Universität zu Köln

Philosophische Fakultät

Institut für Linguistik

Abteilung Phonetik

Magisterarbeit

Aspekte der Kodierung phonetischer Ähnlichkeiten in deutschen Eigennamen

Prüferin: Prof. Dr. Martine Grice

vorgelegt von

Martin Wilz

Berliner Ring 59-61
50321 Brühl

Vorwort

Ich möchte den folgenden Personen danken, die mit Ihrer Unterstützung dazu beigetragen haben, dass diese Arbeit in dieser Form erscheinen konnte.

Insbesondere möchte ich Yvonne Bastian, Martine Grice, Doris Mücke, Christoph Reuter und Christian Schneider für ihre vielzähligen Hinweise und Korrekturvorschläge danken.

Darüber hinaus haben folgende Personen wertvolle Anregungen gegeben sowie bei der Organisation von schwer erhältlicher Literatur geholfen: Damon Allen Davidson, Gustav Gvella, Justyna Hadyniak und Carsten Jacobi.

Inhaltsverzeichnis

1 Einleitung

In dieser Arbeit soll versucht werden, einen Überblick über Möglichkeiten der Kodierung von Ähnlichkeiten in deutschen Familiennamen zu geben. Ausschlaggebend hierfür soll die Aussprache der Namen sein. So ist eine Kodierung von Ähnlichkeiten sinnvoll, wenn für einen Begriff oder Namen die Aussprache, nicht aber die genaue Schreibweise bekannt ist. Dies stellt bei Familiennamen ein besonderes Problem dar, da aufgrund der Entstehungsgeschichte für fast jeden Namen eine Vielzahl von Schreibvarianten mit ähnlicher Aussprache existieren. Andererseits gibt es für viele Schreibweisen die Möglichkeit von mehreren Aussprachevarianten. Die Aufgabe ist dementsprechend für eine Schreibweise eines Familiennamens alle anderen äquivalenten zu finden.

Ein Schwerpunkt dieser Arbeit wird bei den Verfahren für die Phonetische Suche liegen. Diese versuchen mit recht ähnlichen Strategien die genannten Ambiguitäten aufzulösen. Universellere Ansätze aus der klassischen Informatik, wie Edit-Distance- und N-Gram-basierte Verfahren werden ebenfalls vorgestellt und auf ihre Anwendbarkeit für die gegebene Problemstellung untersucht.

Zu diesem Zweck wird aus einer Telefonbuch-CD ein Korpus von ca. 1,2 Millionen Familiennamen extrahiert und für die häufigsten Namen eine Bewertung der Verfahren vorgenommen. Nach einer Auswertung werden ausgewählte Besonderheiten der deutschen Orthographie und deren Aussprachevarianten vorgestellt. Da viele der beschriebenen Verfahren für die Phonetische Suche keine optimalen Suchergebnisse liefern, werden abschliessend Vorschläge gemacht, wie mit wenig Aufwand fehlerfreiere Suchergebnisse zu erreichen sind.

2 Allgemeines über Namen

Als Erstes soll eine kleine Einführung in die verschiedenen Namenstypen gegeben werden. Da sich diese Arbeit im Folgenden auf die Behandlung von Familiennamen[1] beschränkt, wird anschliessend deren Entstehungshistorie skizziert. Diese erklärt die Ursache für die große Varianz an Schreibweisen.

2.1 Namenstypen

Verschiedene Typen von Namen sind bekannt. So werden im Alltag z. B. Vornamen, Familiennamen, Eigennamen, Firmennamen, Ortsnamen und Tiernamen verwendet. Viele dieser Typen sind voneinander abgeleitet oder haben eine ähnliche Struktur. So ist es eine

[1] Die verwendete Datenquelle für den Korpus hätte auch Material für die Untersuchung von weiteren Namenstypen, wie Vor- und Ortsnamen liefern können. Da die Auswahl an Familiennamen ein sehr vielfältiges Spektrum bietet, wurden weitere Namenstypen nicht näher untersucht.

weit verbreitete Vorgehensweise, Straßen und Plätze nach Personen oder Orten zu benennen. Z. B. wurden in Köln ganze Siedlungen mit Straßennamen, die von Vogelnamen abgeleitet wurden, versehen.

Firmennamen hingegen sind häufig Kunstworte und werden mittels Komposition aus Sprachelementen anderer Sprachen[2] oder aus Silben von Eigennamen, z. B. <Haribo>[3] gebildet. Aber auch Familiennamen mit Zusätzen, die die Art des Gewerbes beschreiben, z. B. <Ernst Metallbau GmbH>, sind nicht unüblich.

Vornamen unterliegen vielen Moden und werden gerne aus anderen Kulturen importiert. So ist oft einziges Kriterium für eine behördliche Akzeptanz in Deutschland, dass möglichst das Geschlecht erkennbar ist. Dies führt gerade bei Vornamen zu einer Vielzahl von Schreibweisen, bei der die Aussprache von vielen Sprachen beeinflusst wird. Es dürfte sehr schwer sein, die Aussprache eines Vornamens vorherzusagen, wenn der Sprachursprung nicht bekannt ist. So kann z. B. <Michael> je nach Sprachkontext [mɪçaəl], [mɪkaəl] oder [mʊ̯ɪkəl] ausgesprochen werden.

Familiennamen unterliegen in der jetzigen Zeit keiner wesentlichen Fluktuation. Die Varianz wird derzeit nur durch Einwanderung und Künstlernamen erhöht. Aber auch hier existieren die schon bei Vornamen beschriebenen Probleme. In Vitale (1991, S. 271) werden solche sprachabhängigen Aussprachevarianten als homographische Variationen bezeichnet. Als Beispiel wird der Name <Pace> gegeben, welcher entweder englisch [pɛ̯ɪs] oder italienisch [patçə] ausgesprochen werden kann. Neuere Untersuchungen wie Breuer u. Abresch (2003) schlagen vor, hier eine Aussprache an dem Sprachstand eines durchschnittlichen Sprechers festzumachen.

Auch semantisch betrachtet, bereiten Eigennamen einige Probleme. So sind Sie, wenn Sie dem Rezipienten unbekannt sind, aufgrund ihrer variablen Form höchstens anhand des lexikalischen Kontextes zu erkennen. Dies bereitet für eine maschinelle Verarbeitung große Probleme. Ein Namenslexikon könnte bei einer Vielzahl von Fällen bei der Erkennung helfen. Es müsste aber aufgrund der oben erwähnten generativen Namenstypen immer auf einen aktuellen Stand gebracht werden. Für eine automatische Erkennung durch Maschinen gibt es Ansätze speziell für Namen (vergleiche auch Thielen (1995)). Dieser Bereich wird voraussichtlich für einen längere Zeitraum noch Forschungsgebiet bleiben.

[2]Beliebt sind hier vor allem europäische Sprachen wie Englisch, Spanisch, Griechisch und Latein. Aber auch Kunstsprachen wie Esperanto.

[3]entstanden aus den jeweils ersten Buchstaben von <Hans Riegler Bonn>

2.2 Ursprung von Familiennamen

Familiennamen entstanden in Europa vorwiegend seit dem Hochmittelalter, vor allem vorangetrieben von der Bevölkerungszunahme in den Städten. Dort wurde es notwendig für Personen mit gleichen Rufnamen ein weiteres Unterscheidungsmerkmal einzuführen, um diese Personen eindeutig benennen zu können. Dies geschah zuerst durch Namenszusätze wie ,,der Starke". Da diese aber keine Kontinuität über die Generationen hinweg boten, wurden Familiennamen eingeführt. In kleinen Dörfern gab es bis ins 19. Jahrhundert hinein aufgrund der geringen Bevölkerungsdichte keinen Anlass dafür.

Die Vergabe von Namen geschah häufig auf der Basis von mit der Person verbundenen Begriffen. So ist laut Kunze (2003) der Ursprung von Familiennamen in Rufnamen, Ortsnamen, Hausnamen, Berufsbezeichnungen, körperlichen oder charakterlichen Eigenschaften der Person oder auch in geographischen Gegebenheiten des Wohnraumes zu finden. Bei vielen Namen ist auf den ersten Blick nicht mehr ersichtlich, welche Bedeutung sie einmal getragen haben, da sich die Sprache im Laufe der Zeit in einigen Aspekten stark gewandelt hat. Beispielsweise ist der Name <Brinkmann> aus dem niederdeutschen <Brink> abgeleitet worden und bedeutet laut Kunze (2003, S. 97) soviel wie Grashügel, Anhöhe oder unbebautes Land. Sehr häufig und auch in anderen Kulturen weit verbreitet sind Namensformen, die aus dem Rufnamen des Vaters abgeleitet werden.

Der auf die verschiedenen Arten abgeleitete Wortstamm von Namen wird dabei nicht selten mit bedeutungstragenden Suffixen ergänzt. Dies können Verniedlichungen wie z. B. das rheinländische Suffix <-chen> sein. Nicht nur im Deutschen sind Endsilben wie das norddeutsche/skandinavische <-sen> häufig. Dieses bedeutet soviel wie ,,Sohn des". Namen wie <Hansen>, <Jansen> oder <Petersen> sind mittlerweile in ganz Deutschland verbreitet.

Die nachwirkenden Einflüsse der Lautverschiebungen[4] und Schreibmoden im Laufe der Jahrhunderte erzeugten viele unterschiedliche Schreibweisen. Erst das Bürgerliche Gesetzbuch vom 1.1.1900 enthielt Bestimmungen zur Festlegung auf Schreibweisen von Familiennamen, nachdem die Rechtschreibung der Sprache 1870 mit dem Duden festgelegt wurde.

Eine weitere Quelle für Familiennamen stellt die Einwanderung aus anderen Sprachräumen dar. Die Namen entstanden dort meist nach ähnlichen Prinzipien, wie den oben beschriebenen, sind jedoch an die Quellsprache gebunden. Falls ein vollkommen andersartiges Schriftsystem in der Quellsprache verwendet wurde, wird der Name transliteriert[5].

[4]Es sollte in diesem Zusammenhang erwähnt werden, dass Lautverschiebungen regional und in Dialekten nicht oder nur schleichend übernommen wurden. So ist die Namensgebung zusätzlich stark von regionalen Einflüssen geprägt.

[5]Unter einer Transliteration versteht man die Umsetzung eines Sprachelementes in ein anderes Schriftsystem. Es wird zu diesem Zweck eine meist an phonetisch/phonologischen Prinzipien angelehnte Tran-

Vor der bereits erwähnten Festlegung der Schreibweise von Familiennamen konnte es bei zugewanderten Familien über die Jahre zu einem Assimilationsprozess in der Schreibweise kommen. Heutzutage ist es bei der Einwanderung aus Gebieten mit ähnlichem Schriftsystem nicht unüblich, kleinere Änderungen an der Schreibweise oder Aussprache vorzunehmen. Dies geschieht vor allem, um Alltagsprobleme mit für Deutsche ungewohnter Schreibweise zu vermeiden. Eine Studie, die ähnliche Vorgänge untersucht, findet sich in Scholz (2004)[6].

Ein weiteres auch in anderen Ländern wie England oder Polen bekanntes, aber in Deutschland sehr weit verbreitetes Phänomen, stellen Doppelnamen dar. Sie gibt es in der BRD seit 1953. In der Schweiz sind sie schon seit ca. 1830 möglich. Ab 1994 dürfte die Anzahl von Doppelnamen wieder gesunken sein, da Ehepartner seit dem nicht mehr verpflichtet sind, den gleichen Namen zu führen, bzw. sie können ihren Geburtsnamen behalten.

3 Die Phonetische Suche

Im Bereich der Informatikliteratur und in der Umgangssprache wird mit der ,,Phonetischen Suche" ein Algorithmus benannt, der das Auffinden von gleich und ähnlich ausgesprochenen Zeichenketten[7] erlaubt. Die Bezeichnung ,,Phonetisch" ist darauf zurückzuführen, dass implizit phonetisch/phonologisches Wissen über die Aussprache von Zeichenketten verwendet wird.

Für den Fall, dass nach einem Namen gesucht wird, dessen Aussprache bekannt ist, nicht aber seine Schreibweise, stellt die Phonetische Suche ein Hilfsmittel dar. Dieses Problem tritt aus den im vorangegangen Kapiteln erläuterten Gründen besonders bei Eigennamen häufig auf. So gilt es für den Algorithmus die Suchanfrage so zu formulieren, dass mögliche Ambiguitäten bei der Aussprache berücksichtigt werden. Dies geschieht bei der Phonetischen Suche unter Ausnutzung von Heuristiken über die Beziehungen von Graphemen und Phonemen. Dazu wird meist mit Hilfe von Regeln eine Eingabezeichenkette derart umkodiert, dass möglichst viele Ambiguitäten eliminiert werden. Die Phonetische Suche gleicht somit einer Transliteration in ein ungenaues, phonologisch inspiriertes Alphabet. Die Kodierung von Ähnlichkeiten wird bei der Phonetischen Suche üblicherweise dadurch vorgenommen, dass Information, welche Namensvarianten unterscheidet, ausgelassen wird.

skription in die Orthographie der Zielsprache vorgenommen.

[6]Hier wird die phonetische und phonologische Angleichung von Lehnwörtern aus dem Englischen an die deutsche Sprache beschrieben. Für die dort untersuchte Sprachkombination wird die Komplexität dieser Angleichungen ersichtlich.

[7]Mit Zeichenkette wird im Folgenden die von digitalen Systemen verwendbare Abfolge von Zeichen bezeichnet. In dieser Zeichenkette können Buchstabenfolgen der jeweiligen Orthographie kodiert werden.

Im Folgenden soll noch einmal genauer betrachtet werden, für welche Bereiche eine Phonetische Suche benötigt wird und in welcher Form sie den Benutzern dargeboten wird. Danach wird die Funktionsweise erläutert und diskutiert welche Eigenschaften sowohl im Allgemeinen als auch in der deutschen Sprache besondere Aufmerksamkeit verdienen. Anschliessend werden ausgewählte Ansätze für die Phonetische Suche vorgestellt.

3.1 Anwendungsgebiete für eine Phonetische Suche

Die vorgestellten Verfahren werden immer dann für die Suche nach Eigennamen eingesetzt, wenn die korrekte Schreibweise nicht bekannt ist. Dies ist vor allem dann der Fall, wenn der Name nur akustisch weitergegeben wurde. Dabei wird dem Benutzer für die Suchzeichenkette meist ein Eingabefeld innerhalb einer Suchmaske präsentiert. Die Eingabe wird mit dem vorhandenen Datenbestand verglichen. Anschliessend wird dem Benutzer eine Liste von potentiell gleich geschriebenen Datensätzen präsentiert. Unterstützen könnte dabei die Anordnung der Suchergebnisse nach Ähnlichkeit [8].

Folgende Anwendungsgebiete sind bekannt:

- Das schnelle Auffinden von Kundendaten für Telefon-Hotlines.
- Die genealogische Suche nach Namen. Hier kann es jedoch auch sinnvoll sein, nicht nur ähnlich lautende Namen zu berücksichtigen, sondern auch evt. enthaltene Wortstämme.
- Das Identifizieren von Personen in polizeilichen Ermittlungen, z. B. bei abgehörten Gesprächen.
- Die Recherche von Namen, die nur mündlich/fernmündlich weitergegeben wurden z. B. bei Telefonaten, bei Gesprächen oder im Rundfunk. Laut Kukich (1992) wird bei unbekannten Namen versucht, eine Zeichenkette einzugeben, deren Aussprache phonetisch äquivalent ist.

3.2 Funktionsweise der Phonetischen Suche

Die Phonetische Suche kann der Klasse der Hashing-Algorithmen[9] zugeordnet werden. Der übliche Ansatz für eine Phonetische Suche ist die Erzeugung eines Schlüssels für jede orthographische Repräsentation, die in eine Suche einbezogen werden soll. Üblicherweise, vor allem wenn Datenbanken[10] für die Speicherung von Eigennamen verwendet werden,

[8] In der englischsprachigen Literatur wird hier von ,,Ranking" gesprochen.

[9] Hashing bezeichnet eine Funktion, die eine große Menge von Daten auf eine kleinere Menge abbildet. Meist ergibt sich durch die Art der Funktion eine implizite Gruppierung der Daten. Hashing-Algorithmen werden in vielen anderen Bereichen verwendet z. B. für digitale Signaturen, Prüfsummenberechnungen oder Beschleunigung von Speicherzugriffen.

[10] Es wird im Folgenden davon ausgegangen, dass in der Praxis Datenbanken verwendet werden. Andere Ansätze wie speziell für gewisse Einsatzzwecke programmierte Applikationen sind natürlich auch möglich,

wird dieser Schlüssel schon beim Einfügen des Datensatzes in die Datenbank berechnet. Mit einem Index auf der Datenbankzeile, in der der Schlüssel gespeichert wird, ist ein sehr schnelles Auffinden von Zeichenketten mit gleichem Schlüssel möglich.

3.3 Typische Eigenschaften

Bei der Phonetischen Suche handelt es sich typischerweise um einen kurzen, einfachen Algorithmus, der mit einer möglichst kleinen Anzahl von Regeln auskommt. Wenn überhaupt benötigt, wird ein kleines Morphemlexikon verwendet. Dies erlaubt den Einsatz in einer Vielzahl von Umgebungen, wie z. B. Datenbanksystemen und Programmiersprachen ohne große Bemühungen auf Seiten des Softwareentwicklers.

Die Eingabe an den Algorithmus ist eine Zeichenkette mit einer orthographischen Repräsentation des Eigennamens. Ausgegeben wird üblicherweise ein Schlüssel, der aus Ziffern und/oder Buchstaben besteht. Diese Ziffern entsprechen meist phonetisch/phonologischen Gruppierungen, die sich an den Aussprachemöglichkeiten orientieren. Einige der Algorithmen versuchen Ambiguitäten in der Aussprache dadurch zu modellieren, dass unter Umständen für eine Eingabezeichenkette mehrere Schlüssel erzeugt werden. Ein Ansatz für die Phonetische Suche, der verschiedenen Ausspracheregeln Wahrscheinlichkeiten zuordnet, existiert nicht. Im verwandten Bereich der Graphem-nach-Phonem-Konvertierung und Silbifizierung wurde ein solches Vorgehen von Müller (2000a) beschrieben.

Da ähnliche Orthographien auf gleiche Schlüssel abgebildet werden, ist es alleine mit einer solchen Phonetischen Suche nicht möglich, eine Reihenfolge der Suchergebnisse zu bestimmen. Einige der vorgestellten Verfahren versuchen Ambiguitäten in der Aussprache durch Erzeugung von mehreren Schlüsseln zu modellieren. Dazu wird nacheinander nach jedem erzeugten Schlüssel gesucht. Mit dieser Vorgehensweise wäre ein Ranking möglich. Es liegen keine Untersuchungen darüber vor, ob dadurch eine sinnvolle Reihenfolge entstehen könnte. Hierzu wären entweder Statistiken über die Häufigkeit von Aussprachevarianten oder Bewertungen der generierten Reihenfolge durch Probanden notwendig. Natürlich kann bei jedem der vorgestellten Verfahren die Menge der gefundenen Datensätze für eine weitere Bewertung der Reihenfolge benutzt werden. Dies ist z. B. mit Verfahren wie den später vorgestellten Edit-Distance-Algorithmen möglich.

3.4 Wünschenswerte Eigenschaften

Eine Phonetische Suche sollte im Idealfall für alle Zeichenketten, welche gleich ausgesprochen werden könnten, den gleichen Schlüssel erzeugen. Vieles spricht dafür, dass dieses Ziel nicht erreichbar ist. Jede Sprache hat eigene Ausspracheregeln, so dass eigentlich eine Erkennung der zu verwendenden Sprache von Nöten ist. Aber selbst, wenn man sich auf

jedoch verfügen auch diese meist über Programmmodule, die in ihrer Funktionen einfachen Datenbanken entsprechen.

das Deutsche beschränkt, gibt es viele Ambiguitäten in der Aussprache. So variiert die Aussprache regional stark, selbst, wenn das weite Feld der Dialekte vernachlässigt wird. Zusätzlich werden mit der fortschreitenden Vermischung der Kulturen viele Familiennamen aus ihrem Sprachkontext herausgenommen oder assimiliert. So bleibt vorerst in der Praxis nichts anderes übrig, als ein Verfahren zu entwickeln, welches in einem definiertem sprachlichem Kontext möglichst gut funktioniert. Ein Verfahren, welches für mehrere Sprachen Aussprachevarianten berücksichtigt, dürfte einen erheblichen Forschungsaufwand verlangen. Aber auch danach werden für den praktischen Einsatz relevante Grenzen wie Rechenleistung und Speicherbedarf nicht jeden beliebigen Anwendungszweck zulassen.

Eine weitere vorteilhafte Eigenschaft wäre, wenn der Algorithmus für eine Eingabezeichenkette möglichst wenig Schlüssel generiert oder die Anzahl der erzeugten Schlüssel begrenzt. Ansonsten könnten die Geschwindigkeitsvorteile des Hashingprinzips bei ungünstigen Eingabezeichenketten[11] durch kombinatorische Explosion[12] gefährdet werden.

Gegen eine zu kleine Anzahl von Regeln spricht, dass Sonderfälle der Orthographie nicht oder nicht ausreichend abgedeckt werden können. Es gilt einen Kompromiss zwischen der Anzahl der verwendeten Regeln und der resultierenden Trefferquote zu finden.

Algorithmen für die Phonetische Suche profitieren von einer Auswertung des Graphemkontextes. Dies gilt insbesondere für die deutsche Sprache. Ein bekanntes Beispiel dafür ist das Graphem <ch>, dessen Aussprachevarianten [ç] und [χ] vor allem durch den vorhergehenden Vokal bestimmt werden. Verschiedene Methoden der Berücksichtigung des Kontextes werden bei den einzelnen Verfahren detaillierter besprochen.

