englischen oder an der französischen Aussprache orientieren. Ein weiteres Problem, das zu Variationen führen kann, sind Wortgrenzen und divergierende Ansätze in der Zusammen- und Getrenntschreibung, insbesondere bei der Verwendung des Artikels "al".

Keilschriftsprachen

Die Entwicklung von Software für die Verbesserung der Bearbeitbarkeit von Keilschriftsprachen traf in der Vergangenheit auf ein reges Interesse in der Digital Humanities Community.

Homburg (2016, 2017, 2018) zeigten, dass Fortschritte in der Erstellung einer Natural Language Processing Pipeline und in der Erstellung von State-Of-The-Art semantischen

Wörterbüchern für verschiedene Keilschriftsprachen in Entwicklung sind. Homburg (2015) entwickelte eine auf einem Präfixbaum De La Briandais (1959) basierende Eingabemethode für Keilschriftsprachen, die auf der DHd 2015 präsentiert wurde. State Of The Art Eingabemethoden wie Sogou Pinyin[1] für Chinesisch oder Google Japanese Input[2] (Krueger 2000) für Japanisch beinhalten jedoch prädiktive Algorithmen, welche es erlauben die Korrektheit von Texteingaben in ihrem jeweiligen Kontext einzubeziehen und mit Fuzzy Search Algorithmen ebenfalls eine Korrektur von Tippfehlern vorzunehmen. Für die Eingabe von Keilschrift wurden solche Algorithmen bisher noch nicht erprobt, obwohl diese die Eingabe auch durch Einblendung von Zusatzinformationen ernorm vereinfachen kann und mehr relevante Suchergebnisse angezeigt werden können. Für Keilschriftsprachen im Speziellen ist eine Fuzzy Search für die Unterscheidung gerade auch der verschiedenen Dialekte und Transliterationen der Keilschriftarten interessant, da in diesen unter anderem Vokalverschiebungen und Variationen durch verschiedene Transliterationskonventionen auftreten können. Beispiele hierfür sind die Unterscheidungen von diakritischen Zeichen vs. einer numerischen Annotation (ù vs. u2), Transliterationsunterschiede wie die Verwendung von sh vs. sz und sprachliche Entwicklungen über die Zeit hinweg, in denen z.B. endende Konsonanten weggefallen sind (sogenannte Mimation).

Ansatz

Wir verwendeten ein modifiziertes Levenshtein Distanz Maß, welches speziell für die arabische Schrift angepasst wurde. Der Quellcode des Projektes ist unter der GPLv2 Lizenz in unserem Gitlab freigegeben worden.[3] Die Kosten für die Editieroperationen wurden hierbei durch ein überwachtes Lernverfahren ermittelt. Wir verwendeten eine Substitutionsmatrix sowie eine Matrix für Lösch- sowie Einfügeoperationen, um die jeweiligen Kosten der Überführung von einer Transliteration in die nächste zu bestimmen.

	b	d	e	i	r	u	ī	ay
b	0	1	0,97	0,91	1	0,86	0,86	2
d	1	0	0,91	0,98	1	1	0,81	2
e	0,97	0,91	0	0,48	0,89	0,63	0,28	0,19
i	0,91	0,98	0,48	0	1	0,92	0,09	0,56
r	1	1	0,89	1	0	0,69	0,75	2
u	0,86	1	0,63	0,92	0,69	0	1	0,49
ī	0,86	0,81	0,28	0,09	0,75	1	0	0,45
ay	2	2	0,19	0,56	2	0,49	0,45	0

Abbildung 1: Beispielwerte aus der Substitutionsmatrix.

Für die Matrix für Löschungen und Einfügungen haben wir zwei unterschiedlichen Ansätze verfolgt: Im ersten Ansatz (Levenshtein1) wurden die Lösch- sowie Einfügekosten für jeden Buchstaben ohne Betrachtung des Buchstabenkontexts ermittelt. Im zweiten Ansatz (Levenshtein2) wurden die Lösch- und Einfügekosten in Abhängigkeit des voranstehenden Buchstabens ermittelt.

	b	d	e	i	r	u	ī	ay
b	1	1	0,79	0,98	0,37	1	1	1
d	0,83	1	0,93	0,85	0,7	0,81	1	1
e	0,62	0,55	1	1	0,64	0,53	1	1
i	0,74	0,36	1	0,64	0,56	1	1	1
r	0,91	1	0,74	0,63	1	0,65	1	1
u	0,9	1	1	1	0,6	1	1	1
ī	0,3	0,19	1	1	0,31	1	1	1
ay	2	2	2	2	2	2	2	2

Abbildung 2: Beispielwerte aus der Matrix für Löschungen und Einfügungen (Levenshtein2). Reihen repräsentieren den zu löschenden bzw. den einzufügenden Buchstaben; Spalten repräsentieren den voranstehenden Buchstaben.