Eine kurze Schlüssellänge und/oder ein kleines Zielalphabet für den erzeugten Schlüssel erlauben die effiziente Speicherung in digitalen Speichersystemen. Wilde u. Meyer (1988) schlagen eine Begrenzung der Schlüssellänge auf fünf oder sechs Zeichen vor, um die Anzahl der korrekten Suchergebnisse zu maximieren. Dies wird damit begründet, dass die meisten Schreib- oder Hörfehler am Ende des Wortes auftreten. Laut Kukich (1992) ist dies jedoch nicht unbedingt der Fall.

Eine kurze Schlüssellänge erleichtert darüber hinaus eine weitere Verwendung in ei-

[11] In der informatischen Literatur wird hier mit Worst-Case-Abschätzungen gearbeitet, auf deren Verwendung im Folgenden verzichtet wird, da die vorgestellten Algorithmen nur mit sehr kurzen Eingabezeichenketten arbeiten und eine durch den Anwendungszweck bestimmte Obergrenze der Komplexität haben.

[12] Mit jeder Alternative für die Aussprache multipliziert sich die Anzahl der Schlüssel um die Anzahl der Aussprachevarianten. Bei der Berechnung der Schlüssel für den Korpus konnte beobachtet werden, das der Daitch-Mokotoff Algorithmus im Normalfall nur wenige Schlüssel generiert. So werden lediglich für 3,55 % der Einträge mehr als zwei Schlüssel kodiert. In 91,7 % dieser Fälle waren es vier Schlüssel. Das Maximum lag bei 32 Schlüsseln für einen Eintrag. Dieses Maximum wurde nur acht Mal für Familiennamen im Korpus erzeugt.

ner an eine bestimmte Plattform gebundene Implementierung bzw. lässt Optimierungen der Geschwindigkeit zu, wenn die maximale Schlüssellänge kleiner als die verwendete Wortlänge von Prozessorregistern ist. Dies ist z. B. bei Soundex und Extended Soundex der Fall, die beide auf 32-Bit-Prozessoren direkt in die Prozessorregister geladen werden können[13].

Eine Kombination aus zu kleiner Schlüssellänge und zu kleinem Schlüsselalphabet hat jedoch den Nachteil, dass der Namensraum auf zu wenig Schlüssel abgebildet werden muss. Dadurch werden eine Vielzahl der Suchergebnisse von der Aussprache stark abweichen. Statistiken zur Größe der Schlüsselräume der vorgestellten Algorithmen werden später in Kapitel 6.4 vorgestellt. Es wird sich zeigen, dass einige der Algorithmen die Datensätze des Korpus auf eine zu kleine Menge von Schlüsseln abbilden. Setzt man jedoch die vorgestellten Verfahren nur für eine Filterung der Daten ein, so kann dieser Punkt vernachlässigt werden.

Wünschenswert wäre, das das Alignment[14] von Graphemen und Phonemen berücksichtigt wird. Bei einfachen Algorithmen wie z. B. Soundex wird hier ein Verhältnis von 1:1 angenommen. Dies bringt bei den im Deutschen vorkommenden Graphemen <ph> und <ch> Probleme mit sich, da diese, der Aussprache entsprechend, meist als ein Laut kodiert werden sollten. Wie im Englischen gibt es Laute, bei denen das Verhältnis von Phonemen zu Graphemen 2:1 ist. Beispiel ist das <x> in <Xaver>, welches wie im Deutschen meist als [ks] realisiert wird.

Potentiell positive Auswirkungen hätte die Anordnung der Information innerhalb des Schlüssels. So könnten wesentliche Erkennungsmerkmale der kodierten Zeichenkette am Anfang des Schlüssels kodiert werden. Die unwesentlicheren Merkmale werden am Ende kodiert. Dies würde eine Suche erlauben, bei der der gewünschte Grad der Ähnlichkeit mit typischen Datenbankoperationen[15] möglich wäre. Keines der vorgestellten Verfahren hat eine entsprechende Vorgehensweise.

3.5 Spezielle Probleme bei der Phonetischen Suche mit deutschen Namen

Die von Rechnersystemen eingesetzte Zeichenkodierung stellt einen großen Stolperstein für die meisten Algorithmen dar, die nicht alle Zeichen verarbeiten können, die in deutschen Familiennamen vorkommen. Viele der vorgestellten Verfahren zeigen nicht unwesentliche

[13] 4 x 8-Bit-ASCII-Buchstaben = 32 Bit. Ebenfalls in einem 32 Bit-Wort kodierbar sind Extended Soundex und Daitch-Mokotoff, da es sich um rein numerische Codes handelt. Dabei ergeben 8 bzw. 6 x 4-Bit-Ziffern < 32 Bit.

[14] Als Alignment bezeichnet man in diesem Fall die Zuordnung von Graphemen zu den zu realisierenden Phonemen.

[15] Insbesondere von Interesse ist hier die SQL-Funktion „substring". Diese erlaubt die Extraktion von Teilzeichenketten aus einem Datenbankfeld. Je nach gewünschter Genauigkeit könnte die Länge dieser Teilzeichenkette von Schlüsseln für eine Reihenfolge der Suchergebnisse sorgen.

Probleme bei der Berücksichtigung von Sonderzeichen und Umlauten. Ein weiteres Problemfeld stellen im Deutschen die Doppelnamen dar. So ist nicht immer klar, ob ein Doppelname gemeint ist oder der Name einfach etwas länger ist.

3.5.1 Zeichenkodierung

Da Computer lediglich mit Zahlen umgehen können, bedarf es einer Kodierung der einzelnen Buchstaben. Sowohl in der informatischen Praxis, als auch in der Literatur herrschen schon seit Jahrzehnten englischsprachig geprägte Ansätze vor. Dadurch ergeben sich im praktischen Umgang immer wieder Anpassungsprobleme bei der Verarbeitung von Sprachen, deren Schriftsysteme Zeichen enthalten, die im englischsprachigen Raum nicht vorkommen. Besonders die älteren Ansätze sind vor allem ASCII-basiert[16], bzw. nie für einen Einsatz in nicht-englischen Umgebungen entwickelt worden.

Innerhalb des ASCII-Systems werden die Buchstaben für die Zahlen 0-127 festgelegt. Die insgesamt in einem Byte verwendbaren Werte 128-255 werden für Erweiterungen des Schriftsystems benutzt. Es bestanden in den 90er Jahren eine Vielzahl von diesen Erweiterungen. Dort wurden unter anderem Umlaute kodiert, aber auch Grafikzeichen, wie z. B. Linien und Ränder, die vor allem an das Betriebssystem der Rechner gebunden waren. Diese Systeme standen jedoch meist in Konkurrenz zueinander, da für die Umlaute andere Zahlen kodiert wurden. So müssen Zeichenketten meist erst in eine Kodierung überführt werden, die von der jeweiligen Implementierung des Algorithmus verstanden wird.

Das beschriebene Problem ist auch heute noch relevant. So konkurieren derzeit die verschiedenen Zeichenkodierungen vor allem auf nationaler Ebene. Es existieren mehrere Varianten von ISO 8859, die für die Kodierung von verschiedenen Schriftsystemen verwendet werden. Daneben gibt es mehrere Unicode-Varianten, die einen universelleren Anspruch haben. Sie versuchen sämtliche bekannten Zeichen aller Sprachen zu kodieren, müssen dadurch jedoch die sonst übliche Zuordnung von einem Zeichen zu einem Byte aufheben. Es ist zwar möglich einige der Algorithmen direkt in Unicode-Umgebungen umzusetzen, jedoch müssen andere[17] erst angepasst werden. Für die Kodierung von deutschen Texten wurde im Rahmen dieser Untersuchung das Format ISO 8859-1 gewählt, welches eine Repräsentation der meisten europäischen Schriftzeichen kennt.

Für das Deutsche sind vor allem die Umlaute <ä>, <ö>, <ü> und das <ß> wesentlich. Eine Unterstützung für weitere Zeichen, wie z. B. Vokale mit Akzenten, sind auch nicht uninteressant, da sie in etlichen benachbarten Ländern wie z. B. Frankreich, Belgien oder Dänemark verwendet werden und ein Bevölkerungsaustausch mit diesen Ländern statt-

[16] ASCII = American Standard Code for Information Interchange.

[17] Dies dürfte vor allem die Implementierung von Phonet betreffen, das eine eigene Funktionsbibliothek für die Anwendung der Regeln mitliefert.

findet. Die genannten Zeichen sind im ASCII-Standard nicht vertreten. So enthalten die englischsprachigen Algorithmen keinen Programmcode, der explizit mit diesen Graphemen umgehen kann. Das Problem wird dadurch gemildert, das in den meisten Fällen Zeichen, die den Algorithmen unbekannt sind, ignoriert werden. Im Falle der Umlaute <ä>, <ö>, und <ü> ist dies nicht unbedingt tragisch, da es sich um Vokalgrapheme handelt, die von den meisten englischsprachigen Algorithmen ohnehin entfernt werden. Lediglich am Anfang der Zeichenkette treten hier Probleme auf, wenn der erste Buchstabe Bestandteil des Schlüssels ist, wie z. B. bei Soundex und Metaphone.

Allerdings gibt es Komplikationen bei Namen, die ein <ß> enthalten. Dieses kann in dem beschriebenen Szenario nicht der Klasse der Laute zugeordnet werden, die als [s] realisiert werden, da dieses Zeichen vorher eliminiert wird. Anhand des in Kapitel 5 beschriebenen Korpus ließ sich die Größenordnung dieses Problems ermitteln: Ca. 1.56 % der Familiennamen enthalten mindestens ein <ß>.

Buchstabe	ASCII
ä	ae
ö	oe
ü	ue
ß	ss

Abbildung 1: Tabelle der im Deutschen wichtigen Zeichen, die von der ASCII-Kodierung nicht berücksichtigt werden. Die zweite Spalte gibt Ersetzungsvorschläge an.

Falls es notwendig werden sollte, einen Algorithmus zu verwenden, der lediglich mit ASCII-Kodierungen umgehen kann, können die in der Abbildung 1 gezeigten Ersetzungen[18] benutzt werden. Weiterhin sollten evt. vorhandene Diakritika, wie z. B. Akzente und Cedille entfernt werden. Diese Ersetzungen dürften wesentlich verbesserte Ergebnisse bei ASCII-basierten Implementierungen von Phonetischen Suchen auf deutschen Namen gewährleisten, wenn sie für alle generierten Schlüssel angewendet werden.

3.5.2 Doppelnamen

Namensänderungen verlangen in fast jedem Anwendungsgebiet eine erhöhte Aufmerksamkeit bei der Dateneingabe und der Aktualisierung der Datenbestände. Doppelnamen stellen eine Form der Namensänderung dar: Der Name wird durch Anhängen oder Voranstellen eines weiteren Namens verändert. Jedoch ergeben sich noch weitere Problemfelder durch Doppelnamen:

[18] Ein ähnliches Vorgehen ist häufig in Frühzeiten des E-Mail-Schriftverkehrs benutzt worden. Damals war nicht abzusehen, dass der Empfänger der Nachricht in der Lage war, Nachrichten in einer Kodierung zu lesen, die nicht dem ASCII-Standard entsprach. Dieser stellte zu dieser Zeit den kleinsten gemeinsamen Nenner der Buchstabenkodierungen dar.

So ist die Verbreitung von Doppelnamen relativ hoch. Bei der Beschreibung des Korpus in Kapitel 5 wird genauer darauf eingegangen. Es existieren aber auch eine Reihe von langen, aus verschiedenen Stämmen zusammengesetzten Familiennamen. Je nach Benutzungskontext einer Phonetischen Suche ist nicht abzusehen, ob es sich um einen Doppelnamen (z. B. <Brand-Meyer>) [19] oder einem zusammengesetzten Namen (z. B. <Brandmeyer>) handelt. Falls der Algorithmus Bindestriche oder Leerzeichen nicht entsprechend behandelt, ist es mit der jeweiligen Suchanfrage nicht möglich, den jeweils anderen Namen zu finden.

Noch komplizierter ist die Lage, wenn die Träger von Doppelnamen in der Datenbank nur mit einer Namenskomponente bekannt sind. Postel (1969) schlägt vor, sämtliche Namenskomponenten in allen Permutationen zu suchen. Dies ist in der Praxis jedoch schwer zu realisieren, wenn nicht eindeutig klar ist, was ein Doppelname ist und was nicht. Abhilfe könnte hier eine Silbentrennung und/oder morphologische Segmentierung schaffen. Ohne größeres Lexikon[20] wäre dies nur mit großer Fehlerquote zu realisieren. Fehlerfreier wäre eine optionale Silbifizierung oder Worttrennung durch den Anwender.

3.6 Bekannte Verfahren für die Phonetische Suche

Die Phonetische Suche ist ein Verfahren, deren erste Varianten schon seit dem letzten Jahrhundert bekannt sind. Ursprünglich wurden anhand der Schlüssel Karteikarten der amerikanischen Volkszählung sortiert. Zwar existieren seit den späten sechziger Jahren Ansätze für die deutsche Sprache, jedoch werden bis in die heutige Zeit hinein vorwiegend Algorithmen aus dem englichsprachigen Bereich eingesetzt, da der Zugang für Programmierer durch vielzählige Implementierungen und weite Verbreitung in der Informatikliteratur für Einsteiger[21] leichter ist.

Es werden vor allem die Algorithmen Soundex, Phonix und Metaphone auch im deutschsprachigen Bereich eingesetzt, obwohl diese für eine englische/amerikanische Aussprache entwickelt wurden und somit nur eingeschränkt auf das Deutsche anwendbar sind. Speziell für die deutsche Sprache konzipiert wurden die ,,Kölner Phonetik", PHONEM und Phonet.

Der Algorithmus nach Daitch und Mokotoff legt seinen Schwerpunkt auf das Jüdische und auf osteuropäische Sprachen. Beide Sprachgruppen haben viele Eigenschaften, die dem Deutschen ähneln, weshalb dieser Algorithmus mit positiven Erwartungen in die Untersuchung aufgenommen wurde.

[19] Ein Test zeigte, dass im Korpus 2243 von solchen, allein in der Orthographie übereinstimmenden Kombinationen vorkommen. Wieviele Varianten gleicher Aussprache noch dazukommen, ist nicht ohne weiteres abzuschätzen. Ich vermute eine obere Grenze bei ca. 0,5 % der deutschen Namen.

[20] Soweit dem Autor bekannt, existieren keine verfügbaren Lexika, die in größerem Umfang Morpheme von Familiennamen enthalten.

[21] Dieser Trend wird durch Übersetzungen englischsprachiger Fachliteratur verstärkt.

	Soundex	E. Soundex	Metaphone	Phonet	Phonet2	Phonix	Daitch-M.	Phonem	K. Phonetik
müller	M460	54600000	MLR	MÜLA	NILA	M4000000	689000	MYLR	657
schmidt	S253	25300000	SKMTT	SHMIT	ZNIT	S5300000	463000	CMYD	8628
schneider	S253	25360000	SKNTR	SHNEIDA	ZNEITA	S5300000	463900	CNAYDR	8627
fischer	F260	12600000	FSKR	FISHA	FIZA	F8000000	749000	VYCR	387
weber	W160	16000000	WBR	WEBA	FEBA	$1000000	779000	VBR	317
meyer	M600	56000000	MYR	MEIA	NEIA	M0000000	619000	MAYR	67
wagner	W256	25600000	WKNR	WAKNA	FAKNA	$2500000	756900	VACNR	367
schulz	S242	24200000	SKLS	SHULS	ZULZ	S4800000	484000	CULC	85
becker	B260	12600000	BKR	BEKA	BEKA	B2000000	759000	BCR	147
hoffmann	H155	15500000	HFMN	HOFMAN	UFNAN	$7550000	576600	OVMAN	036
schäfer	S216	21600000	SKFR	SHEFA	ZEFA	S7000000	479000	CVR	837

Abbildung 2: Die von den im Folgenden untersuchten Verfahren erzeugten Schlüssel für die zehn häufigsten Namen im Korpus.

Bevor die einzelnen Verfahren vorgestellt werden, soll hier schon eine erste Übersicht über die von den Verfahren generierten Schlüssel gegeben werden. In Abbildung 2 ist zu erkennen, dass die Schlüssel je nach Verfahren numerisch, alphanumerisch oder alphabetisch sind. Unterscheiden kann man zwischen Verfahren mit fester und variabler Schlüssellänge.

3.6.1 Soundex

Soundex wurde schon im frühen 20. Jahrhundert bei der amerikanischen Volkszählung verwendet. Ein Patent wurde 1918 und 1922 an Margaret K. Odell und Robert C. Russel erteilt. Wahrscheinlich aufgrund seiner Einfachheit und Geschwindigkeit[22] sind Soundex und seine modifizierten Versionen, die am häufigsten verwendeten Algorithmen zur Kodierung von Namensähnlichkeiten. Dies mag auch an der Beschreibung in frühen Standardwerken der Informatikliteratur liegen, wie z. B. Knuth (1973).

Soundex kodiert die Grapheme durch den Anfangsbuchstaben und numerisch referenzierte Gruppen von Konsonanten. Besonders zeichnet es sich durch die so gewonnene Einfachheit aus. Er ist im englischsprachigen Raum sehr populär, da dort das Verhältnis von Graphemen zu Phonemen auf den ersten Blick nicht so große Probleme bereitet, wie in anderen Sprachen.

Der ursprüngliche Soundexalgorithmus wendet die in Abbildung 3 gezeigten Regeln für die Erzeugung eines Schlüssels an. Erstes Zeichen des Schlüssels ist ein Buchstabe. Dieser wird von dem Anfangsbuchstaben des Eigennamens entnommen, auch wenn es sich bei diesem um einen Vokal handelt.[23] Die folgenden Buchstaben werden solange in die folgenden Ziffern kodiert, bis drei Ziffern gefunden sind. Die Kodierung erfolgt anhand der angegebenen Tabelle. Insbesondere werden Vokale sowie <h> und <w> auf eine Null abgebildet. Diese bleiben aber nur solange bestehen, bis doppelte Zeichen entfernt werden. Das verhindert eine Zusammenfassung von gleichwertigen Konsonanten, die durch einen Vokal

[22] Nach Erikson (1997) ist die Berechnung eines Soundexcodes unter Umständen sogar schneller als der direkte Zeichenkettenvergleich.

[23] Umlaute werden an dieser Stelle allerdings ignoriert.

Code	Soundex	Extended Soundex
1	BFPV	BP
2	CGJKQSXZ	FV
3	DT	CKS
4	L	GJ
5	MN	QXZ
6	R	DGT
7		L
8		MN
9		R

Abbildung 3: Tabelle der Buchstabengruppen und der zugeordneten Ziffern in zwei Versionen von Soundex. Nicht aufgeführte Buchstaben wie z. B. Vokale werden nicht berücksichtigt.

getrennt werden. Ansonsten werden Vokale nicht für die Schlüsselerzeugung berücksichtigt. Wenn der Schlüssel anschliessend weniger als drei Ziffern enthält, wird er mit Nullen aufgefüllt. Schlüssel mit mehr als drei Ziffern werden auf die ersten drei Ziffern beschränkt.

Zum Beispiel wird <Hoffmann> zuerst als H0115055 kodiert. Anschliessend werden doppelte Konsonanten entfernt. Das Zwischenresultat ist H01505. Dann werden die Vokale, also die Nullen getilgt. Nun lautet der Schlüssel H155. Da bereits drei Ziffern enthalten sind, werden keine Nullen mehr angehängt. Aber auch der Name <Heppenheimer> wird mit der H155 kodiert. Erster Schritt ist hier die H01105000507. Nach Entfernung von doppelten Zeichen ist dies die H01050507. Der Schlüssel H1557, welcher nach Eliminierung der Nullen entsteht, ist um einen zu lang. Die 7 am Ende wird also abgeschnitten.

Es bleibt zu vermuten, dass sich Soundex nur sehr wenig für den Einsatz im Deutschen eignet. Insbesondere wird angenommen, dass im Deutschen im Vergleich zum Englischen wesentlich mehr Information über den Kontext von Grapheme, sowie eine Auswertung von Vokalen benötigt wird, damit die Suche nicht zu viele falsche Treffer liefert.

3.6.2 Extended Soundex

Aus dem ursprünglichen Soundex sind über die Zeit hinweg zahlreiche Varianten entstanden. Da dies ein weites Feld ist, über das derzeit keine Literatur berichtet, soll im Rahmen dieser Arbeit lediglich noch die Variante Extended Soundex vorgestellt werden. Diese wird auf diversen Webseiten[24] erwähnt, ein eindeutiger Urheber oder eine Literaturquelle konnte jedoch nicht festgestellt werden.

In der verbesserten Version Extended Soundex wurden gegenüber Soundex nur unwesentliche Änderungen vorgenommen. So wurde die Anzahl der Ziffern optional auf fünf

[24] z. B. http://www.epidata.dk/documentation.php

oder acht erhöht. Je nach Variante wird die Standardkodierung von Soundex weiterverwendet oder es werden neue Zeichengruppen gebildet, wie z. B. in Abbildung 3 angegeben. Der erste Buchstabe wird nicht mehr im Schlüssel verwendet. Anstatt dessen wird seine kodierte Form benutzt. Dies ist insbesondere bei Namen ungünstig, die mit einem Vokal anfangen, da dieser getilgt wird. Wie problematisch dieses Verhalten ist, wird später in der Auswertung der Suchverfahren erörtert werden.

3.6.3 Metaphone

Metaphone wird in Philips (1990) beschrieben und stellt einen weiteren Algorithmus für die englische Sprache dar. In Philips (2000) wird eine verbesserte Version mit dem Namen Double Metaphone vorgestellt. Metaphone benutzt einfache Regeln zur kontextsensitiven Transformation von Zeichen. Die resultierenden Schlüssel haben eine variable Länge und bestehen aus Buchstaben. Wie auch bei Soundex werden Vokale wenig berücksichtigt, jedoch bleiben sie am Anfang der Zeichenkette erhalten. Durch Einbeziehung eines minimalen Kontextes können dabei Grapheme mit mehreren Zeichen, die einem Phonem entsprechen, behandelt werden.

Die beschriebenen Transformationen werden auf den aufeinander folgenden Zeichen der Eingabezeichenkette vorgenommen. Dabei stehen den Erkennungsregeln jeweils das aktuelle Zeichen und das darauf folgende zur Verfügung.[25] Die auszuführenden Aktionen sind zwar variabel, meist beschränkt sich der Algorithmus aber darauf, ein Zeichen an die Zielzeichenkette anzuhängen.

AE	E
(GKP)N	N
WH	H
WR	R
W(AEIOU)	W
X	S
MB	M
TH	0

Abbildung 4: Tabelle einiger Transformation im Metaphone-Algorithmus. Die erste Spalte zeigt eine Zeichenkette, die gesucht wird. Wird diese gefunden, wird sie durch das folgende Zeichen ersetzt. Zeichen in Klammern geben einen Kontext von Zeichen an.

Da die von Metaphone eingesetzten Regeln recht zahlreich sind und die in Computersprachen verbreiteten if-then-else Konstrukte benutzt werden, die sich tabellarisch nur schlecht repräsentieren lassen, soll in Abbildung 4 nur ein kurzer Auszug der Metaphone-Regeln gezeigt werden. Schon aus diesem Auszug ist leicht zu erkennen, dass Metaphone

[25] Es handelt sich bei diesem Algorithmus somit um einen endlichen Automaten, dem die Möglichkeit gegeben wird, ein weiteres Zeichen der Eingabe als Kontext zu lesen.

die Phonotaktik des Englischen stark berücksichtigt.

Damit ergeben sich Konflikte bei der Kodierung deutschsprachiger Zeichenketten. Während dies bei <wr> => <r> noch recht unproblematisch sein sollte, entspricht das <th>[26] im Deutschen dem [t] und nicht dem [θ]. Weiterhin kategorisiert Metaphone das <y>, wenn es vor Vokalen auftritt, nicht als Vokal, sondern als Konsonant. Dies ist im Deutschen in vielen Namen nicht der Fall.

Gegenüber dem ursprünglichen Metaphone ist Double Metaphone um die Fähigkeit erweitert worden, mehrere Schlüssel zu erzeugen. Die Regelbasis wurde an einzelnen Stellen erweitert und es wurden Kodierungsfehler getilgt.

3.6.4 Phonix

Bei Phonix, beschrieben in Gadd (1988) und Gadd (1990), handelt es sich um ein Soundex-Derivat für die englische Sprache. So ist der Schlüssel, wie bei Soundex, eigentlich numerisch. Ist der erste Buchstabe ein Vokal wird dies jedoch durch ein <$> markiert.

Ähnlich wie bei Metaphone, wird versucht, anhand von Regeln einen optimaleren Umgang mit Graphemen zu erlauben, die aus mehreren Zeichen bestehen. Dazu wird eine Ersetzungsfunktion verwendet, wenn eine dazugehörige Regel den Kontext korrekt beschreibt. Die erwähnte Regel besteht aus einer Position, einem gruppierten Kontext und zwei Zeichenketten. Die Position ist entweder beliebig oder der Anfang bzw. das Ende der Zeichenkette. Der Kontext gibt an, ob als benachbarte Zeichen Vokale, Konsonanten oder beides erlaubt sind. Die beiden Zeichenketten geben an, welche Zeichenkette durch eine andere ersetzt werden soll.

Es werden in dem Artikel von Gadd (1988) zwei Varianten von Phonix beschrieben. Die zweite Variante hebt sich von der schon beschriebenen Variante vor allem durch eine gesonderte Behandlung der Endgrapheme ab. Diese werden im zweiten Ansatz eliminiert und erhöhen somit die Trefferquote für den im Artikel vorgesehen Anwendungszweck der Literaturrecherche vorzugsweise von englischsprachigen Titeln.

Phonix gelang eine größere Verbreitung durch die Integration in WAIS.[27]

[26] Dies ist kein kleines Problem für die Anwendung im Deutschen. So enthalten ca. 3,1 Prozent der Familiennamen im Korpus ein <th>.

[27] Bei WAIS handelt es sich um eine spezielle Datenbank für die Informationssuche in unterschiedlichen Dokumententypen. Sie wird vor allem im universitären Kontext und in Bibliotheken benutzt.

3.6.5 Daitch-Mokotoff

Bei dem Daitch-Mokotoff-Algorithmus handelt es sich um ein weiteres Soundex-Derivat. Es ist in Zusammenarbeit von Gary Mokotoff und Randy Daitch entstanden und wird in Mokotoff (2003) beschrieben. Festgelegtes Ziel war ursprünglich eine Anpassung von Soundex an das Jüdische. Später wurde die Regelbasis für osteuropäische Sprachen erweitert. Mit mehrzeichigen Regeln und der Möglichkeit mehrere Schlüssel zu generieren stellt Daitch-Mokotoff die komplexeste Soundexvariante dar.

Daitch-Mokotoff benutzt eine fixe Anzahl von 6 Ziffern als Schlüssel. Der erste Buchstabe wird, wie bei Extended Soundex, als Zahl kodiert. Vokale erhalten in dieser Position der Zeichenkette eine Sonderbehandlung und werden mit einer 0 kodiert. Ansonsten werden sie aber nicht kodiert. Bei der Kodierung wird die Soundextypische Zuordnung von einem Buchstaben zu einem Phon konsequent aufgehoben und es ist somit möglich, mehrzeichige Grapheme direkt einer Kategorie zuzuordnen. Zusätzlich ist es auch möglich in einer Regel mehrere Grapheme auf mehrere Ziffern abzubilden.

Zeichenketten	Alternativen	am Anfang	vor Vokal	ansonsten
AI	AY,AJ	0	1	
CHS		5	54	54
J		1/4	1/4	1/4
K		5	5	5
SZ		4	4	4
S		4	4	4
SZCZ	SZCS	2	4	4
ZDZ	ZDZH, ZHDZH	2	4	4

Abbildung 5: Auszug aus der Tabelle der Transformationsregeln nach Daitch-Mokotoff. Zu den jeweiligen Zeichenketten gibt es gleichwertige, alternative Schreibweisen, die äquivalent behandelt werden. Für die jeweiligen Kontexte wird die Ziffer oder Ziffernfolge angegeben, auf die abgebildet wird. Alternativen, bei denen mehrere Ziffern ausprobiert werden sollen, werden durch ein / getrennt.

Die Eingabezeichenkette wird dabei von links nach rechts auf passende Regeln (für Beispiele siehe Abbildung 5) untersucht. Eine Regel besteht aus drei Komponenten:

- Einer Zeichenkette, die in der Eingabezeichenkette gesucht wird.
- Einer Liste von Ziffernfolgen. Falls es sich um eine einzelne Ziffernfolge handelt, wird diese allen Schlüsseln angehängt. Falls es sich um mehrere Ziffernfolgen handelt, müssen alle Teilschlüssel kopiert werden und eine Kombination mit der jeweiligen Ziffernfolge erstellt werden.
- Einem Kontext. Wie bei Phonix sind hier drei Positionangaben möglich: Start der

Zeichenkette, vor einem Vokal und die beliebige Position. Das Ende der Zeichenkette wird nicht separat berücksichtigt.

Nicht alle Grapheme sind dabei in jedem Kontext erlaubt. Die längste, passende Suchzeichenkette wird immer präferiert. Wenn Suchzeichenkette und Kontext in der Eingabezeichenkette übereinstimmen, wird die Ziffernfolge dem bisher kodierten Schlüssel angehängt, bzw. bei mehreren Schlüsseln entsprechende Varianten erzeugt. Durch die Variantenbildung ist es möglich, Ambiguitäten in der Aussprache zu behandeln. Von den anderen Verfahren versucht nur Phonix diese Eigenschaft umzusetzen.

Für die Kodierung des Namens <Hirsch> wird folgendermaßen vorgegangen: Der erste Buchstabe ist ein <h>. Dieser wird mit einer 5 kodiert. Der darauf folgende Vokal vorerst mit einer 0. Da das <rs> mehrere Varianten besitzt, wird es entweder mit einer 94 oder mit einer 4 kodiert. Nun existieren bereits zwei Teilschlüssel 5094 und 504. Das folgende <ch> wird entweder mit 5 oder 4 kodiert. Nun existieren bereits vier Varianten 50945, 5094, 5045, 5044. Nun werden die doppelten Zeichen entfernt. Hier wird lediglich aus dem 5044 eine 504. Nach Entfernen der Vokale und Auffüllen der Schlüssel mit Nullen, bleiben die Kodierungsvarianten 594500, 594000, 545000 und 540000.

3.6.6 Die ,,Kölner Phonetik"

Das Verfahren ,,Kölner Phonetik" wurde von Postel (1969) veröffentlicht. Obwohl der Name es suggeriert, hat Sie nichts mit dem Kölner Institut für Phonetik zu tun, sondern wurde aus unbekannten Gründen derart benannt. Es handelt sich hierbei um einen frühen Ansatz, Soundex an das Deutsche anzupassen. Ähnlich wie auch bei Soundex wird eine Zuordnung von Zeichen auf Ziffern vorgenommen. Für die Auswahl der jeweiligen Ziffer wird maximal ein Buchstabe als Kontext benutzt. Dieser Buchstabe kann jedoch auf beiden Seiten des ausgewerteten Zeichens stehen. Für den Wortanfang stehen wie bei Phonix oder Daitch-Mokotoff für einzelne Grapheme spezielle Regeln zur Verfügung. Für einen Überblick der Ersetzungsregeln siehe Abbildung 6.

Wie bei den anderen an Soundex angelehnten Verfahren ist die Behandlung von Vokalen nur rudimentär gegeben. Außer am Wortanfang und im vorhin erwähnten Entscheidungskontext werden diese nicht berücksichtigt. Bedingt durch den frühen Zeitpunkt[28] der Publikation werden Umlaute und das ß nicht berücksichtigt.

Die ,,Kölner Phonetik" ist heute noch Bestandteil von Ausschreibungen im öffentlichen Verwaltungsbereich. Darüber hinaus hat Sie wenig Verbreitung gefunden.

[28] In den Frühzeit der Datenverarbeitung waren Zeichenkodierungen, die Umlaute kannten, nur sehr wenig verbreitet. Der Autor arbeitete mit einer IBM /360, die eine eigene Kodierung benutzte.

Zeichen	Kontext	Symbol
A,E,I,J,Y,O,U	im Anlaut	0
H		-
B,P		1
D,T	nicht vor C,S,Z	2
F,PH,V,W		3
G,K,Q		4
C	im Anlaut, vor A,H,K.L,O,Q,R,U,X	4
	ansonsten, vor A,O,U,H,K,X,Q	
X	wenn nicht nach C,K,Q	48
L		5
M,N		6
R		7
S,Z		8
C	im Anlaut, nicht vor	
	A,H,K,L,O,Q,R,U,X folgt	
	ansonsten, nicht vor A,O,U,H,K,X,Q	
	nach S, Z	
D,T	vor S,C, Z	
X	nach C,K,Q	

Abbildung 6: Ersetzungsregeln der ,,Kölner Phonetik" nach Postel (1969).

3.6.7 PHONEM

In Wilde u. Meyer (1988) wird ein Verfahren vorgestellt, das auf der Basis von Buchstabenersetzungen arbeitet. Der hier benutzte Name PHONEM wurde gewählt, obwohl der Autor keinen eindeutigen Namen vergeben hatte. So wird im Artikel auch von Phonemwort gesprochen. Der Name PHONEM entspricht dem der Funktion für dBase, deren Implementierung in 8086-Assembler[29] dem Artikel beiliegt.

Der eigentliche Algorithmus arbeitet direkt auf der Zeichenkette. Die Zeichenkette wird von links nach rechts mit einer Liste von zwei Buchstaben langen Zeichenketten verglichen und bei Übereinstimmung ersetzt. Ein zweiter Schritt wiederholt diesen Vorgang mit einzelnen Buchstaben. Eine Zusammenfassung der Ersetzungen findet sich in Abbildung 7. Anschliessend werden in der gesamten Zeichenkette eventuell vorhandene Leerzeichen, doppelte und nicht erlaubte Zeichen entfernt. Erlaubte Zeichen sind ,,ABCDLMNORSUVWXYÖ".

Für den Namen <Mueller> wird der Schlüssel folgendermassen erzeugt: Das Vergleichen der zwei Buchstaben langen Zeichenketten beginnt mit dem <Mu>. Hierfür ist keine Regel hinterlegt, also wird nichts verändert. Das <ue> wird durch ein <y> ersetzt. Für

[29]Es handelt sich hier um eine nur eingeschränkt nutzbare Form der Implementation. Die Programmiersprache wurde wahrscheinlich, dem damaligen Zeitgeist entsprechend, der Geschwindigkeitsvorteile wegen gewählt.

Zeichen in der Zeichenkette	Kodierung	Zeichen in der Zeichenkette	Kodierung
Z,K,G,Q	C	KS	X
A,AE	E	QU	KW
U,I,J	Y	OE	Ö
F,W,PF	V	EI,EY	AY
P	B	EU	OY
T	D	OU	U
SC,SZ,CZ,TZ,TS	C		

Abbildung 7: Ersetzungen im PHONEM-Algorithmus. Regeln für Vokale mit Akzenten wurden ausgelassen.

<le> und <er> sind ebenfalls keine Regeln vorhanden. Somit ist der erste Zwischenschlüssel <myller>. Das Ersetzen der einzelnen Buchstaben bringt auch keine Veränderungen. Nun werden doppelte Zeichen entfernt. Da das <e> nicht Bestandteil der erlaubten Buchstaben ist, wird dieses entfernt. Damit ist der Schlüssel <mylr>.

Die Verbreitung des Verfahrens PHONEM dürfte an der engen Koppelung an dBase[30] gescheitert sein. Portierungen[31] sind nicht bekannt.

3.6.8 Phonet

Phonet ist der derzeit neueste, bekannte Ansatz für eine Phonetische Suche im Deutschen. Zwei Varianten[32] werden in Michael (1988) beschrieben, die sich durch die Größe der Zielalphabete unterscheiden. Phonet zeichnet sich gegenüber den bisherigen deutschsprachigen Ansätzen durch seine relativ hohe Komplexität[33] und die Verwendung einer großen Anzahl von Regeln aus. Phonet versucht dabei, der Bedeutung der Vokale im Deutschen Rechnung zu tragen. Allerdings wird wie bei PHONEM in der ersten Variante eine relativ große Anzahl an Vokalklassen verwendet. Die zweite Variante ist im Bereich der Vokale deutlich variabler.

Die Eingabezeichenkette wird bei Phonet mittels zweiteiliger Regeln verändert. Durch eine Erkennungszeichenkette wird der Kontext beschrieben in dem eine Regel verwendet wird. Diese Zeichenkette hat eine ähnliche Syntax wie die in Kapitel 8.1 beschriebenen regulären Ausdrücke. Durch zusätzliche Steuerzeichen ist es möglich nur Teile der Eingabezeichenkette zu verändern. Die zweite Zeichenkette enthält die Zeichen, welche in der

[30]Bei dBase handelt es sich um eine einfache Datenbank, die vor 15 Jahren zwar weit verbreitet, heute jedoch fast bedeutungslos geworden ist.

[31]Unter einer Portierung versteht man das Anpassen einer Software an andere Umgebungen, wie z. B. Programmiersprache oder Betriebsystem.

[32]Die zweite Variante von Phonet wird im Folgenden Phonet2 genannt.

[33]Der Ansatz von Phonem ähnelt einem cluster-basierten Graphem-nach-Phonem-Konverter. Da keine konkrete Implementation bekannt ist, kann lediglich abgeschätzt werden, dass die Komplexität ähnlich ist.

Eingabenzeichenkette eingefügt werden. Für die beiden Varianten von Phonet existiert je eine Variante dieser zweiten Zeichenkette. Die Erkennungsregel wird von beiden Varianten benutzt.

Da jede dieser Regeln auf die eben beschriebene Art die Eingabezeichenkette modifizieren kann, ist es notwendig, die Reihenfolge der Regeln zu beachten. Diese Einschränkung erhöht jedoch den Aufwand neue Regeln zu erzeugen, da Abhängigkeiten berücksichtigt werden müssen, um fehlerhafte Kodierung zu vermeiden.

Die ursprünglich mit dem Artikel veröffentlichte Version von Phonet enthielt ca. 650 Regeln. Eine neuere Version enhält mittlererweile 850 Regeln. Wie eine später vorgenommene, genauere Untersuchung der Regeln zeigen wird, sind ca. 155 der Regeln für die Familiennamen im benutzten Korpus überhaupt nicht relevant. Es kann nur vermutet werden, dass durch diese Regeln sehr spezielle Besonderheiten von Vornamen oder Fremdworten berücksichtigt werden. Wegen der großen Anzahl der vorhandenen Regeln soll in Abbildung 8 nur eine kleine Auswahl an Regeln präsentiert werden, um einen Einblick in das Verfahren zu geben.

Erkennungsregel	Kodierung bei Phonet1	Kodierung bei Phonet2
AUX	O	U
AU	AU	AU
AVER–<	AW	
AVIER$	AWIE	AFIE
AV(EÈÉÊI)-	AW	
AV(AOU)-	AW	
AYRE$	EIRE	EIRE
AYRE(NS)$	EIRE	EIRE
AYRE(AIOUY)–	EIR	EIR
AYR(AÄIOÖUÜY)-	EIR	EIR
AYR<	EIA	EIA
AYER–<	EI	EI
AY(AÄEIOÖUÜY)–	A	A
A(IJY)<	EI	EI

Abbildung 8: Auszug der von Phonet benutzten Regeln. Die erste Spalte ist die Erkennungszeichenkette, die beiden anderen Spalten enthalten die jeweilige Kodierung für Phonet und Phonet2. In der Erkennungsregel werden Zeichen wie das - verwendet, um die Position der zu ersetztenden Zeichen zu markieren. So wird in AVER– lediglich das <AV> durch ein <AW> ersetzt.

Für die Anwendung der Regeln ist eine Beispielimplementation in C[34] verfügbar. Auf-

[34]Bei C handelt es sich um eine Programmiersprache, die einerseits recht schnell ist, andererseits aber auf einer Vielzahl von Rechnerplattformen verfügbar ist.

grund der Komplexität dieser Implementation und der nur spärlich vorhanden Dokumentation der Regeln ist der Zugang jedoch sehr schwer.

3.6.9 IPA

Hier soll noch die Möglichkeit erwähnt werden, eine phonetische Transkription in einem geeignetem Alphabet[35] vorzunehmen. Für das Englische sind mehrere solcher Systeme z. B. in Zobel u. Dart (1996) oder Lutz u. Greene (2003) vorgestellt worden. Verfügbare Implementationen sind aber auch für das Englische nicht bekannt. Da bisher kein verwendbarer Graphem-nach-Phonem-Konverter für das Deutsche veröffentlicht wurde, wurde dieser Ansatz vorerst nicht weiter verfolgt. Ein typischer Konverter dürfte jedoch durch seine ungleich größere Komplexität gegenüber einer Phonetischen Suche nicht unbedingt für den Anwendungszweck einer direkten Suche geeignet sein.[36] Zudem sind in der Literatur umrissene Konverter meist nicht unter Berücksichtigung von Eigennamen konzipiert worden.

In Lutz u. Greene (2003) wird dementsprechend vorgeschlagen, die Orthograpie der Eigennamen anhand geeigneter Regeln automatisch in ein Zeichensystem nach IPA-Alphabet zu überführen. Es wird ein grober Überblick für ein solches System[37] gegeben, welches jedoch nur kommerziell angeboten wird. Durch die Genauigkeit des IPA-Alphabetes eignet sich die erzeugte Zeichenkette jedoch nicht mehr als direkter Schlüssel für eine Suche. Für einen Vergleich der Transkriptionen wird ein gewichtetes Verfahren auf Basis der ,,edit-distance"-basierten Algorithmen vorgeschlagen, wie sie im folgenden Kapitel beschrieben werden.

Belhoula (1993) stellt eine Idee für die Konvertierung von Graphemen nach Phonemen speziell für deutsche Namen vor, die auf Buchstaben-Clustern basiert. Er gibt Beispiele für eine morphologische Trennung von Familien- und Ortsnamen. Auf Basis eines Korpus, der ca. 130.000 Familiennamen enthält, wird eine Häufigkeitsanalyse für Morpheme vorgenommen. Ein Schwerpunkt liegt bei der Betonung sowie bei der Vorhersage der Länge von Vokalen. Schon aus den dort angegebenen Beispielen zeigt sich jedoch, dass ein solches Vorgehen sehr aufwendig ist. So wird als häufigstes Morphem <mann> angegeben. Die-

[35] Hierfür würde sich z. B. SAMPA oder die IPA-Notation anbieten.

[36] In Damper u. a. (1999) findet ein Vergleich von verschiedenen Methoden der Konvertierung von Graphemen nach Phonemen statt. Verglichen werden von Experten entworfene, regelbasierende Systeme mit verschiedenen Varianten von selbstlernenden Systemen, deren Komplexität jedoch die praktische Anwendung unwahrscheinlich werden lassen. Das regelbasierte System schneidet in diesem Vergleich sehr schlecht ab. Allerdings stellt der Anwendungszweck der Text-to-Speech-Systeme sehr viel genauere Anforderungen an die Performanz der Verfahren, als es für den Vergleich von Zeichenketten nötig ist. Hier wäre zu evaluieren, ob ein regelbasiertes System als Grundlage dafür dienen kann.