Desweiteren wurden spezielle Anpassungen für die arabische Schrift wie z.B. diakritische Zeichen (#), welche typisch für die gegebenen Transliterationen sind, in das erweiterte Alphabet aufgenommen. Außerdem waren Kombinationen aus zwei Buchstaben zu berücksichtigen, die ein arabisches Phonem repräsentieren (z.B. sh). Da die klassische Levenshtein Distanz nicht aus Buchstabenkombinationen errechnet werden kann, musste der Algorithmus auf diese angepasst werden. In einer vereinfachten Version (Levenshtein1Simple und Levenshtein2Simple) wurden die Buchstabenkombinationen im Vorhinein durch einen Index ersetzt, sodass eine klassische Berechnung über den originären Levenshtein Algorithmus erfolgen konnte. Dieser vereinfachte Ansatz wies eine deutlich höhere Performanz auf.

		b	d	e	r	i
	0	0,99	1,83	2,38	3,11	3,67
b	0,99	0	0,83	1,38	2,12	2,68
u	1,89	0,9	1	1,47	2,08	2,64
d	2,7	1,71	0,9	1,45	2,18	2,74
a	3,25	2,25	1,45	1,38	2,11	2,67
y	4,25	3,25	2,45	1,09	1,82	2,38
r	5,25	4,25	3,45	2,09	1,09	1,65
ī	5,56	4,57	3,76	2,4	1,4	1,18

Abbildung 3: Berechnung der Levenshtein Distanz für ein Beispiel Wortpaar mit Levenshtein2. Buchstabenkombinationen werden durch einen modifizierten Algorithmus berücksichtigt.

		b	d	e	r	i
	0	0,99	1,83	2,38	3,11	3,67
b	0,99	0	0,83	1,38	2,12	2,68
u	1,89	0,9	1	1,47	2,08	2,64
d	2,7	1,71	0,9	1,45	2,18	2,74
ay	4,7	3,71	2,9	1,09	1,82	2,38
r	5,7	4,71	3,9	2,09	1,09	1,65
ī	6,01	5,02	4,21	2,4	1,4	1,18

Abbildung 4: Berechnung der Levenshtein Distanz für ein Beispiel Wortpaar mit Levenshtein2Simple. Buchstabenkombinationen werden vorab auf einen Index gematcht.

Experimente und Ergebnisse

Die arabische Version des Suchalgorithmus wurde auf zwei Wörterbüchern getestet. Das erste Wörterbuch beinhaltete Toponyme von archäologischen Fundorten in Syrien, im Irak und in der Türkei, welche aus dem TEXTELSEM Repositorium des i3Mainz stammten (tts_arch)[4]. Das zweite Wörterbuch beinhaltete syrische Toponyme aus GeoNames (geo_SY)[5]. Zusätzlich wurde die Übertragbarkeit des Suchalgorithmus auf andere Sprachen auf einem sumerischen Wörterbuch getestet, das aus dem "Semantic Dictionary for Ancient Languages" extrahiert wurde (sum)[6]. Alle Wörterbücher wurden in ein Trainings- sowie ein Testkorpus aufgeteilt. Gemessen wurde die Mean Average Precision (MAP) bei einer Rückgabe der Ergebnisse in Form eines Rankings. Da die durchgeführten Tests so konzipiert waren, dass jeweils nur ein Ergebnis als zutreffend gewertet wurde, genügte für jedes Suchwort die Berechnung eines Präzisionswertes, der anschließend über alle Testsuchwörter gemittelt wurde. Die Ergebnisse unserer Tests sind in Tabelle 1 festgehalten. Sie zeigen, dass unser Algorithmus in der Lage war, Toponyme mit einer Präzision zwischen 90% und 95% abhängig vom Wörterbuch zu finden. Verglichen mit einem ungewichteten Levenshtein Distanzmaß kann unser Ansatz somit eine Verbesserung der Präzision zwischen 9 Prozentpunkten auf dem sumerischen Wörterbuch und

27 Prozentpunkten auf dem TEXTELSEM Wörterbuch erreichen.

Datenset	MAP bei Volltextsuche	MAP bei ungewichteter Levenshtein Distanz	MAP bei eigenem Algorithmus (beste Version)	Algorithmus
tts_arch	0.24	0.63	0.90	Levenshtein2Simple
geoSY_xs	0.01	0.81	0.95	Levenshtein1
Sum	0.01	0.83	0.92	Levenshtein2Simple

Tabelle 1: Testergebnisse. Die Tests zeigen, dass die Levenshtein2Simple Version des Algorithmus im allgemeinen Fall eine bessere Präzision sowie die beste Performanz aufweisen konnte.