[37] Das vorgestellte System kann wohl mit Transliterationen aus dem Chinesischen und mit spanischen Namen umgehen. Genauere Angaben ließen sich dem Artikel jedoch nicht entnehmen. Da es sich um ein kommerzielles Produkt handelt sind außer den Rahmenbedingungen keine genaueren Information ersichtlich.

ses ist jedoch lediglich 364 Mal in dem dort verwendetem Korpus vorhanden. Im Schnitt wurden für die Beispiele Häufigkeiten um die 30 angegeben. Selbst wenn alle Morpheme ähnlich häufig und alle Namen zerlegbar wären, hätte die benötigte Regelbasis für die Umsetzung von Morphemen bereits über 3500 Regeln. Da ein Name im Zweifelsfall aus mehreren Morphemen besteht, kann man schnell ausrechnen, dass sich die Berücksichtigung selbst für besonders häufige Morpheme nicht lohnt. Vor allem dann nicht, wenn diese ähnlich leicht auf Phoneme abzubilden sind, wie das Beispielmorphem <mann>.

4 Klassische Verfahren für den Vergleich von Zeichenketten

Der im Folgenden vorgestellte Typ von Verfahren dient dem Vergleich von zwei Zeichenketten. Im Gegensatz zu den bisher vorgestellten Verfahren wird kein Wissen über den Inhalt der Zeichenkette benutzt. Auch wird keine weitere Kodierung benötigt. Sie sind somit universell einsetzbar, haben dafür eine Reihe anderer Schwierigkeiten mit der Verarbeitung von natürlichsprachigen Zeichenketten. Vor allem das schon erwähnte Alignment zwischen Graphemen und Phonemen bereitet Probleme.

4.1 Einführung

In ihrer Reinform handelt es sich bei dieser Kategorie von Algorithmen um logische Verfahren, die ein Maß der Ähnlichkeit von zwei Zeichenketten ermitteln. Sie sind für eine Phonetische Suche nur unter der Prämisse zu gebrauchen, dass ähnlich gesprochene Wörter auch eine ähnliche Orthographie besitzen. Durch den rein logischen Charakter sind diese Algorithmen sprachunabhängig, wobei zu vermuten ist, dass je nach Komplexität der Orthographie der verwendeten Sprache unterschiedliche Ergebnisse zu erwarten sind.

Hier sollen Hamming-Distanz, N-Gram-basierte Suche und Damerau-Levenshtein exemplarisch vorgestellt werden. Für den interessierten Leser findet sich bei Gusfield (1997) eine weitergehende Zusammenstellung von vielfältigen Verfahren. Der Anwendungsschwerpunkt liegt hier in der Bioinformatik. Eine detailliertere Betrachtung der Resourcenabschätzung für Variationen der Damerau-Levenshtein-Algorithmen findet sich in Navarro (2001).

Je nach Algorithmus kann für eine jeweilige Sprache eine Gewichtung nach phonetisch/phonologischen Kriterien erfolgen. So ist es möglich zuerst phonetische Schlüssel mit einem der Verfahren für die Phonetische Suche zu generieren und die Schlüssel dann mit einem der vorgestellten Verfahren zu vergleichen, um Fehler oder Ambiguitäten der Kodierung zu berücksichtigen.

Übliche Anwendungsgebiete für edit-distance und N-gram-basierte Algorithmen sind im Bereich natürlicher Sprache vor allem die Rechtschreibkorrektur, aber auch die Sprach-

und Schrifterkennung. Die vorgestellten Verfahren werden zum Teil auch für die Erkennung ähnlicher Genomsequenzen in der Bioinformatik benutzt.

4.2 Anwendungsmöglichkeit für eine Phonetische Suche

Damerau-Levenshtein und Hamming-Distanz eignen sich nicht unbedingt für eine Phonetische Suche auf großen Datenbanken, da für eine Suche die Eingabezeichenkette mit dem gesamtem Datenbestand verglichen werden muss. Deshalb sind Kriterien für die Einschränkung der zu tätigenden Vergleiche wünschenswert. Bei kleineren Datenbanken kann dieser Aspekt vernachlässigt werden.

Für N-Gram-basierte Verfahren ist eine Anwendung in relationalen Datenbanken denkbar. Für eine Suche muss jedoch ein Index der N-Gramme erstellt werden. Dies ist eine Aufgabe, die viel Speicherplatz benötigt. Somit sollte in großen Datenbanken erst eine Vorauswahl (Partitionierung) der Daten vorgenommen werden, bevor diese Klasse von Algorithmen angewendet wird. Dieser Punkt wird in Kapitel 8 noch einmal angesprochen werden.

Hinzu kommt, dass bei Hamming-Distanz und Damerau-Levenshtein bei kurzen Zeichenketten die Wahrscheinlichkeit groß ist, daß mit wenigen Operationen eine vollkommenen andere Zeichenkette kurzer Länge gefunden werden. Dies macht die angegebenen Algorithmen sehr fehleranfällig und erfordert eine Gewichtung nach Länge der Zeichenkette. Das Auffinden einer optimalen Gewichtung für Zeichenlängen, wäre Aufgabe weiterer Untersuchungen.

4.3 Hamming-Distanz

Die Hamming-Distanz ist die einfachste Form des Stringvergleichs. Sie zählt die Anzahl der Zeichen gleicher Position in beiden Eingabezeichenketten, die unterschiedlich sind. Sie kann im Gegensatz zu den folgenden Verfahren schnell errechnet werden. Sie hat für den Einsatz im natürlichsprachigen Vergleich von Zeichenketten allerdings wenig Aussagekraft, da verschobene Teilzeichenketten nicht berücksichtigt werden, wie das Beispiel in Abbildung 9 zeigt.

M	ü	l	l	e	r
M	ö	l	l	e	r
0	1	0	0	0	0

M	ü	l	l	e	r	
M	u	e	l	l	e	r
0	0	1	0	2	3	4

Abbildung 9: Berechnung der Hamming-Distanz durch Aufaddieren der unterschiedlichen Buchstabenpositionen. Während eine Distanz von eins im linken Beispiel Müller vs. Möller ein ganz gutes Maß für Ähnlichkeit ist, ist die Distanz von vier im rechten Beispiel <Mueller> vs <Müller> als Ähnlichkeitsmaß indiskutabel.

4.4 Levenshtein und Damerau

Bei der Levenshtein-DistanzLevenshtein (1965) handelt es sich um eine Methode, die Ähnlichkeit zwischen zwei Zeichenketten zu berechnet. Für die Berechnung der Distanz wird die Anzahl der Operationen errechnet, die diese Zeichenketten voneinander entfernt sind. In diesem Verfahren wird ein mögliches Einfügen, Löschen und Austauschen von jeweils einem Zeichen berücksichtigt. Damerau, der einen ähnlichen Ansatz verfolgt, verfügt über die Möglichkeit der Transposition zweier Zeichen[38], ein Fehler, der auch bei geübten Schreibern häufig vorkommt Kukich (1992). Für einen phonetischen Vergleichs von zwei Zeichenketten ist Transposition jedoch weniger interessant.

Der ursprüngliche Algorithmus stammt aus der dynamischen Programmierung. Es wird im Speicher eine zweidimensionale Matrix aufgebaut, die diagonal von der oberen, linken Ecke aus gefüllt wird. Dazu wird für jede noch nicht ausgefüllte Zelle der Matrix überprüft, welche Kosten die bereits ausgefüllten benachbarten Zellen bereits haben. Der jeweils geringste Wert wird übernommen und um die Kosten der Einfügeoperation[39] modifziert. Falls die der aktuellen Zelle entsprechenden Buchstaben ungleich sind, werden die Kosten für eine Ersetzung addiert. Ein Beispiel für eine solche Matrix findet sich in Abbildung 10.

		M	ü	l	l	e	r
	0	1	2	3	4	5	6
M	1	0	1	2	3	4	5
u	2	1	1	2	3	4	5
e	3	2	2	2	3	3	4
l	4	3	3	2	2	3	4
l	5	4	4	3	2	3	4
e	6	5	5	4	3	2	3
r	7	6	6	5	4	3	2

Abbildung 10: Berechnung der Levenshtein-Distanz für Müller vs. Mueller. Die Operationen Einfügen, Löschen und Ersetzten haben ein Gewicht von 1. Die hier gezeigte Matrix zeigt die Distanz-Werte, die während des laufenden Algorithmus erzeugt werden. In der rechten unteren Ecke findet sich am Ende die Gesamtdistanz. Z. B. 1 für <u> => <ü> und 1 für das Tilgen des <e>.

Der Ansatz aus der dynamischen Programmierung zeichnet sich durch ein ungünstiges Laufzeitverhalten aus, ist aber immer noch sehr gut geeignet, um den eigentlichen Vorgang zu erklären. Neuere Algorithmen wie z. B. Hyyrö (2003) sind besonders bei der Verarbeitung von langen Zeichenketten schneller und benutzen Optimierungsmöglichkeiten, die auf

[38] Z. B. <schmidt> vs <shcmidt>. Hier hat die rechte Hand das <h> schon getippt, bevor die linke das <c> eingeben konnte.

[39] Man beachte, dass Einfügeoperation und Löschoperation von den Kosten her identisch sind. So entspricht ein Einfügen in der einen Zeichenkette einem Löschen in der anderen.

Eigenschaften der im ursprünglichen Algorithmus aufgebauten Matrix aufbauen. Nirgendwo in der Literatur wurde aber eine Abschätzung des Laufzeitverhaltens verschiedener Varianten für sehr kurze Zeichenketten[40] vorgenommen, wie sie für den Vergleich von einzelnen Namen interessant sind.

Falls die zu errechnende Gesamtdistanz einen Schwellwert haben soll, wie es bei Suchfunktionen meist der Fall ist, kann der Algorithmus durch ein Stopkriterium beschleunigt werden. Dies wird z. B. von Erikson (1997, S.20) vorgeschlagen. Dabei wird die Berechnung abgebrochen, sobald ein Schwellwert erreicht wird.

An den vorgestellten Beispielen zeigt sich, dass die ,,edit-distance"-basierten Verfahren große Probleme mit Graphemen haben, die Homonyme mit einer unterschiedlichen Anzahl von Zeichen besitzen. So zeigt das vorgestellte Beispiel <Mueller> vs. <Müller> eine Distanz von 2, obwohl die Aussprache identisch ist. Für eine Angleichung ist es also wünschenswert, alle Grapheme in eine Repräsentation mit gleicher Zeichenlänge zu bringen, eine Eigenschaft, die die meisten Verfahren für die phonetische Schlüsselgenerierung erfüllen. Für diesen Zweck sei nochmals auf die schon im vorherigen Kapitel erwähnte Möglichkeit der automatischen Erzeugung von Transkriptionen in Formaten, wie SAMPA oder dem IPA-Alphabet hingewiesen.

Weiterhin wurde mehrfach[41] vorgeschlagen, eine Gewichtung für die Ersetzung von Zeichen einzuführen. So könnten Grapheme, die phonetisch sehr ähnlich sind, geringeres Gewicht haben, als Grapheme, die sehr unterschiedlichen Lauten entsprechen. So sind sich die Phone /b/ und /p/ sehr viel ähnlicher, als dies bei einem /p/ und einem /ʃ/ der Fall ist. Erstere unterscheiden sich lediglich in dem Merkmal Stimmhaftigkeit, während bei dem zweiten Beispiel eine andere Artikulationsstelle und eine andere Artikulationsart vorliegt. Eine mögliche Quelle für eine solche Gewichtung für das Deutsche wären die Ähnlichkeitsuntersuchungen von Transkriptionen aus Vieregge (1985). Hier wird für Phone des Deutschen eine Ähnlichkeitsmatrix angegeben, welche auf distinktiven Merkmalen beruht. Die Gewichtung dieser Matrix wurde in empirischen Studien zu menschlichen Transkriptionsverhalten verifiziert und dürfte auch für den vorgeschlagenen Zweck anwendbar sein.

4.5 N-Gram basierte Verfahren

Bei den N-Gram-basierenden Verfahren werden aus den zu vergleichenden Zeichenketten alle Zeichenkombinationen vorgegebener Länge extrahiert. Ein Vergleich der Teilzeichenketten beruht auf der Annahme, dass sich zwei Zeichenketten ähnlich sind, wenn eine große

[40] Dies wäre ein Punkt, der noch zu untersuchen wäre, aber leider auch den Rahmen dieser Arbeit sprengen würde.

[41] Z. B. von Erikson (1997) und Navarro (2001)

Anzahl von Zeichenkombinationen gleicher Länge übereinstimmen.

In der Literatur wird für diese Verfahren unter anderem auch der Begriff Q-Gram benutzt. Eine genauere Bezeichnung ergibt sich bei einer fixen Länge der extrahierten Zeichenketten. Diese werden bei Kombinationen von zwei Zeichen Bi- oder Digramme sowie für eine Teilzeichenkettenlänge von drei Trigramme genannt.

M	Mü	Mül	üll		lle	ler	er	r
M	Mu	Mue	uel	ell	lle	ler	er	r
X					X	X	X	X

Abbildung 11: Beispiel für die Zerlegung der Zeichenkette <Müller> und <Mueller> in Trigramme. Die Namen unterscheiden sich in vier Trigrammen. Fünf sind in beiden enthalten und wurden mit einem X markiert.

Anhand der Anzahl von übereinstimmenden N-Grammen, kann eine orthographische Ähnlichkeit bestimmt werden. Ein Beispiel für einen Vergleich von Trigrammen in den Namen <Müller> und <Mueller> gibt Abbildung 11. Naturgemäß ist die Anzahl von übereinstimmenden N-Grammen für lange Zeichenketten aussagekräftiger, da hier für eine Ähnlichkeit mehrere Trigramme übereinstimmen. Es empfiehlt sich deshalb eine Gewichtung nach der Zeichenkettenlänge.

Ein Vorteil gegenüber anderen edit-distance-Algorithmen ist die Möglichkeit, einen Index von Trigrammen anzulegen und damit relativ effizient über Mengenoperationen für eine Sucheingabe andere Zeichenketten zu finden, die eine vorgegebene Anzahl von äquivalenten N-Grammen enthält.

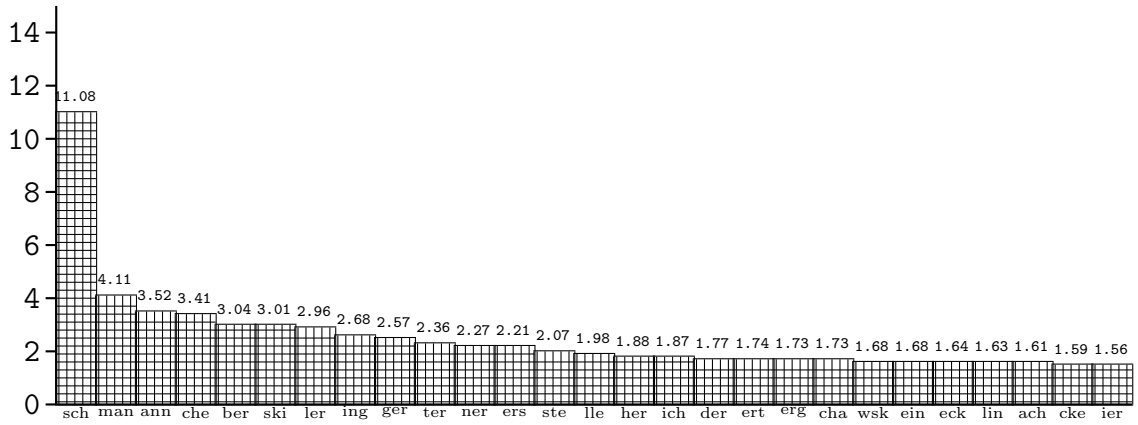

Abbildung 12: Die 25 häufigsten Trigramme im Korpus. Die Zahlenangaben sind in Prozent der Gesamtanzahl der Einträge angegeben. Sonderzeichen wie Bindestriche und ' wurden beim Erstellen des Trigramindexes nicht berücksichtigt.

Darüber hinaus eignen sich die Zerlegung und Indizierung von Trigrammen für die sta-

tistische Untersuchung von Zeichenketten. So ist es möglich, häufig auftretende Zeichenkombinationen zu identifizieren und besonders zu berücksichtigen. Für den verwendeten Korpus wurde ein Index von Trigrammen angelegt. Abbildung 12 zeigt die 25 häufigsten Trigramme im Korpus.

5 Der verwendete Korpus

Bei dem verwendeten Korpus handelt es sich um eine Liste von ca. 1,12 Millionen Familiennamen mit unterschiedlichen Schreibweisen. Die Namen wurden mit Hilfe der Beschreibung in Erdgeist (2002) aus einer Telefonbuch CD-ROM Telekom (2000) extrahiert, welche ein Telefonverzeichnis für ganz Deutschland enthält.

Der Korpus wird im Folgenden als Datenbasis für Suchabfragen verwendet. Da angenommen wird, dass der Korpus für deutsche Familiennamen repräsentativ ist, wird er außerdem für die Gewinnung von statistischem Material über die Häufigkeit von Graphemkombinationen verwendet.

5.1 Gewinnung der Daten

Von der CD-ROM wurden aus den Binärdaten alle Datensätze aus dem Namensfeld extrahiert, die der Form von Familiennamen entsprachen. Für diesen Zweck wurde angenommen, dass ein Nachname höchstens aus Umlauten, Buchstaben, dem Zeichen ' und höchstens einem Bindestrich besteht. Ein Name, der einen Bindestrich enthält, wird im Folgenden als Doppelname gewertet.

Namenszusätze wie z. B. ,,von" oder ,,de" werden innerhalb des verwendeten Telefonbuchs in einem anderem Feld gespeichert. Sie wurden in dieser Untersuchung ignoriert[42].

Insgesamt konnten auf diese Weise ca. 1,18 Millionen unterschiedliche Einträge ermittelt werden, die der oben beschriebenen Form entsprechen. Die Daten enthielten jedoch noch Institutionen, Firmennamen und Abkürzungen. Da diese nicht Bestandteil dieser Untersuchung sein sollen, wurden diese Einträge semiautomatisch bereinigt. Dazu wurde zum einen gezielt nach Zeichenketten wie z. B. ,,GmbH", ,,Gesellschaft", ,,Institut" gesucht, zum anderen wurde versucht, die Namen zu entfernen, bei denen es sich offensichtlich um Abkürzungen[43], Kunstnamen, öffentliche Institutionen oder Gewerbe handelt. Insgesamt verringerte sich dadurch die Anzahl der Einträge um ca. 63.000.

[42] Siehe hierzu auch Postel (1969). Dieser zeigt Fälle, in denen diese bei einer Suche Bedeutung haben.

[43] Als Abkürzungen wurden alle Einträge gewertet, die keine Vokale enthalten und kürzer als fünf Zeichen lang sind.

5.2 Statistik der Namen

Für statistische Zwecke wurde im Rahmen dieser Arbeit bei der Extraktion die Häufigkeit gleicher Familiennamen gezählt. Es wurden dabei keine Versuche unternommen, doppelte Telefonanschlüsse einer Person zu identifizieren. Damit sind die genannten Zahlen lediglich als Anzahl der registrierten Anschlüsse zu werten. Wie viele Personen keine veröffentlichten Anschlussdaten oder überhaupt keine Anschlüsse besitzen, ist nicht bekannt.

In Kunze (2003) wird eine ähnliche Statistik für das Jahr 1995 beschrieben. Es wird eine Anzahl von ca. 960.000 verschiedenen Familiennamen angegeben. Da dort keine genauere Quelle für die Daten und die Methode der Datenextraktion, sowie der Behandlung von Doppelnamen angegeben wurde, kann kein direkter Vergleich vorgenommen werden. So gibt Kunze (2003) einen Faktor von durchschnittlich 2,8 Personen pro Anschluss an. Ob dieser auch noch im Jahr 2000 gültig ist, war nicht überprüfbar. Es ist jedoch davon auszugehen, dass sich die Anzahl der Telefonanschlüsse im Jahr 2000 gegenüber 1995 vergrößert hat.[44]

Nach der beschriebenen Bereinigung der Korpusdaten enthielten von den 1.118.653 Millionen Namen 240.828 einen Bindestrich und wurden somit als Doppelnamen gewertet. Dies entspricht einer Doppelnamenquote von 21,55 %. Nach einer Aufteilung der Doppelnamen in ihre Teile, ergab sich eine Gesamtanzahl von 895.993 unterschiedlichen Familiennamen.

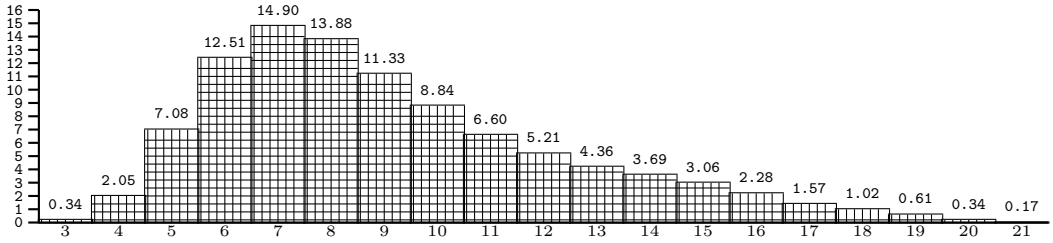

Abbildung 13: Verteilung der Zeichenanzahl von Familiennamen im untersuchten Korpus in %. Aus Platzgründen wurden Namen mit mehr als 21 Zeichen weggelassen. Sie machen insgesamt weniger als 0,05 % der Einträge aus.

Bei der Betrachtung der Länge der Familiennamen in Abbildung 13 fällt in der Verteilung ein deutlicher Höhepunkt im Bereich von sechs bis neun Zeichen auf. Mit jedem zusätzlichen Zeichen reduziert sich die Häufigkeit von längeren Namen. Die längsten Namen im Korpus, z. B. <qinteirobarrasdasilvaoliveira> oder <walischewski-dittmannsberger>, sind bis zu 29 Zeichen lang. Vor der Filterung war <Kiechlingsbergen-Königschaffhausen> der längste Name. Nach kurzer Recherche ließ sich dies aber als Ortsname

[44]Laut Angaben des Statistischen Bundesamtes hatte Deutschland im Jahr 2000 82,26 Millionen Einwohner. Insgesamt entsprechen die 1,12 Millionen Familiennamen ca. 36 Millionen Telefonanschlüssen. Daraus lässt sich ein Faktor von ca. 2,28 errechnen.

identifizieren. So sind vermutlich ein geringer Anteil der Namen im Korpus Ortsnamen, jedoch sind diese nicht mit vertretbarem Aufwand zu entfernen. Da Ortsnamen viele Ähnlichkeiten mit Familiennamen haben, ist nicht davon auszugehen, dass das Ergebnis dadurch signifikant beeinflusst wird.

müller	291322	krüger	48086	kaiser	35511	baumann	25794	heinrich	20882
schmidt	214647	braun	47346	fuchs	35006	franke	24914	haas	20621
schneider	128798	hofmann	46656	peters	34529	albrecht	24777	schreiber	20476
fischer	110357	lange	45739	scholz	33728	ludwig	24257	graf	20081
weber	96048	hartmann	45657	möller	33563	simon	24217	schulte	19619
meyer	94783	schmitz	44416	lang	33248	schuster	24192	dietrich	19588
wagner	88823	krause	44372	weiß	33106	böhm	24155	kühn	19391
schulz	84422	schmitt	44301	jung	31331	winter	23657	kuhn	19278
becker	83302	werner	44149	hahn	30504	kraus	23462	ziegler	19267
hoffmann	80521	meier	42050	schubert	29995	schumacher	23412	pohl	19124
schäfer	68235	lehmann	41721	vogel	29819	martin	23180	engel	19008
koch	67626	schmid	40509	friedrich	29620	krämer	22962	horn	18759
richter	67204	schulze	39647	günther	29540	vogt	22707	busch	18555
bauer	66568	köhler	38607	keller	29131	otto	22643	voigt	18328
klein	60758	maier	38450	winkler	28662	stein	22617	sauer	18319
wolf	57540	herrmann	38307	berger	28617	jäger	22441	bergmann	18306
schröder	57463	könig	37399	frank	28593	groß	21606	thomas	18216
neumann	53988	walter	37312	roth	28307	sommer	21446	wolff	18059
schwarz	49770	mayer	36442	beck	27669	seidel	21215	arnold	17845
zimmermann	48800	huber	35759	lorenz	26798	brandt	20958	beyer	17843

Abbildung 14: Die 100 häufigsten Familiennamen im Korpus. Die Zahlen geben die Anzahl der Einträge mit gleicher Schreibweise auf der Telefonbuch-CD an.

Nach Filterung der Einträge repräsentiert der Korpus noch 31.63[45] Millionen Telefonanschlüsse. Die entnommen Teilkorpora der hundert häufigsten Namen (siehe Abbildung 14) entsprechen 4.17 Millionen Telefonanschlüssen. Die tausend häufigsten würden bereits 9.96 Millionen Anschlüssen repräsentieren. Bei der Annahme eines konstanten Verhältnisses von veröffentlichten Telefonanschlussdaten zur Bevölkerung besitzen 13,18 % der deutschen Bevölkerung einen der hundert häufigsten Familiennamen. Für die tausend häufigsten Namen ergibt sich bereits eine Bevölkerungsquote von 31,48 %.

5.3 Validierung der Qualität

Wie sich bereits in den vorherigen Kapiteln gezeigt hatte, war einiges an Aufwand nötig, um den Korpus von Einträgen mit Firmennamen und/oder Institutionen zu bereinigen. Um die Qualität der Filterung zu überprüfen wurden dem Korpus zufällig 1.000 Stichproben entnommen und von deutschsprachigen Sprechern anonym per WWW bewertet[46]. Den Befragten wurde jeweils eine Liste von 25 Einträgen vorgegeben. Sie mussten einschätzen, ob es sich um einen Familiennamen oder Firmennamen bzw. sonstige öffentliche Institutionen handeln könnte. Für schwer entscheidbare Fälle war es zusätzlich zulässig ,,Beides"

[45]Dieser Wert wurde durch Aufaddieren der Häufigkeiten der nicht gefilterten Anschlüsse errechnet.

[46]Eine genaue Anzahl an Versuchspersonen kann nicht angegeben werden, da die Umfrage anonym war und eine Mehrfachbewertung möglich war. Es wird vermutet, dass insgesamt ca. 15 Personen teilgenommen haben.

auszuwählen.

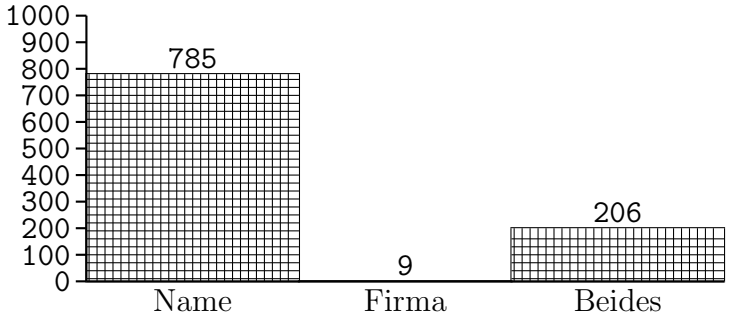

Abbildung 15: Anzahl der Zuordnungen in in die Kategorien Familienname, Firma und Beides

In Abbildung 15 sind die Ergebnisse der Umfrage zusammengefasst. Sie lassen vermuten, dass es sich bei einer Obergrenze von ca. einem Prozent der Korpusdaten eindeutig nicht um Familiennamen handelt. Die tatsächliche Quote von eindeutig falschen Namenstypen dürfte jedoch niedriger sein, da auch nach der Umfrage im Rahmen der folgenden Untersuchungen mehrere Datensätze als Name einer Institution, Abkürzung oder Ortsname identifiziert und entfernt wurden.

Zusätzlich zu dieser Kategorisierung wurde innerhalb der Umfrage nach dem vermuteten Sprachursprung des Namens gefragt. Zur Auswahl der vermuteten Sprachherkunft standen: Arabisch, Asiatisch, Afrikanisch, Deutsch, Englisch, Französisch, Griechisch, Italienisch, Osteuropäisch/Russisch, Skandinavisch, Spanisch, Türkisch und ,,nicht zuzuordnen".

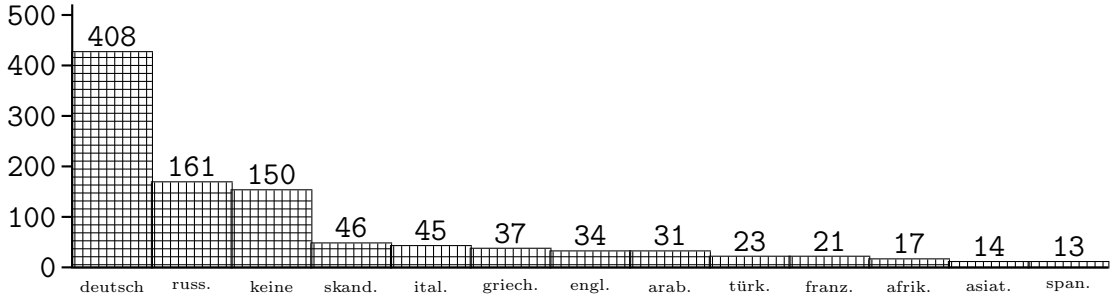

Abbildung 16: Umfrageergebnis zu vermuteten Quellsprachen der Familiennamen

Die Zuordnung in Abbildung 16 zeigt, dass osteuropäische und russische Familiennamen eine relativ hohe Verbreitung im deutschsprachigen Raum haben. Bei diesen Zahlen sollte jedoch bedacht werden, dass die tatsächliche Häufigkeit der Namen in der Bevölkerung um ein Vielfaches kleiner ist, da jede Schreibweise der Familiennamen nur einmal im Korpus enthalten ist.

Die relativ hohe Quote von nicht einem Sprachkontext zuordbaren Familiennamen liegt zum einen an der Ähnlichkeit einiger Sprachen, so dass eine eindeutige Zuordnung nicht möglich ist. Zum anderen war vermutlich keiner der Probanden mit allen zur Auswahl stehenden Sprachen vertraut. Des Weiteren sind im Korpus einige Doppelnamen enthalten, bei denen jeder Name jeweils einem anderem Sprachkontext zuzuordnen wären.

Andere Untersuchungen wie Vitale (1991) oder Black u. Llitjos (2001) zeigen, dass eine Berücksichtigung der Ursprungssprache von Eigennamen zu einer Verbesserung der Aussprache führt. Aus dieser Motivation heraus, wurde ein Versuch unternommen, eine Sprachkategorisierung mit der Hilfe von bestehender Software durchzuführen. Mittels Lingua::Identify, einem Perl-Modul für die Sprachidentifikation[47] von Texten wurden die Namen der Umfrage ebenfalls bewertet. Jedoch zeigte sich nach Sichtung der Ergebnisse und einem Abgleich mit den von den Probanden vorgenommenen Bewertungen, dass Lingua::Identify für die Bewertung von Familiennamen nicht geeignet ist.

Es wird davon ausgegangen, dass Lingua::Identify eher für die Klassifikation von längeren Texten geeignet ist. Eine Trigramstatistik, welche aus eindeutig einer Sprache zugeordneten Namen generiert wurde, dürfte bessere Ergebnisse bringen. In Vitale (1991) werden zusätzlich zu einer solchen Statistik Regeln für eine Eliminierung oder eindeutige Bestimmung von Sprachen verwendet. Da das aus der Umfrage gewonnene Material vom Umfang her statistisch nicht relevant sein dürfte, wurde kein Versuch unternommen, mit Hilfe von diesem eine Trigrammstatistik zu erstellen.

Einen ausschliesslich regelbasierten Ansatz für die Kategorisierung der Ursprungssprache schlägt Belhoula (1993) vor. Dort soll die Suche nach Zeichenketten mit variabler Länge eine Zuordnung der Sprache erlauben. Besonderheit ist hier ein Positionskontext, wie er auch in einigen der Regelsysteme für die Phonetische Suche verwendet wird. So sind z. B. Namen, die mit <Gio> anfangen, dem Italienischen zuzuordnen. Die Erstellung solcher Regeln in einem automatisierten Verfahren dürfte jedoch die Komplexität einer N-Gram-Statistik bei weitem übertreffen. Auch sind derzeit keine Forschungen darüber bekannt, ob zusätzliche Positionsangaben zu einer genaueren Kategorisierung verhelfen könnten.

[47] Dieses Modul benutzt, wie es auch im Artikel von Black u. Llitjos (2001) vorgeschlagen wurde, vorgefertigte Trigramstatistiken für die Klassifikation von Zeichenketten. Diese sind für die Sprachen und Dialekte Afrikaans, Bulgarisch, Bretonisch, Bosnisch, Welsh, Dänisch, Deutsch, Englisch, Esperanto, Spanisch, Finnisch, Fränzösisch, Friesisch, Irisch, Kroatisch, Ungarisch, Isländisch, Italienisch, Latein, Holländisch, Norwegisch, Polnisch, Portugiesisch, Albanisch, Schwedisch und Türkisch verfügbar. Leider ist über den Ursprung dieser Statistiken nichts bekannt. Insbesondere fehlen in der Dokumentation der Module Angaben über das Material, aus dem diese Statistiken erstellt wurden.

6 Vergleich der Verfahren für eine Phonetische Suche

Mehrere Untersuchungen über den Vergleich von Verfahren für die Phonetische Suche sind für andere Sprachkontexte, wie z. B. Englisch oder Dänisch, bekannt. Neben dem anderen Sprachkontext weichen aber auch andere Rahmenbedingungen dieser Untersuchungen, wie der verwendete Korpus oder die Art der Bewertung, zum Teil erheblich von dieser Arbeit ab. Im Folgenden soll ein Überblick über die bekannten Untersuchungen gegeben werden.

6.1 Bekannte Untersuchungen

Lait u. Randell (1993) vergleichen Soundex, Metaphone und zwei weitere Algorithmen für den sprachunabhängigen Vergleich von Zeichenketten. Eines davon, das ,,k-approximate String-Matching" wird nicht genauer beschrieben. Das Zweite, nach der Urheberin Guth-Algorithmus genannt, scheint eine stark vereinfachte Form der Levenshtein-Distanz zu sein. Zusätzlich wird am Ende des Artikels ein weiteres, neu entwickeltes Soundex-Derivat mit dem Namen Fonex[48] vorgestellt.

Der Aspekt einer datenbankbasierten Speicherung und Abfrage von Familiennamen durch die generierten Schlüssel wird bei Lait u. Randell (1993) nicht berücksichtigt, obwohl sowohl Soundex als auch Metaphone diese Anwendung nahelegen. Vielmehr wird angenommen, dass jeder Eingabename mit jedem anderen Namen verglichen werden muss.

Die Untersuchung basiert auf einem Korpus von äquivalenten englischen Familiennamen aus Park u. Park (1992). Diese werden zusätzlich um simulierte Tippfehler ergänzt, um das Verhalten der Algorithmen auf Robustheit bezüglich dieser zu testen. Am Rande werden Aspekte von Namenszusätzen besprochen. So sind z. B. <St. Clair> und <Sinclair> in der Aussprache sehr ähnlich.

Es wird eine Reihe von Kriterien aufgestellt, die ein Korpus für den Namensvergleich erfüllen sollte:

- Im Korpus sollten alle möglichen Anfangsbuchstaben in den Familiennamen vertreten sein. Dies wird damit begründet, dass Probleme einzelner Algorithmen, insbesondere die des unmodifizierten Soundex, ansonsten nicht in die Bewertung einfliessen.
- Die in dem Korpus enthaltenen Familiennamen sollten insbesondere in Bezug auf die Buchstaben einer Normalverteilung für Namen entsprechen. Dieses Kriterium dürfte dem Schwerpunkt der Arbeit bei Tippfehlern zugeordnet werden.
- Die Länge der Namen sollte einer repräsentativen Verteilung der Längen entsprechen.

[48] Da zum einen diese Variante über den Artikel hinaus keine Verbreitung gefunden hat und zum anderen kein Anlass für die besondere Tauglichkeit für das Deutsche gegeben war, wurde darauf verzichtet, dieses Verfahren zu implementieren und in den Vergleich mit aufzunehmen.

- Die Datenmenge sollte nicht zu klein gewählt werden, ansonsten könnten Geschwindigkeitsmessungen zu viele Nebenaspekte des verwendeten Betriebssystems, wie z. B. das Startverhalten des Programms, miteinbeziehen.

Pfeifer u. a. (1995) untersuchen vor allem die Kombination von Algorithmen für die Phonetische Suche. Zuerst werden Soundex, die Varianten von Phonix, Damerau-Levenshtein und zwei Verfahren namens Skeleton Key und Omission Key[49] miteinander verglichen.

Anschliessend werden Kombinationen der vorgestellten Verfahren mit Hilfe von Recall- und Precision-Diagrammen[50] beurteilt. Gesamtergebnis der Untersuchung ist, dass die Phonixvariante mit Ending-Sound in Kombination mit Digrammstatistik das beste Ranking erzeugt.

In Erikson (1997) handelt es sich um eine Untersuchung für das Schwedische. Diese Arbeit misst zum einen die Geschwindigkeit der untersuchten Algorithmen, aber auch die Fehlerrate. Für die Kategorisierung der Suchresultate werden dabei eng bemessene Kriterien verwendet. So ist ein positives Suchergebnis bei Erikson (1997) nur dann gegeben, wenn ein Name exakt gleich ausgesprochen werden kann. Es werden deshalb Probleme bei der akustischen Übertragung und sonstige Ähnlichkeiten bei der Aussprache nicht berücksichtigt.

Für diesen Aufsatz wurde ein Korpus von 5.327 unterschiedlichen schwedischen Namen verwendet. Weiterhin wurde für die Generierung von Suchanfragen eine nicht weiter spezifizierte Zahl von Namen aus dem Stockholmer Telefonbuch verwendet.

Bei Erikson (1997) werden einige Verfahren vorgestellt, die der Autor als ungeeignet einstuft. So wird berichtet, dass das Training von neuronalen Netzwerken wegen des nicht polynomiellen Laufzeitverhaltens zu langsam ist. Die schon erwähnten Skeleton-Keys und das Verfahren nach Guth sind ebenfalls nicht effektiv genug bzw. liefern zu viele, fehlerhafte Ergebnisse. Für die sprachunabhängige Verarbeitung wird die Verwendung von Digrammen, für die sprachabhängige Verarbeitung die eines gewichteten[51] Edit-Distance-Verfahrens empfohlen. Das vereinfachte Ergebnis dieser Arbeit lässt sich folgendermassen

[49] Hierbei handelt es sich um Verfahren für die Umgruppierungen und Sortierung der Buchstaben nach Konsonanten und Vokalen. Sie schnitten sowohl bei den Untersuchung von Pfeifer u. a. (1995), als auch bei Erikson (1997) besonders schlecht ab, so dass Sie im Rahmen dieser Arbeit keine weitere Berücksichtigung mehr finden.

[50] Recall/Precision wird in Raghavan u. a. (1989) beschrieben. Es handelt sich um ein Verfahren, welches die Güte eines Rankings bewertet und diese mit Hilfe von Diagrammen visualisiert.

[51] Für die Gewichtung wurde auf bestehende Äquivalenztabellen für schwedische Grapheme zurückgegriffen. Diese sind jedoch lediglich in schwedischer Sprache publiziert worden und konnten deshalb im Rahmen dieser Arbeit nicht berücksichtigt werden. Aufgrund der im Artikel gegebenen Beispiele wird ausserdem angenommen, dass das Schwedische weniger Probleme mit dem Alignment von Graphemen

formulieren: Je einfacher der Algorithmus, desto schneller ist er und desto fehlerhafter sind die Suchresultate.

In Patman u. Shaefer (2003) finden sich die gesammelten Argumentationen gegen den Einsatz von Soundex-basierten Verfahren. Dabei werden Probleme der Kodierung im englischen Sprachkontext angesprochen. Vor allem wird der ursprüngliche Soundex betrachtet und in ausgewählten Punkten mit einem eigenem Produkt[52] verglichen. Verbessere Varianten, wie z. B. Daitch-Mokotoff, Metaphone oder Phonix werden nicht ausreichend diskutiert.

6.2 Vorgehensweise

Auf Basis des extrahierten Korpus wurde eine Datenbank angelegt. Sie enthält eine einfache zweidimensionale Tabelle. In dieser Tabelle wird für jeden der getesteten Algorithmen eine Spalte für die Schlüssel angelegt. Anschliessend wurden mittels eines Skripts die Namen des Korpus' geladen. Für jeden Namen wurden die Schlüssel für die verschiedenen Algorithmen erzeugt und zusammen mit dem Namen in der Datenbank eingefügt. Anschliessend wurde für den Teilkorpus der 100 häufigsten Familiennamen für alle Algorithmen die Suchergebnisse gespeichert. Danach wurde die Güte der Suchergebnisse bewertet.

6.3 Die verwendete Softwareumgebung

Um aus einer einheitlichen Umgebung auf alle Algorithmen zuzugreifen, wurde die Programmiersprache Perl[53] verwendet. Sie zeichnet sich besonders durch ihre Fähigkeit bei der Verarbeitung von Daten und Zeichenketten aus.

Für Phonet[54], Phonix, Metaphone und ein Soundex waren Implementationen in Perl bereits verfügbar. Für Daitch-Mokotoff, Phonem und die ,,Kölner Phonetik" lagen keine verwendbaren Implementationen vor. Sie wurden entsprechend der Beschreibungen in den jeweiligen Artikeln als Perl-Module implementiert. Die Quelltexte dieser Module finden sich im Anhang dieser Arbeit. Zusätzlich wurde das vorhandene Soundex-Modul um Extended Soundex erweitert[55].

---

und Phonemen besitzt als das Deutsche. Ein Äquivalenztabelle für Grapheme dürfte im Deutschen um einiges schwieriger umzusetzen sein.

[52] Siehe hierzu auch die Beschreibung der IPA-basierten Verfahren in Kapitel 3.6.9.

[53] Für eine weitergehende Beschreibung von Perl sei hier auf Schwartz u. Christiansen (1997) verwiesen. Für einen praktischeren Ansatz für den in anderen Sprachen versierten Progammierer sei hier Christiansen u. Torkington (1998) erwähnt. Besonderes Gewicht auf die Objektorientierung legt Conway (2000).

[54] Da das Perl-Modul für Phonet in der verwendeten Konfiguration teilweise Schlüssel nicht korrekt erzeugte, wurde das ebenfalls im Quelltext verfügbare Kommandozeilentool von Phonet für die Generierung eingesetzt.

[55] Aufgrund ihrer trivialen Natur sind diese Änderungen nicht im Anhang zu finden.

Als Datenbank wurde MySQL ausgewählt, obwohl aufgrund der geringen Komplexität der Testdatenbank auch andere Datenbanken diese Aufgabe erfüllen könnten. Da SQL-Datenbanken jedoch seit einigen Jahren einen Standard darstellen, wurde MySQL als Vertreter dieser Datenbankkategorie verwendet. Obwohl MySQL eine Vielzahl von Erweiterungen des SQL-Standards bietet, wurde versucht, auf MySQL-spezifische Funktionalität zu verzichten. Lediglich beim Einfügen wurde aus Geschwindigkeitsgründen die Multiple-Insert-Funktionalität[56] verwendet.