Zusammenfassung

Unsere Version der gewichteten Levenshtein Distanz erwies sich als ein vielversprechender Ansatz für die Verbesserung von Suchergebnissen in digitalen Gazetteeren. Zusätzlich konnten wir durch die Anwendung des Algorithmus auf das sumerische Keilschriftwörterbuch die Übertragbarkeit des Algorithmus auf andere Sprachen demonstrieren. Obwohl die vorgeschlagene Adaption des Levenshtein Algorithmus für sumerische Keilschrift erfolgreich war, könnten in anderen Fällen möglicherweise neue Probleme auftreten. Da der Algorithmus bisher nur Kombinationen aus zwei Buchstaben berücksichtigt, würde er nicht für Transliterationen funktionieren, die auch Kombinationen aus mehr als zwei Buchstaben enthalten, beispielsweise für die Transliteration des kyrillischen Alphabets, die Kombination wie ``shtsh" für den kyrillischen Buchstaben # enthält. Für Fälle wie diesen müsste der Ansatz weiterentwickelt werden. Darüber hinaus wäre zu überlegen, inwieweit die Performanz des Algorithmus weiter verbessert werden könnte. Durch die Verwendung eines Burkhard-Keller-Baumes konnte die Performanz immerhin so weit gesteigert werden, dass die Suchzeit auf einem Testkorpus mit über 35.000 Einträgen auf unter eine halbe Sekunde im Durchschnitt reduziert wurde. Für die Verwendung mit größeren Wörterbüchern könnte jedoch eine weitere Verbesserung der Performanz wünschenswert sein. Als Möglichkeit hierfür wäre etwa die Verwendung eines Levenshtein-Automaten nach Schulz / Mihov (2002) zu prüfen, der als besonders effiziente Umsetzung des Levenshtein-Algorithmus gilt.

Fußnoten

1. https://pinyin.sogou.com/
2. https://www.google.co.jp/ime/
3. https://gitlab.rlp.net/mscherl/FuzzySearch
4. http://www.higeomes.org/
5. http://www.geonames.org/
6. https://situx.github.io/SemanticDictionary/

Bibliographie

Brockelmann, C. / Fischer, A. / Heffening, W. / Taeschner, F. (1935): *Die Transliteration der arabischen Schrift in ihrer Anwendung auf die Hauptliteratursprachen der islamischen Welt. Denkschrift dem 19. Internationalen Orientalistenkongreß in Rom.*

De La Briandais, R. (1959): *File searching using variable length keys*. In: Papers presented at the the March 3-5, 1959, western joint computer conference. pp. 295–298. ACM.

Homburg, T. (2017): *Postagging and semantic dictionary creation for hittite cuneiform*. In: DH2017 .

Homburg, T. (2018): *Semantische Extraktion auf antiken Schriften am Beispiel von Keilschriftsprachen mithilfe semantischer Wörterbücher*. In: Dhd2018 .

Homburg, T., Chiarcos, C. (2016): *Word segmentation for akkadian cuneiform*. In: LREC 2016 .

Homburg, T., Chiarcos, C., Richter, T., Wicke, D. (2015): *Learning cuneiform the modern way*, http://gams.uni-graz.at/o:dhd2015.p.55 .

Krueger, M.H., Neeson, K.D. (2000): *Japanese text input method using a limited roman character set*, uS Patent 6,098,086

Kukich, K. (1992): *Techniques for automatically correcting words in text*. ACM Computing Surveys 24,4 .

Lasko, T.A., Hauser, S.E. (2001): *Approximate string matching algorithms for limited-vocabulary ocr output correction*, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.80.1064&rep=rep1&type=pdf .

Levenshtein, V.I. (1966): *Binary codes capable of correcting deletions, insertions, and reversals*. Soviet Physics Doklady 10,8 .

Pedersen, T.T. (2008): *Transliteration of arabic*, http://transliteration.eki.ee/pdf/Arabic_2.2.pdf .

Schlott-Kotschote, A. (2004): *Transkription arabischer Schriften. Vorschläge für eine einheitliche Umschrift arabischer Bezeichnungen.*

Schulz, K.U., Mihov, S. (2002): *Fast string correction with levenshtein automata*. International Journal on Document Analysis and Recognition 5 .

UNGEGN Working Group, R.S. (2016): *Arabic. report on the current state of united nations romanization systems for geographical names*. version 4.0, http://www.eki.ee/wgrs/rom1_ar.pdf .

Weigel, A., Baumann, S., Rohrschneider, J. (1995): *Lexical postprocessing by heuristic search and automatic determination of the edit costs*. In: Proceedings of the Third International Conference on Document Analysis and Recognition .

Zgank, Kacic (2012): *Predicting the acoustic confusability between words for a speech recognition system using levenshtein distance*, http://eejournal.ktu.lt/index.php/elt/article/download/2628/1917 .

Zió#ko, B., Ga#ka, J., Skurzok, D., Jadczyk, T. (2010): *Modified weighted levenshtein distance in automatic speech recognition*, http://www.dsp.agh.edu.pl/_media/pl:bziolko_kkzmbm2010final.pdf .