Software	Version	Quelle
Perl	5.8.5	http://www.cpan.org/src/
MySQL	4.0.22	http://www.mysql.com
DBI	1.45	http://search.cpan.org/ timb/DBI-1.45/
DBD::mysql	2.9004	http://search.cpan.org/ rudy/DBD-mysql-2.9004/
Text::Metaphone	1.96	http://search.cpan.org/ mschwern/Text-Metaphone-1.96/
Text::DoubleMetaphone	0.07	http://search.cpan.org/author/MAURICE/Text-DoubleMetaphone-0.07/
Wait::Filtert	1.800	http://search.cpan.org/ ulpfr/WAIT-1.800/
Phonet	1.3.1	http://www.heise.de/ct/ftp/99/25/252/

Abbildung 17: Auflistung der eingesetzten Software und Bezugsquellen

Die Phonix-Implementierung in dem benutzten Perl-Modul WAIT::Filter scheint nicht in allen Punkten mit der Beschreibung in dem Artikel Gadd (1988) übereinzustimmen. Insbesondere die Generierung von mehreren Schlüssel wird nicht unterstützt. Aus den generierten Schlüsseln lässt sich ersehen, dass ,,Ending-Sounds"[57] nicht kodiert werden. Aufgrund der Ergebnisse von Pfeifer u. a. (1995) ist zu vermuten, dass mit dieser Funktionalität eine Verbesserung der Ergebnisse für diesen Algorithmus möglich wäre.

Da die Beschreibung der ,,Kölner Phonetik" noch einige Unklarheiten aufwies, wurde die Implementation dahingehend modifiziert, dass die in dem Artikel von Postel (1969) gegebenen Beispielschlüssel mit den generierten Schlüsseln übereinstimmen. Die in dem Artikel beschriebene Vorgehensweise für Namenszusätze wurde nicht berücksichtigt, da diese nicht Bestandteil des Korpus sind. Außerdem wurde das <ß> in die Klasse 8 und die Umlaute in die Liste der Vokale aufgenommen.

6.4 Allgemeine Statistik

Zu diesem Zeitpunkt der Untersuchung ist es bereits möglich allgemeinere Aussagen über die erzeugten Schlüssel vorzulegen. So haben einige der Verfahren eine feste Schlüssellänge durch Auffüllen von Leerpositionen und/oder Abschneiden von zu langen Schlüsseln. Bei

[56] Hierbei werden mehrere Datensätze auf einmal angelegt. Dies reduziert den Kommunikationsaufwand zwischen den Skripten und der Datenbank. Diese Möglichkeit der Beschleunigung existiert auch bei anderen Datenbanken, ist jedoch kein Standard. Die Verwendung von Multiple-Inserts erschien sinnvoll, da das Einfügen der gesamten Daten samt Generierung der Schlüssel aus den Namen auf dem verwendeten Rechner über zwei Stunden dauerte.

[57] siehe hierzu auch die Beschreibung des Algorithmus in Kapitel 3.6.4

den Verfahren mit variabler Schlüssellänge lässt sich aus der Abbildung 18 recht gut ersehen, welche Algorithmen Vokale eliminieren.

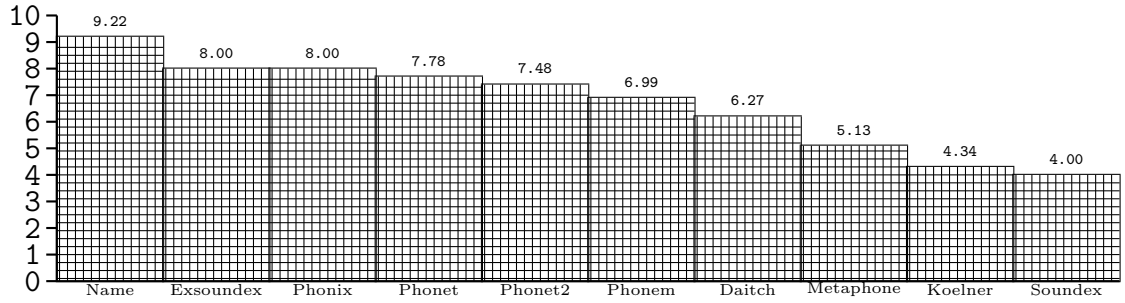

Abbildung 18: Durchschnittliche Länge der Schlüssel in Zeichen für die jeweiligen Schlüssel. Die durchschnittliche Länge des Ausgangseintrags ist mit ,,Name" beschriftet.

Ein Maß für die Komplexität der Algorithmen könnte die Anzahl der unterschiedlichen Schlüssel sein, auf die die Namen des Korpus abgebildet werden. Abbildung 19 zeigt, dass hier vor allem das ursprüngliche Soundex eine relativ niedrige Anzahl von verschiedenen Schlüsseln generiert und damit vermutlich zu ungenau arbeitet. Das andere Ende der Skala bietet die erste Variante von Phonet, welches wahrscheinlich zu genau arbeitet.

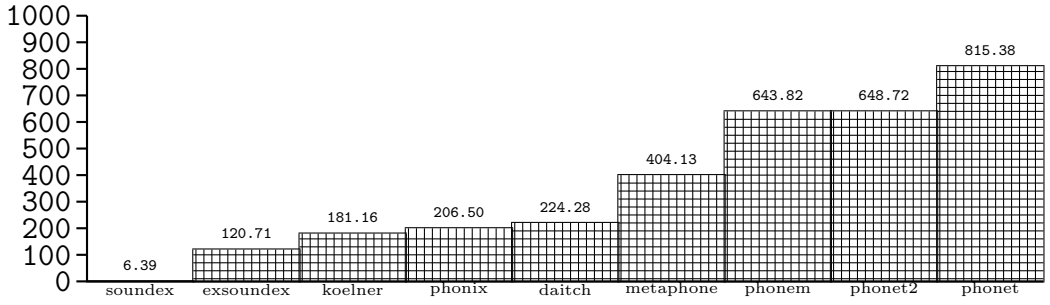

Abbildung 19: Anzahl der unterschiedlichen Schlüssel in tausend Stück, die die unterschiedlichen Verfahren generieren.

6.5 Kategorisierung der Suchergebnisse

Für die orthographisch hundert häufigsten Familiennamen wurde jeweils für jedes der Verfahren eine Suche ausgeführt. Die gesammelten Suchergebnisse für jeweils einen Namen wurden in Dateien geschrieben. Die dabei erzeugte Datenmenge war sehr groß. So ergab die Suche nach dem Namen <Müller> insgesamt 1.086 verschiedene Ergebnisse. Für <Schmidt> werden sogar 10.557 vermeintliche Varianten gefunden. Es soll hier exemplarisch für den Namen <Müller> eine Aufstellung über die Anzahl der gefunden Suchergebnisse gegeben werden.

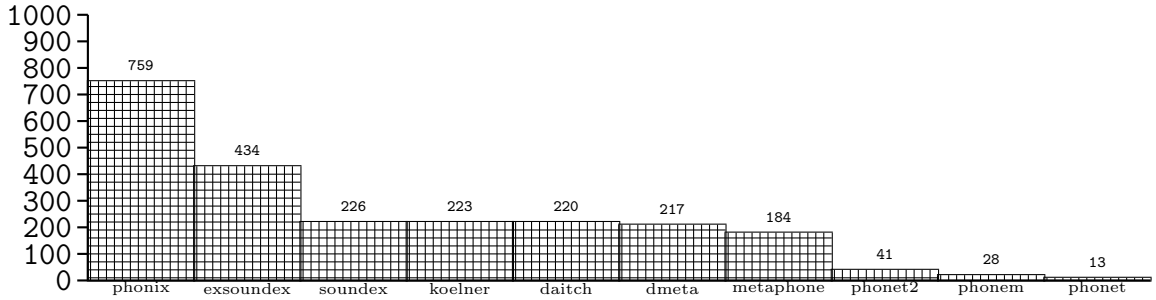

Abbildung 20: Anzahl der Suchergebnisse für <Müller>. Insgesamt wurden 1.098 unterschiedliche Ergebnisse gefunden.

Um zu vermitteln, wie die Verteilung der Suchergebnisse auf die Algorithmen ist, wird für die ersten hundert Namen der Mittelwert des prozentualen Anteils an der Anzahl der gelieferten Namen angegeben. Insbesondere Extended Soundex, ,,Kölner Phonetik" und Soundex liefern sehr viele Ergebnisse. Der durchschnittliche Anteil der richtigen Suchergebnisse für alle Verfahren zusammen liegt bei lediglich ca. 4,72 %. Wie Abbildung 21 zeigt, sind vor allem Soundex und Extended Soundex für die größte Anzahl der Suchergebnisse verantwortlich.

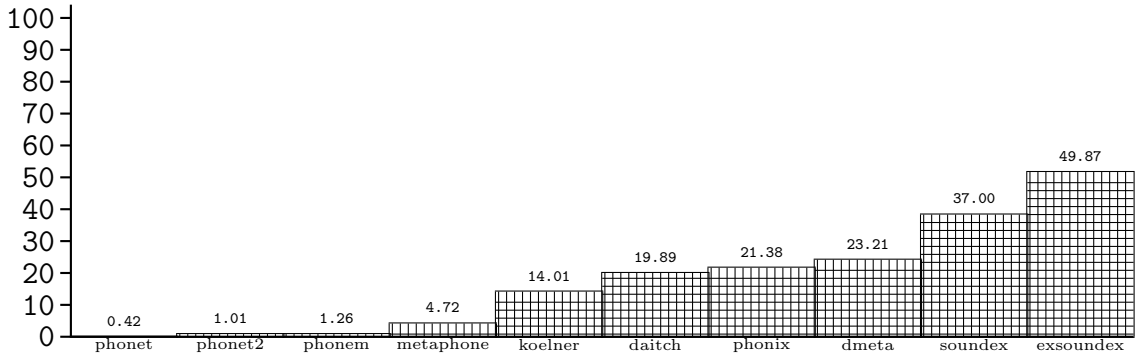

Abbildung 21: Mittlerer Anteil an insgesamt gelieferten Suchergebnissen in % für die fünfzig häufigsten Namen.

Die für einen Namen gefundenen Suchergebnisse wurden von Hand in drei Kategorien eingeordnet: Gut, Akzeptabel und Fehlerhaft. Gut entspricht einer möglichen Übereinstimmung in der Aussprache. Ein akzeptables Ergebnis liegt vor, wenn die Aussprache ähnlich ist. Ein Fehler wird gewertet, wenn die Aussprache des gesuchten Namens auf jeden Fall verschieden von dem gefundenen Namen ist.

Aufgrund der Datenmenge wurde für das Bewerten der Namensähnlichkeit ein kurzes Programm geschrieben, welches ein Suchfeld und zwei Tabellen anzeigt. In dem Suchfeld ist die Eingabe von regulären Ausdrücken[58] möglich. Die dem regulären Ausdruck entsprechenden Einträge werden in der ersten Tabelle angezeigt. Die weitere Tabelle enthält

[58]siehe Kapitel 8.1

die als nächstes zu bearbeitenden Einträge. Über drei Buttons kann die durch den regulären Ausdruck gegebene Auswahl einer der drei Kategorien zugeordnet werden. Eine Bildschirmaufnahme der Programmoberfläche findet sich in Abbildung 22.

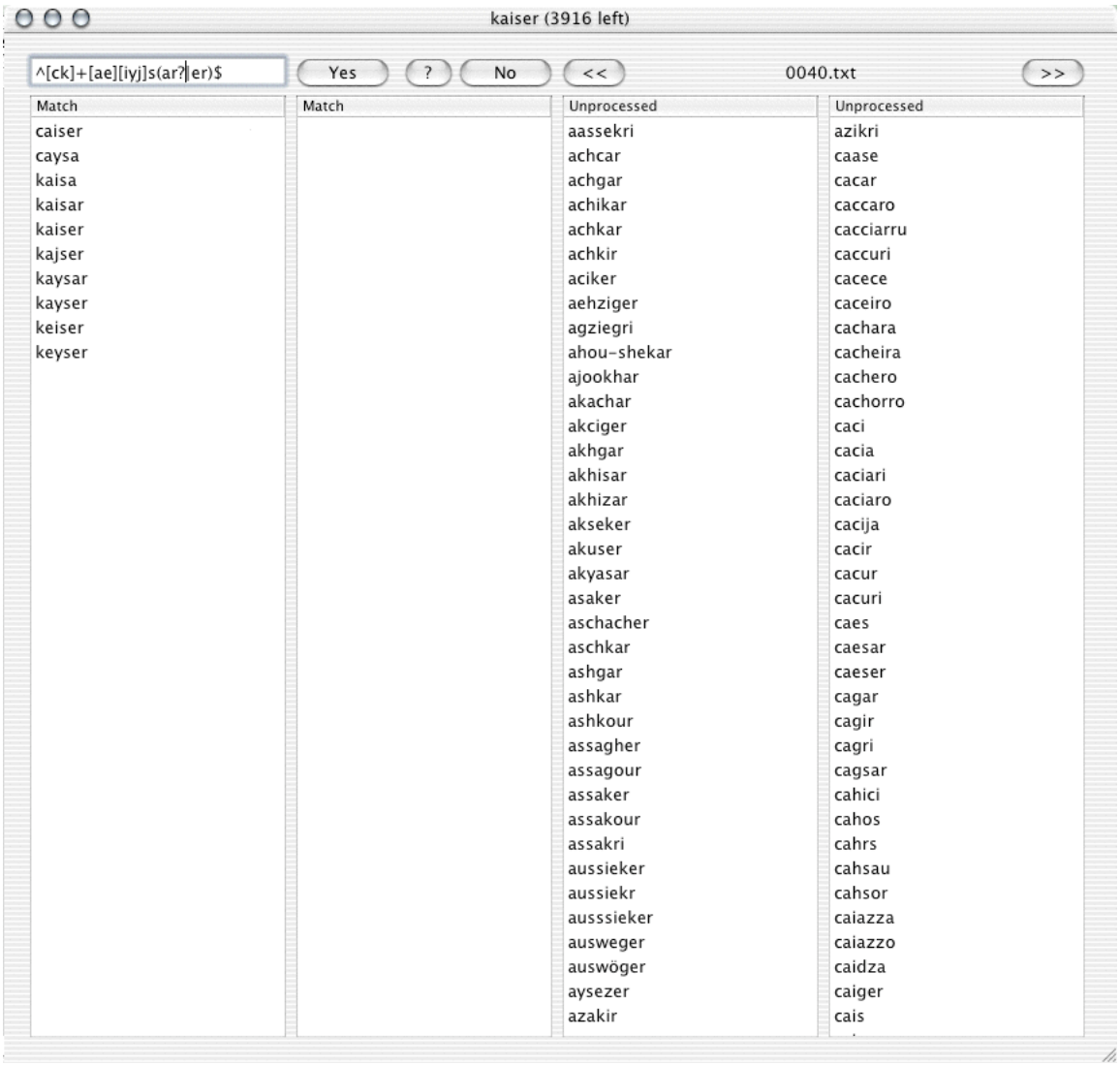

Abbildung 22: Bildschirmaufnahme des für das Kategorisieren benutzten Programmes. Der benutzte reguläre Ausdruck findet einige bekannte Schreibweisen des Namens <Kayser>.

Für die Bewertungen der Familiennamen auf Platz 50 - 100 wurden seltene Namen[59] entfernt. Dies war zum einen notwendig, da die Anzahl der gefundenen Suchresultate relativ hoch ist und die Kategorisierung ein zeitaufwendiger Prozess ist. Zum anderen ist die Güte der Suchergebnisse in der Praxis vor allem von der Qualität der Verfahren bei häufig auftretenden Familiennamen abhängig. Ausserdem hatte sich bei der Auswertung der fünfzig häufigsten Namen gezeigt, dass einige der gefundenen Namensvarianten im Kor-

[59] Als Schwellwert für die Seltenheit wurde eine Anzahl von weniger als 10 Einträgen im Telefonbuch angenommen. Eine Auszählung der Häufigkeiten im Korpus zeigte, dass lediglich 17 % der Familiennamen mehr als 10 Telefonanschlüsse besitzen. So konnte von einer Reduzierung der Datenmenge um den Faktor 1/5 ausgegangen werden.

pus wie z. B. <Hoffmnann> mit großer Wahrscheinlichkeit auf Tippfehler zurückzuführen sind. Diese Namensvarianten wurden als positiver Nebeneffekt durch diese Massnahme ebenfalls getilgt.

	Anzahl	sndex	exsndex	meta	dmeta	phonet	phonet2	phonix	daitch	phonem	koelner
müller	291322	X	X	X	X	X	X	X	X	X	X
miller	7159	X	X	X	X		X	X	X	X	X
mueller	1548	X	X	X	X	X	X	X	X	X	X
mühler	176	X	X	X	X	X	X	X	X	X	X
milla	137						X	X			
mühlherr	68	X	X		X			X		X	X
mülherr	37	X	X		X			X		X	X
myler	33	X	X	X	X	X	X	X	X	X	X
millar	17	X	X	X	X			X	X		X
myller	10	X	X	X	X	X	X	X	X	X	X
müllar	7	X	X	X	X			X	X		X
müler	6	X	X	X	X	X	X	X	X	X	X
muehler	4	X	X	X	X	X	X	X	X	X	X
mülller	2	X	X	X		X	X	X		X	X
myla	1					X	X	X			
nüller	1		X				X		X		X
müllerr	1	X	X	X	X			X	X	X	X
muehlherr	1	X	X		X			X		X	X
muela	1					X	X	X			
muellar	1	X	X	X	X			X	X		X
mueler	1	X	X	X	X	X	X	X	X	X	X
mülleer	1	X	X	X	X			X	X	X	X

Abbildung 23: Tabelle mit allen, als korrekt eingestuften Ergebnissen für die Suchanfrage <Müller>. Für jedes Verfahren wird angegeben, ob es die Schreibvariante findet. Die Zahl gibt die Häufigkeit im Telefonbuch an.

Für den Namen Müller soll hier gezeigt werden, wie die Kategorisierung vorgenommen wurde. So wurden von den insgesamt 1.098 gefundenen Entsprechungen lediglich die 23 in Abbildung 23 gezeigten als korrekt eingestuft. Die 37 Varianten aus Abbildung 24 wurden als akzeptabel bewertet und den Algorithmen somit weder positiv noch negativ angerechnet.

mella	mellah	mellar	meller	mellár	miehler	mila	milad
milah	milak	milar	millé	miloi	milor	moelle	moeller
moulla	moulller	muehle	muellwer	mula	mular	muler	mulla
muller	mulley	miloi	milor	moelle	moeller	moulla	moulller
muehle	muellwer	mular	muler	mulla	muller	mulley	mullér
mulè	mulé	myle	möhle	möhler	möler	mölle	möller
möllor	mülle	müllwer					

Abbildung 24: Tabelle mit allen, als akzeptabel eingestuften Ergebnissen für die Suchanfrage <Müller>.

6.6 Bewertung

Zwei mögliche Anwendungsfälle sollten bei der Bewertung berücksichtigt werden. So ist es auf jeden Fall notwendig, dass die Algorithmen möglichst alle Entsprechungen für eine Suchanfrage finden. Wenn die Suchresultate danach nicht mehr aufbereitet bzw. gefiltert werden, ist es ausserdem sinnvoll, wenn die Fehlerquote der Algorithmen gering ist. So wird kein Anwender eine Liste von mehreren tausend Namen durchgehen wollen, um 20 Schreibweisen zu finden, die der Suchanfrage in der Aussprache gleichen. Eine Aufbereitung könnte z. B. durch ein Ranking geschehen, das durch die Verfahren aus Kapitel 4 erzeugt werden könnte.

Die Auswertung der Anzahl der richtig zurückgelieferten Suchergebnisse für die ersten fünfzig Namen wurde in den Abbildungen 25 und 26 zusammengefasst. Für die fünfzig häufigsten Namen zeigt sich, dass hier die englischsprachigen Algorithmen nicht so schlecht abschneiden, wie aufgrund der Unterschiede in der Orthographie englischer und deutscher Namen zu vermuten wäre. So befindet sich Extended Soundex im Durchschnitt an der dritten Stelle, wenn es um die Anzahl der korrekten Ergebnisse geht. Lediglich Metaphone benutzt zu eng auf das Englische zugeschnittene Regeln und findet deshalb im Schnitt nur 46 % der korrekten Schreibvarianten.

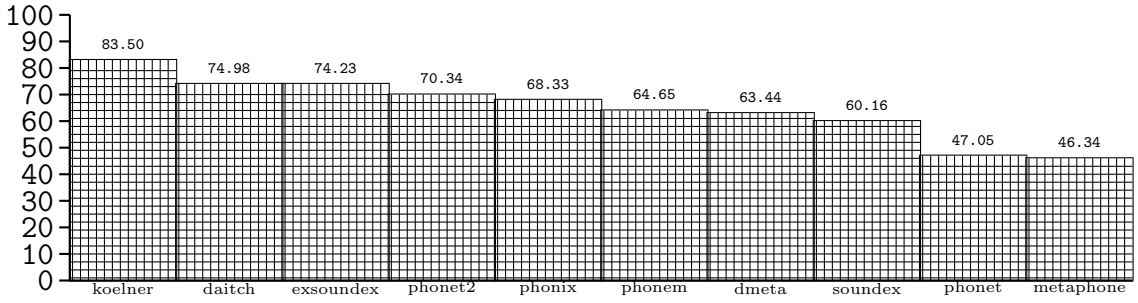

Abbildung 25: Gemittelter, prozentualer Anteil der als korrektes Ergebnis eingestuften Namen für die fünfzig häufigsten Namen.

Die Abbildung 26 zeigt die durchschnittliche Fehlerquote der Algorithmen. Es bestätigt die Vermutung, dass vor allem die Verfahren, die sämtliche Vokale eliminieren, zu viele unpassende Suchergebnisse erzeugen. So liefert Phonet nur wenige Suchergebnisse, von diesen sind jedoch im Durchschnitt lediglich 4,67 % fehlerhaft. Im Gegensatz dazu liefert die ,,Kölner Phonetik" die meisten Resultate, jedoch sind dort im Durchschnitt 87,96 % unbrauchbar. Somit ließe sich die ,,Kölner Phonetik" gut für eine Vorfilterung von großen Datenmengen verwenden, während Phonem mehr für eine schnelle Recherche geeignet ist, bei der es unerheblich ist, ob möglichst alle korrekten Suchergebnisse gefunden werden.

Für die hundert häufigsten Namen wurden diese Auswertungen wiederholt. Hier wurden, wie im vorherigen Kapitel beschrieben, seltene Namen eliminiert. Dies entspricht mehr

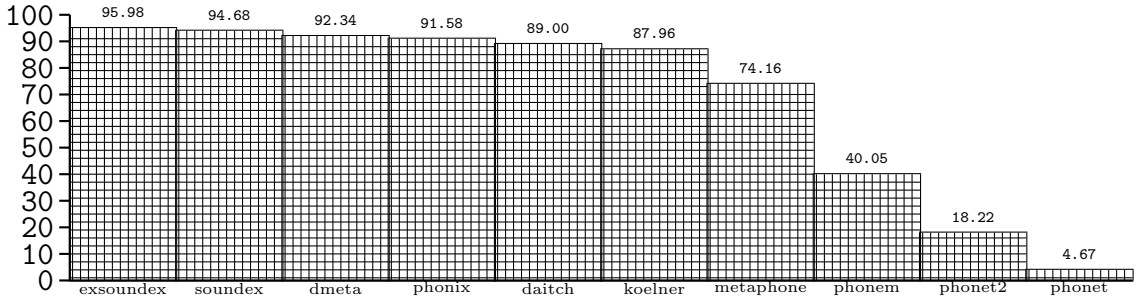

Abbildung 26: Fehlerquote der untersuchten Algorithmen in % bei den 50 häufigsten Namen.

dem üblichen Anwendungsszenario von z. B. Kundendatenbanken, da exotische Namen, die im Alltag selten vorkommen, damit nicht bei jeder Suche als potentielles Suchresultat berücksichtigt werden. In den folgenden Diagrammen ist also eine implizite Gewichtung nach Namenshäufigkeit enthalten.

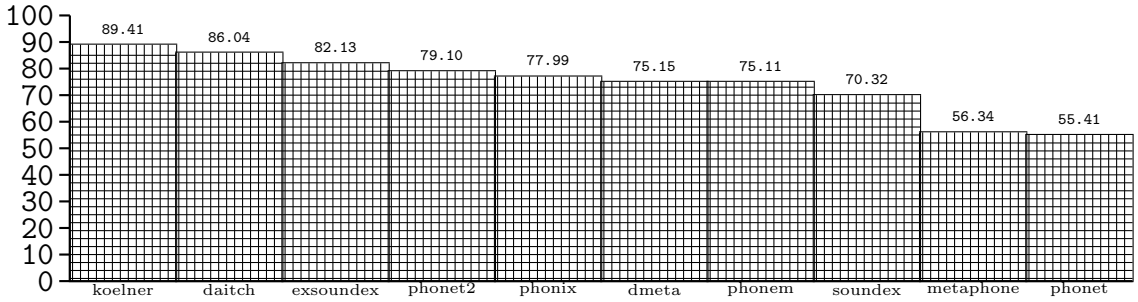

Abbildung 27: Gemittelter, prozentualer Anteil der als korrektes Ergebnis eingestuften Namen für die 100 häufigsten Namen.

In den Abbildungen 27 und 28 zeigt sich eine Übereinstimmung zu den Ergebnissen der 50 häufigsten Namen. Unter diesen Randbedingungen arbeiten die Algorithmen insgesamt besser. So ist die Anzahl der gefundenen korrekten Schreibvarianten höher und die Fehlerquote signifikant niedriger. Insgesamt gibt es keine nennenswerten Änderungen in der Güte der Algorithmen.

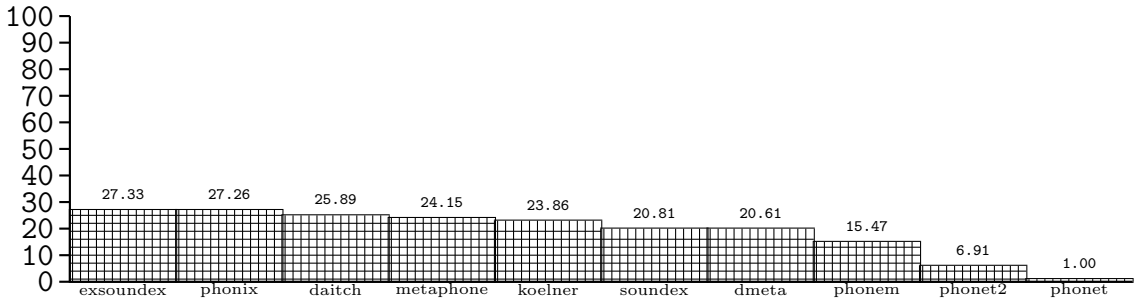

Abbildung 28: Fehlerquote der untersuchten Algorithmen in % bei den 100 häufigsten Namen

6.7 Auffälligkeiten bei den Resultaten der untersuchten Verfahren

Bei der Auswertung der Namen wurde für jeden Namen eine Kreuztabelle angelegt, wie sie schon in Abbildung 23 gezeigt wurde. In einer solchen Tabelle wird für jedes Verfahren angegeben, welche äquivalenten Schreibweisen gefunden wurden. Aus diesen Tabellen wurde versucht, für nicht gefundene Schreibvarianten herauszufinden, was die Ursache für das Fehlschlagen des jeweiligen Verfahrens ist. Im folgenden werden aus Gründen des Umfangs die Kreuztabellen nicht mehr komplett angegeben, sondern nur noch häufige Schreibvarianten gezeigt werden, die die Problemfälle der jeweiligen Algorithmen verdeutlichen.

Bereits in Abbildung 23 zeigte sich, dass fast alle Verfahren die häufigen Varianten von <Müller> finden. Lediglich die erste Variante von Phonet gruppiert die Vokale zu genau, so dass <Miller> nicht gefunden wird. So werden <i> und <ü>, die sich in diesem Fall lediglich durch die Rundung unterscheiden, verschieden kodiert. Die unschärfere Version Phonet2 hat jedoch keine Probleme mit dieser Kombination.

Bei den weniger häufigen Varianten von <Müller> zeigt sich, dass die meisten Verfahren nicht mit der Vokalisierung eine Wortfinalen <er> rechnen. Lediglich Phonix und Phonet2 können damit umgehen und finden die Variante <Mila>.

	Anzahl	sndex	exsndex	meta	dmeta	phonet	phonet2	phonix	daitch	phonem	koelner
schmidt	214647	X	X	X	X	X	X	X	X	X	X
schmitt	44301	X	X		X	X	X	X	X	X	X
schmid	40509	X	X		X	X	X	X	X	X	X
schmied	1925	X	X		X	X	X	X	X	X	X
schmiedt	296	X	X	X	X	X	X	X	X	X	X
schmit	220	X	X		X	X	X	X	X	X	X
schnitt	123	X	X				X	X	X		X
schmith	67	X	X		X	X	X	X	X	X	X
szmidt	25	X	X		X	X	X		X	X	X
schnith	19	X	X				X	X	X		X
szmyt	18	X	X		X		X		X	X	X

Abbildung 29: Tabelle mit den als korrekt bewerteten Ergebnissen für die Suchanfrage <Schmidt>.

Bei der Suche nach dem Namen <Schmidt> offenbarte sich ein Problem in Metaphone. So wird in der verwendeten Implementation <dt> anders kodiert, als <t>. Ob in diesem Fall ein <n> eine ähnliche Aussprache haben kann, wie ein <m> ist sicherlich strittig. Jedoch kodieren viele Verfahren Nasale in der gleichen Kategorie.

Neben der schon erwähnten Vokalisierung eines wortfinalen <er> zeigt sich bei dem Namen <Meyer>, welche Algorithmen die Graphemkombination von Diphthong und finalem <r> berücksichtigen. Auch die Verwendung von <y> und <j> als Vokal innerhalb eines Diphthongs wird von den englischsprachigen Algorithmen nicht vorgesehen. Genaueres zu diesem Thema findet sich in Kapitel 7.2.

	Anzahl	sndex	exsndex	meta	dmeta	phonet	phonet2	phonix	daitch	phonem	koelner
meyer	94783	X	X	X	X	X	X	X	X	X	X
meier	42050	X	X		X	X	X	X	X	X	X
maier	38450	X	X		X	X	X	X	X	X	X
mayer	36442	X	X	X	X	X	X	X	X	X	X
mayr	7103	X	X		X			X		X	X
mair	1911	X	X		X			X		X	X
majer	947							X	X	X	X
neyer	281		X				X		X		X
meyr	247	X	X		X			X		X	X
meya	181							X			

Abbildung 30: Tabelle mit den als korrekt bewerteten Ergebnissen für die Suchanfrage <Meyer>.

Beim Namen <Schultz> zeigt sich bei den Verfahren mit englischsprachiger Herkunft, dass bei <z> nicht die Möglichkeit einer Aussprache als Affrikate vorgesehen wird. Somit finden Soundex, Extended Soundex und Phonix die anderen graphemischen Repräsentationen der Affrikate nicht. In diesem Fall sind dies die Namen mit dem Bestandteil <tz> und <ts>.

	Anzahl	sndex	exsndex	meta	dmeta	phonet	phonet2	phonix	daitch	phonem	koelner
schulz	84422	X	X	X	X	X	X	X	X	X	X
schultz	11945					X	X		X	X	X
schulcz	22	X	X		X		X		X	X	X
schults	5					X	X		X	X	X

Abbildung 31: Tabelle mit den als korrekt bewerteten Ergebnissen für die Suchanfrage <Schultz>.

Beim Familiennamen <Groß> zeigen sich die schon in Kapitel 3.5.1 vermuteten Probleme mit Umlauten. Vor allem haben die englischsprachigen Algorithmen Probleme mit dem <ß>, welches nicht kodiert wird. Phonix schafft eine Erkennung nur, da es das letzte Zeichen nicht kodiert. Lediglich Phonix, ,,Kölner Phonetik", Phonet2 und Phonem können nahezu alle äquivalente Schreibweisen finden.

	Anzahl	sndex	exsndex	meta	dmeta	phonet	phonet2	phonix	daitch	phonem	koelner
groß	21606	X	X	X	X	X	X	X	X	X	X
gross	5921					X	X	X	X	X	X
gros	983					X	X	X	X	X	X
grohs	911					X	X	X	X	X	X
groos	766					X	X	X	X	X	X
kroos	305						X		X	X	X
kroß	220		X	X	X		X		X	X	X
krooß	132		X	X	X		X		X	X	X
grosz	97						X		X		X

Abbildung 32: Tabelle mit den als korrekt bewerteten Ergebnissen für die Suchanfrage <Groß>.

Das <th> im Namen <Thomas> bereitet insbesondere Metaphone Schwierigkeiten, die in Kapitel 3.6.3 schon angesprochen wurden. Zusätzlich zeigen sich Probleme mit dem

	Anzahl	sndex	exsndex	meta	dmeta	phonet	phonet2	phonix	daitch	phonem	koelner
thomas	18216	X	X	X	X	X	X	X	X	X	X
tomas	315	X	X		X	X	X	X	X	X	X
domas	66		X		X		X		X	X	X
thomaß	45					X	X	X	X	X	X

Abbildung 33: Tabelle mit den als korrekt bewerteten Ergebnissen für die Suchanfrage <Thomas>.

Anfangsbuchstaben in Soundex. Auch Phonet unterscheidet zwischen einem initialem <t> und einem <d>. Hierbei scheint es sich um einen Grenzfall zu handeln, bei dem von einigen Verfahren ein Unterschied angenommen wird, von anderen eine Äquivalenz.

	Anzahl	sndex	exsndex	meta	dmeta	phonet	phonet2	phonix	daitch	phonem	koelner
dietrich	19588	X	X	X	X	X	X	X	X	X	X
dittrich	8350	X	X	X	X	X	X	X	X	X	X
diedrich	2300	X	X	X	X	X	X	X	X	X	X
diederich	1580	X	X	X	X			X	X	X	X
dieterich	1281	X	X	X	X			X	X	X	X
diettrich	171	X	X	X	X	X	X	X	X	X	X
ditterich	145	X	X	X	X			X	X	X	X
ditrich	105	X	X	X	X	X	X	X	X	X	X
dietterich	37	X	X	X	X			X	X	X	X

Abbildung 34: Tabelle mit den als korrekt bewerteten Ergebnissen für die Suchanfrage <Dietrich>.

Abbildung 34 zeigt ein Beispiel, bei dem insbesondere Phonet schlecht abschneidet. Während die anderen Verfahren entweder Vokale nicht kodieren, oder wie Phonem das <e> speziell behandeln, kodiert Phonet dieses. Damit können Varianten wie <Dieterich> nicht gefunden werden, obwohl das Schwa mit hoher Wahrscheinlichkeit nicht oder nur reduziert ausgesprochen wird.

	Anzahl	sndex	exsndex	meta	dmeta	phonet	phonet2	phonix	daitch	phonem	koelner
fuchs	35006	X	X	X	X	X	X	X	X	X	X
fux	177				X	X	X	X			X
fuks	81				X	X	X	X	X		X
fuchß	73					X	X	X	X	X	X
fucks	41				X	X	X	X	X		X

Abbildung 35: Tabelle mit den als korrekt bewerteten Ergebnissen für die Suchanfrage <Fuchs>.

Eine Spezialität der deutschen Orthographie findet sich bei dem Namen <Fuchs>. Bei diesem wird das <chs> als [ks] realisiert. Selbst speziell für das Deutsche entwickelte Verfahren wie Phonem finden hier keine Entsprechungen für Varianten mit <k> oder <x>, die in diesem Fall allerdings sehr selten sind. Wie sich jedoch bei <Friedrichs> zeigt, geht diese Fähigkeit von Phonet auf Kosten der Namen, bei denen das <chs> frikativisch realisiert wird. Eine ausführlichere Besprechung der Aussprachevarianten des <ch> findet sich in Kapitel 7.4.

7 Orthographie und Aussprache im Deutschen

Wie im vorherigen Kapitel schon ersichtlich wurde, ist es für eine Phonetische Suche eine erfolgreiche Strategie, die Phonotaktik des Deutschen zu berücksichtigen. Deshalb sollen in diesem Abschnitt ausgewählte, für die Kodierung von Ähnlichkeiten relevante Grapheme des Deutschen und Laute, mit denen diese realisiert werden können, beschrieben werden. Im Mittelpunkt des Interesses stehen die Grapheme, die durch ihre Häufigkeit relevant sind, sowie Kombinationen von Graphemen, die mehrere Aussprachevarianten zulassen. Dabei sollen besonders Vokale, Diphthonge und Vokalisierungen betrachtet werden.

Bevor jedoch detailliertere Aspekte der Aussprache diskutiert werden, ist es zunächst erforderlich, mögliche Quellen für Ausspracheregeln zu identifizieren. Neben Einführungen in die Phonologie bieten insbesondere Aussprachewörterbücher einen guten Ausgangspunkt für Ausspracheregeln. Vereinzelte Dokumente aus dem Umfeld der Text-to-Speech-Projekte oder der automatischen Generation von Transkriptionen z. B. Wothke (1993) und Bosch u. Wolters (1997) beschreiben ebenfalls spezielle Teile der verwendeten Regeln. Darüber hinaus existiert eine Vielzahl von kleineren Aufsätzen wie z. B. Krech (2002), Adda-Decker u. Lamel (2000) und Adda-Decker u. a. (2000), die sich mit spezielleren Aspekten der Aussprache befassen.

7.1 Quellen für Aussprachevarianten

Siebs (1969) beschäftigt sich mit der von ihm eingeführten Hochlautung. Es handelt sich um eine besonders deutliche Aussprache, wie sie vor allem im Schauspiel benötigt wird. Neben Regeln zur korrekten Aussprache lassen sich aus diesem Werk vor allem aus der Vielzahl von Warnungen vor falscher Aussprache mögliche Aussprachevarianten ableiten.

In Mangold (2000, S. 65ff) wird ebenfalls eine Standardlautung vorgestellt, wie sie vor allem von Nachrichtensprechern benutzt wird. Hier lassen sich vor allem durch die zahlreichen Beispiele Ausspracheregeln ableiten. Auf Seite 64ff wird zusätzlich eine Umgangslautung eingeführt, die verschiedene Besonderheiten der Umgangssprache aufgreift. Auf den Seiten 69-106 wird für Graphemkombinationen eine Anleitung für die Aussprache gegeben. Eine Vielzahl von Aussprachevarianten mit Ursprung in anderen europäischen Sprachen wird dabei angegeben.

Wothke (1993) beschäftigt sich mit der automatischen Transkription von geschriebenem Text auf der Basis eines Morphemlexikons als Vorstufe für eine Spracherkennung. Dabei werden einzelne Ausspracheregeln diskutiert. Weiterhin wird deutlich gemacht, dass für die Transkription einiger Wörter morphologisches oder lexikalisches Wissen nötig ist. So ist es bei <Häuschen> notwendig zu wissen, dass <Haus> ein Morphem ist, um die Silbengrenze zu erkennen. Das Gegenbeispiel <täuschen> belegt diese Argumentation. Es

ist nicht auszuschließen, dass auch Namen existieren, bei denen ähnliches Wissen benötigt wird. Der in dem Artikel vorgestellte Ansatz benötigt ein Lexikon von mehreren tausend Morphemen. Eine Einsatzmöglichkeit für die Transkription von Familiennamen wäre somit fraglich, selbst wenn das beschriebene System verfügbar wäre. Insbesondere wird auf Seite 503 darauf hingewiesen, dass ,,von Eigennamen nicht erwartet werden kann, dass diese morphologisch segmentierbar sind".

Als weitere Quelle wurden die Regeln des schon in Kapitel 3.6.8 erwähnten Algorithmus' Phonet untersucht. Für jede Regel wurde die Häufigkeit der potentiellen Anwendung im Korpus untersucht. Für 155 der 850 Regeln wurde keine Anwendungsmöglichkeit gefunden. Die häufigsten Regeln werden im Folgenden im jeweiligen Kontext kommentiert.

7.2 Die Bedeutung von Vokalen im Deutschen

Vokalische Grapheme können im Deutschen im stärkeren Maße als Informationsträger betrachtet werden, als in anderen Sprachen. So ist die Graphem-nach-Phonem-Beziehung für Vokale eindeutiger, als z. B. im Englischen. Auch ist die Abbildung von Graphemen auf Vokalen weniger vom Kontext abhängig. Im Englischen ist es möglich, dass mehrere Grapheme als Kontext benötigt werden, um das entsprechende Phonem zu bestimmen. So wird bei <Pile> das <i> als [ɐ͡ɪ] gesprochen, während das <e> nicht gesprochen wird. Im Gegensatz dazu wird in <Pill> das <i> als [ɪ] realisiert. Gar keine Hilfe gibt der Kontext bei <Wind>, welches sowohl als [ʊɪnd] als auch [ʊɐ͡ɪnd] ausgesprochen werden kann.

Englischsprachige Algorithmen tendieren aus diesem Grund dazu, Vokale überhaupt nicht zu kodieren, da im Englischen in den meisten Fällen alleine aus den Konsonanten genügend Information für eine Auffindung von Äquivalenzen gezogen werden kann. Dies geht sogar so weit, dass noch nicht einmal die Position von Vokalen berücksichtigt wird. Die deutschsprachigen Algorithmen Phonet und PHONEM legen jedoch Wert auf die Beachtung der Vokale. So berichten Wilde u. Meyer (1988), die ebenfalls den Soundex-Algorithmus begutachten, dass <Suhrbier> und <Sauerbrei> den gleichen Soundex-Schlüssel erzeugen.

Relativieren kann man die Bedeutung der Vokale durch die guten Resultate der ,,Kölner Phonetik", die Vokale nicht beachtet. Jedoch ist hier die Fehlerquote ungleich höher, als bei Phonet oder PHONEM. Es gilt also, die Vokale soweit zu berücksichtigen, dass die Fehlerquote sich verringert, jedoch die Anzahl der korrekten Suchergebnisse gleich bleibt.

Gegen eine Kodierung von Vokalen sprechen neben Dialekten des Deutschen die Besonderheiten in Bezug auf das Schwa. Bei Dialekten weicht die Aussprache von Vokalen selbst im Vergleich zu der Umgangslautung des Hochdeutschen so sehr ab, dass keine vernünftige Prognose gegeben werden kann, welches Graphem als welcher Vokal realisiert

wird. In Bezug auf das Schwa sind gleich drei Phänomene zu vermerken: Zentralisierung, Reduktion und Epenthese.

So sind einige Vokalgrapheme sehr anfällig dafür, in der Aussprache in Richtung Schwa zentralisiert zu werden. Dies betrifft vor allem Vokale in unbetonten Silben und solche, die nur mit kurzer Dauer realisiert werden. Dadurch ist es möglich, dass auch Grapheme, die sich ansonsten eher unterscheiden, in gesprochener Sprache gleich realisiert werden.

Die Reduktion bezeichnet das extreme Verkürzen oder Weglassen von Lauten in der Aussprache. Sie ist meist an eine Zentralisierung gekoppelt und tritt besonders häufig bei schneller Sprache oder auch in der Umgangssprache auf. Alle Vokale können davon betroffen werde, jedoch ist das <e> im besonderen Maße betroffen. Dieses wird gerne in der unbetonten Silben z. B. in <el> oder <en> getilgt, da im Deutschen Nasale und Laterale silbisch werden können. Wie aus der Arbeit von Krech (1998)[60] hervorgeht, scheint der Kontext von vorausgehenden Vokalen, Liquiden oder stimmhafte Plosiven eine Tilgung des Schwas zu motivieren. Beispiel für eine solche Reduktion ist der Name <Wagener>, der bei schneller Aussprache als [vaɡnɐ] realisiert werden kann. Er ist somit identisch mit <Wagner>.

Eine Spezialität der deutschen Sprache stellt die Schwa-Epenthese dar. Hier werden in bestimmten Kontexten Schwas oder zentralisierte Laute wie z. B. das [ɐ] eingefügt. Welchen Einfluss dies auf die Aussprache von Namen hat, ist nur schwer zu bemessen, da maschinell nicht ohne größeren Aufwand zu ermitteln. Anhand des geringen Auftretens in den untersuchten hundert häufigsten Namen ist jedoch davon auszugehen, dass vor allem seltenere Namen davon betroffen sind. So ist es schon vorstellbar, dass z. B. <Schewaratz> bei schneller Aussprache wie <Schwarz> klingt. Anderenfalls ist es durchaus vorstellbar, dass bei <Schwarz> ein Schwa zwischen [ʃ] und [v] eingeschoben wird: [ʃəvaːts].

7.3 Ausgewählte Aussprachevarianten von Vokalen im Deutschen

Im Deutschen existieren eine Vielzahl von vokalischen Graphemen und Kombinationen von diesen. Schriftlich können diese auf verschiedene Weise repräsentiert werden. Sie werden regional unterschiedlich realisiert, jedoch lässt sich für jedes Graphem eine ungefähre Zuordnung zu einer Gruppe von Phonemen vornehmen. Kohler (1999) gibt eine Anzahl von 16 Phonemen und drei Diphthonge an, die im Deutschen verwendet werden.

Die Grapheme <aeiouöäüy> werden auf jeden Fall vokalisch gebildet. Für die Grapheme <jvw> besteht je nach Aussprache und Kontext die Möglichkeit einer Realisation als Frikativ, Approximant oder als Vollvokal. Siehe hierzu auch Mücke (1998). So kann das

[60] Hier wird gesprochene Sprache auf das Vorkommen von Reduktionen des Schwalautes analysiert und es werden Statistiken zu den jeweiligen Kontexten angegeben.

/v/ laut Siebs (1969) vokalisch oder stimmlos gebildet werden. Die vokalische Realisation entspricht dabei nicht mehr den Ausspracheregeln der Hochlautung.

Die üblicherweise vorgenommene Unterscheidung zwischen Kurz- und Langvokalen ist für die Kodierung von Ähnlichkeiten nicht besonders relevant. Eine Ausnahme bildet die vorhin erwähnte lautsprachliche Reduktion von Kurzvokalen. Sie sollte berücksichtigt werden, wenn Varianten ohne graphemische Repräsentation vorhanden sind. In dem oben schon erwähnten Beispiel <Schevartz> sollte das <e> demnach nicht kodiert werden. Von den vorgestellten Algorithmen, die Vokale kodieren, wird dies lediglich von PHONEM umgesetzt. Dieser eliminiert allerdings sämtliche <e>, auch wenn diese Silbenträger sind.

Potentiell ähnliche Phoneme erzeugen die Grapheme <e> und <ä>. Die entsprechenden gerundeten Laute werden durch ein <ö> oder ein <oe> umschrieben, aber von den meisten Verfahren in eine eigenständige Kategorie eingeordnet. Selbst Siebs (1969, S. 73) sieht hier eine Verwandschaft. Das <o> hat eine weite Spannbreite in der Realisierung und wird meist als [o] bis zum [ɔ]. Phonet2 geht von einer besonderen Nähe zum <u> aus, da es beide Vokale gruppiert. Allerdings könnte genauso gut auch eine Gruppierung mit dem <a> angenommen werden. Eine weitere Gruppe, welche auch in Phonet umgesetzt wurde, bilden die Grapheme <j>,<i>,<ü> bzw. <ue> und <y>. So kann laut Siebs (1969, S. 66) das <i> z. B. in <Hirsch> als [y] realisiert werden. Dies ist gleichzeitig die Regel für das <ü>.

Paarungen von vokalischen Phonemen stellen ein weiteres, großes Problemfeld dar. Bei ihnen ist bei unbekannten Namen nicht in allen Fällen klar, ob Kombinationen von zwei Vokalen als Diphthong, Umlaut oder als verlängerter Vokal ausgesprochen wird. So wird die Verdoppelung von Vokalen meist als Markierung für eine Verlängerung der Vokaldauer verwendet, wie z. B. in <Reetz>. In <Schaefer> wird das <ae> als alternative Schreibweise für den Umlaut <ä> benutzt. Im Namen <Maier> wird das <ai> als Diphthong realisiert.

Jedoch sind nicht alle Paarungen so eindeutig, wie die gegebenen Beispiele. Eine Vielzahl von Ambiguitäten tritt meist dann auf, wenn Einflüsse anderer Sprachen für eine andere Aussprache verantwortlich sind. Im Folgenden sollen einige Beispiele dafür gegeben werden.

- Bei dem Namen <Voigt> wird das <oi> als gelängter Vokal ausgesprochen. Andererseits wird das <oi> in <Stoiber> als [ɔɪ̯] realisiert. Vergleiche hierzu auch Siebs (1969, S. 72).

- Bei Namen wie Michael werden die Vokale der Grapheme <ae> einzeln realisiert. Dies steht im Wiederspruch zu der Verwendung des <ae> als alternative Schreibweise für das <ä>. Für die Zeichenfolge <oa> gibt es einen solche Ambiguität ebenfalls, wie die Namen <Noack> und <Joachim> beweisen.

- Das <ij> ist ein Beispiel für fremdsprachige Grapheme. Es wird vor allem in niederländischen Namen als Diphthong [ɐ͡ɪ] verwendet. In slavischen und romanischen Sprachen wird dagegen das <j> eher als Frikativ behandelt. Andere Vokalkombination mit <j>, wie <aj>, <ej> und <ij> sind ebenfalls typisch für Familiennamen. Obwohl auch Varianten von deutschen Namen wie z. B. <Majer> existieren, werden diese nicht in gängigen Aussprachewörterbüchern erwähnt. Das dem <j> ähnliche <y> hat in diesem Kontext eine weitere Verbreitung und dementsprechend wird z. B. in Mangold (2000) die Aussprache von Kombination mit anderen Vokalen erklärt.

- Vor allem in nordischen Städtenamen, z. B. in <Soest> wird das <e> als Längungsanzeige für den vorhergehenden Vokal benutzt. In häufigen Namen mit diesem Graphemen, wie z. B. <Schroeder>, wird die Vokalkombination von <o> und <e> eher als Umlaut verwendet.

- Bei <oo> überwiegt im Deutschen sicherlich die Verwendung als langes [o]. Die englische Aussprachevariante als langes [u] könnte jedoch bei Zusammenfassung von <o> und <u> in eine Kategorie berücksichtigt werden.

- Das <ie> wird im Deutschen für die Längung des Vokals verwendet. Im Französichen wird es laut Mangold (2000, S. 88) in Kombination mit einem <r> als Diphthong realisiert. Eine Sortierung aller Namen mit Teilzeichenkette <ie> und <ier> ergab lediglich Namen, bei denen keine französische Abstammung vermutet wurde. Somit dürfte es sich hier um einen seltenen Konfliktfall handeln.

In Abbildung 36 wird gezeigt, in welchem Umfang Zeichenketten mit zwei Vokalen im Korpus vorhanden sind. Hieraus lässt sich eine erste Abschätzung tätigen, wie wichtig die Berücksichtigung von Ambiguitäten potentiell werden könnte.

	a	e	i	o	u	y	j	ö	ä	ü
a	0.675 %	0.475 %	2.238 %	0.150 %	0.362 %	0.744 %	2.049 %	0.001 %	0.000 %	0.002 %
e	0.926 %	0.972 %	6.631 %	1.393 %	1.626 %	1.196 %	1.081 %	0.028 %	0.008 %	0.030 %
i	1.825 %	6.683 %	0.056 %	0.453 %	0.554 %	0.258 %	0.339 %	0.009 %	0.008 %	0.002 %
o	0.222 %	0.254 %	1.288 %	0.590 %	0.114 %	0.186 %	0.685 %	0.000 %	0.001 %	0.006 %
u	4.017 %	1.386 %	0.516 %	2.920 %	0.033 %	0.144 %	0.560 %	0.001 %	0.195 %	0.001 %
y	1.240 %	1.567 %	0.194 %	0.307 %	0.149 %	0.048 %	0.006 %	0.015 %	0.001 %	0.033 %
j	0.922 %	0.391 %	0.484 %	0.329 %	0.137 %	0.040 %	0.015 %	0.003 %	0.001 %	0.002 %
ö	0.002 %	0.003 %	0.005 %	0.001 %	0.002 %	0.012 %	0.052 %	0.003 %	0.000 %	0.001 %
ä	0.001 %	0.001 %	0.004 %	0.000 %	0.002 %	0.001 %	0.076 %	0.000 %	0.004 %	0.000 %
ü	0.003 %	0.002 %	0.001 %	0.004 %	0.001 %	0.063 %	0.072 %	0.000 %	0.000 %	0.004 %

Abbildung 36: Im Korpus gefundene Vokalkombinationen. Die Zeilen geben den ersten Buchstaben, die Spalten den Folgebuchstaben an.

Allein für den Diphthong [ɐɪ] gibt es somit eine Vielzahl von Graphemkombinationen, die diese Aussprache provozieren. Von denen mit zwei Buchstaben sind es <ai>, <aj>, <ay>, <ei>, <ej>, <ey> und mit den vorher diskutierten Einschränkungen <ij>. Darüber hinaus wurden im Korpus noch die Varianten <aij>,<iij> und <eij> gesichtet. Diese scheinen allerdings sehr selten zu sein.

Um zu überprüfen, welche Vokalkombinationen mit drei Buchstaben für das Deutsche relevant sind, wurde zusätzlich die Abbildung 37 angefertigt. Sie zeigt die häufigsten Kombinationen. Es lässt sich ableiten, dass sich eine Behandlung der Zeichenketten <ija>,<ije> und <eau> lohnen würde. Daraufhin wurde nach Namen mit <ija> und <ije> gesucht. Eine Sortierung der Suchergebnisse nach Häufigkeit ergab, dass bei diesen Vokalkombinationen mit Namen wie z. B. <Mijatovic> oder <Dimitrijevic> ein eher slawischer Namensursprung angenommen werden kann. Somit könnte eine Regel abgeleitet werden, die <ij> Frikativisch oder als langes [i] interpretiert, wenn ein weiterer Vokal folgt. Ansonsten ist eher ein niederländischer Kontext anzunehmen und es sollte eine Entsprechung für das [ɐɪ] verwendet werden.

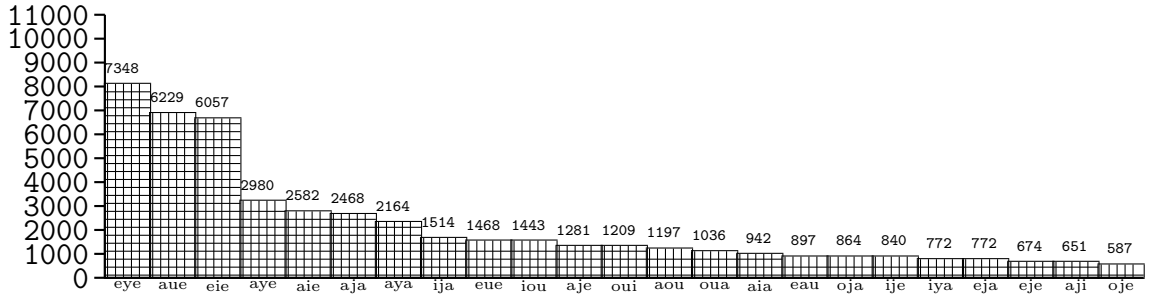

Abbildung 37: Anzahl der Einträge im Korpus mit den angegebenen Graphemketten. Einträge, die weniger als 500 mal vorkamen wurden ausgelassen.

Darüber hinaus gibt es in den Einträgen des Korpus einige Kombinationen von Vokalen mit Konsonanten, die eine vokalische Aussprache veranlassen. Häufig folgt auf einen Vokal ein <h>, welches eine Längung bewirkt. Falls auf das <h> ein weiterer Vokal folgt bewirkt dies, dass kein Diphthong entsteht, wie es ohne <h> der Fall ist. Bei deutlicher Aussprache wird in diesem Fall meist der glottale Frikativ gebildet. Ansonsten kann dieser aber auch getilgt werden. Andere Graphemkombinationen, wie ein wortfinales <ow> werden ebenfalls als Langvokal realisiert, z. B. in <Bülow>. Ebenso das aus dem Französischen stammende <aux> am Ende des Namens. Gegenbeispiele wie <Lauxmann> oder <Nowak> zeigen, dass diese Regel nur am Ende der Zeichenkette berücksichtigt werden sollte.

Das wortfinale <ais>, welches im Französichen als [ɛ] realisiert werden kann ist Bestandteil deutscher Namen wie z. B. <Mais> oder <Krais>. Hier ist maschinell nicht zu

unterscheiden, wie die Aussprache zu realisieren ist und es wird eine Präferenz für die deutsche Aussprache vorgeschlagen.

Gegen die Übernahme von Aussprachеregeln aus dem Englischen, wie z. B. <ew> -> [u] spricht die Verwendung in einer Vielzahl von häufigen deutschen Namen wie <Drews> oder <Ewald>. Eine englische Aussprache von <aw> ist hier häufiger, obwohl auch Namen wie <Stanislaw> keine eindeutige Zuordnung zulassen. Aussprachеregeln für das wortfinale <ice> oder das <ise> können verwendet werden, wenn kein weiterer Vokal vorangeht. Allerdings würden nur sehr seltene Namen von diesen Regeln profitieren. An anderen Positionen im Wort, z. B. bei <cicek> ist eher von einer slawischen Aussprache auszugehen. Bei vielen dieser Regeln aus anderen Sprachkreisen ist allerdings fraglich, ob sich aufgrund der Seltenheit der Aufwand lohnt. Selbst eindeutige Regeln, wie z. B. <oque> -> [ɔk] würden im Korpus lediglich 67-mal bei jeweils sehr seltenen Namen[61] angewendet.

7.4 Ausgewählte Aussprachevarianten von Konsonanten

Auch bei den Konsonanten gibt es bei deutschen Namen einige Eigentümlichkeiten. Es werden nicht nur doppelte Konsonanten für die Markierung einer Kürzung des vorangehenden Vokals verwendet, sie können auch eine Silbengrenze markieren. So wurden im Korpus äquivalente Schreibweisen wie <tt>,<dt>,<dd> und <td> gefunden. Letzteres ist allerdings sehr selten und tritt besonders bei Namen auf, die aus mehreren Komponenten bestehen, wie z. B. <Rottdorf> aber auch Namen wie <Ratdke> wurden gesichtet. Für die Bewertung der Aussprache ist dies jedoch meist unerheblich, da diese Varianten meist alle als [t] gebildet werden.

Der Einfachheit halber werden deshalb in den meisten Verfahren Obstruenten gleicher Artikulationsstelle zusammengefasst. Selbst unter den englischsprachigen Algorithmen findet sich dieses Verhalten, obwohl eine regelmässige Auslautverhärtung dort nicht bekannt ist. Im Deutschen lässt sich diese Vorgehensweise zusätzlich durch koartikulatorische Prozesse und regionale Variation begründen, wie Siebs (1969, S. 22) bestätigt: ,,In Mittel- und Oberdeutschland gibt es weithin keine stimmhaften Lenes.” Es scheint, dass es für Obstruenten eine zuverlässige Zuordnung von Schriftzeichen auf das phonologische Merkmal stimmhaft oder stimmlos gibt.

In Krech (1998) finden sich Bemerkungen zur /r/-Realisierung. Dieses tendiert demnach zu einer vokalisierten Aussprache, wobei eine besondere Klangfärbung erhalten bleibt. Laut Siebs (1969) ist in der Hochlautung das /r/ als Zungenspitzentrill oder als uvularer Laut zu bilden. Vokalisierungen sind demnach in der Hochlautung nicht erwünscht, sind aber in der Umgangssprache vor allem in der Silbenkoda der Regelfall. Dem Autor sind

[61] Im Mittel traten die Namen ca. 5 mal im Telefonbuch auf.

darüber hinaus auch Aussprachevarianten als [χ] bekannt.

Silbenfinal ergibt sich bei der häufigen Buchstabenkombination <er> (wie z. B. in <Meyer>) eine weite Spannbreite von Aussprachevarianten. Diese liegen zwischen den sehr genauen [eɾ] oder [eɾ], der diphthongischen Variante [əɐ̯] oder dem [ɐ]. Demnach ist es vorteilheift, die Schreibweisen mit <a> und <ar> in dieser Position[62] als äquivalent zu betrachten.

Im starken Maße vom Kontext abhängig ist die Realisation des <ch>. Viele der Algorithmen haben hier besondere Problem, wie sich aus Abbildung 35 ersehen lässt. Eine Aufstellung von Aussprachevarianten in verschiedenen Kontexten findet sich bei Wothke (1993, S. 489). Eine Zusammenfassung dieser Regeln wird in Abbildung 38 gegeben. Es bleibt anzumerken, dass einige der Regeln bei Namen nicht ausreichen, z. B. bei der Behandlung des <chs> bei geläufigen Namen wie <Friedrichs> oder <Heinrichs>. Auch bei Siebs (1969, S. 100ff) wird die Aussprache als [k], [ç], [χ] beschrieben, wobei in der Praxis besonders im Rheinland auch eine Realisation als [ʃ] häufiger vorkommt.

Kontext	Aussprache
Am Anfang des Wortes, wenn <l> oder <r> folgt.	[k]
Am Anfang des Wortes, wenn kein <l> oder <r> folgt.	[ʃ]
Wenn ein <s> vorhergeht	[ʃ]
Wenn ein <sl> folgt, oder ein <s>in Kombination mit verschieden Suffixen	[k]
nach den Vokalen <a>, <o> oder <u>	[χ]
in allen anderen Fällen	[ç]

Abbildung 38: Zusammenfassung der Aussprache Varianten des <ch> nach Wothke (1993, S. 489).

Gerade die Graphemkombination <chs> lässt vermuten, dass eine eindeutige Entscheidung für eine Aussprachevariante nicht in allen Fällen gefällt werden kann. Wothke (1993) nennt hier das Beispiel <Wachstube>, welches sowohl als [ʊaxʃtubə] als auch für [ʊakstubə] ausgesprochen werden kann. Hier kann lediglich auf semantischer Ebene entschieden werden, welche Aussprache gewünscht ist. Da diese bei Namen fehlt, ist es hier im Zweifelsfall erforderlich, die genaue Aussprache zu kennen. Bei einer Suche nach Namen mit Bestandteil <chs> wurden insgesamt 3.040 Namen gefunden. Obwohl diese Anzahl auf den ersten Blick relativ gering ist, hat sie trotzdem Bedeutung, denn unter den tausend häufigsten sind es bereits 8 %. Paare wie Höchstetter mit vermuteter Aussprache [høçʃtɛtɐ] und Höchster [høkstɐ] lassen vermuten, dass es hier keine einfachen Gesetzmässigkeiten geben wird.

Obwohl laut Hochlautung in Siebs (1969) das <pf> nicht als [f] gesprochen werden

[62]Das Problem bei der Verwendung dieser Erkenntnis liegt darin, dass keines der Verfahren für die Phonetische Suche eine Silbentrennung vornimmt. Somit kann zwar in einigen Fällen durch den Vokalkontext die Position bestimmt werden, in weniger eindeutigen Fällen wäre jedoch eine Silbentrennung nötig, um zwischen dem unproblematischen Silbenonset und der variantenreichen Silbenkoda zu unterscheiden.

soll, ist dies in der Umgangssprache üblich. Demnach bietet es sich an <pf>,<ph>,<f>, <v> und mit Einschränkungen[63] das <w> als ähnlich zu betrachten.

Für die Affrikate [t͡s] existieren verschiedene Schreibweisen, wie z. B. <z> und <tz>, <ts> und <ds>. Auch die anderen, schon erwähnten, aus mehreren Zeichen bestehenden Varianten des [t] können als Bestandteil einer Affrikate verwendet werden und wurden im Korpus gesichtet. Aber auch ein einfaches <s> wird in bestimmten Kontexten als Affrikate gesprochen, z. B. in <Schulls>.

Ein weiteres interessantes Graphem stellt das <g> dar. Es wird im Deutschen normalerweise als stimmhafter, velarer Plosiv umgesetzt. Da auch dieser Laut der Auslautverhärtung unterliegt, kann er in der Silbenkoda stimmlos werden.[64]. Eine weitere Variante ist hier die Realisation als [ç], die fast immer die Kombination mit einem <i> verlangt. In anderen europäischen Sprachen existieren darüber hinaus noch Aussprachevarianten als [ʒ] oder [dʒ], so dass vor allem im Wortanfang eine Ähnlichkeit mit dem <j> geprüft werden sollte.

Fraglich ist, ob eine Unterscheidung zwischen [ʃ] und [s] für Zwecke einer Phonetischen Suche sinnvoll ist. So können die Graphemketten <st> und <sp> mit beiden Lauten generiert werden, wenn sie silbeninitial sind. Selbst nach Hochlautung gibt es laut Siebs (1969) keine regelhafte Aussprache für Worte mit diesen Anfangsgraphemen. Hier wird vor allem die Herkunft des Wortes als Kriterium für die Aussprache des <st> oder <sp> angegeben. Gleichzeitig werden aber mehrere Ausnahmen genannt.

Eine Nichtberücksichtigung von Ausspracheregeln mit [ʃ] in dem beschriebenen Kontext würde allerdings wenige Probleme bereiten, da kaum Namen mit alternativen Schreibweisen wie <scht> oder <schd> im Korpus vorhanden sind.[65]

8 Ansätze für eine Verbesserung der Verfahren für die Phonetischen Suche

Für viele der vorgestellten Verfahren für die Phonetische Suche macht es Sinn, die zurückgelieferten Suchergebnisse erst einmal als fehlerbehaftete Vorauswahl zu betrachten. Dies nutzt die Geschwindigkeitsvorteile des verwendeten Hashingprinzips aus, um die Anzahl der Datensätze auf eine um Größenordnungen kleinere Vorauswahl zu beschränken. Anschliessend können diese Daten auch mit aufwendigeren Mitteln aufbereitet werden. So

[63] Das <w> wird im Gegensatz zu den anderen genannten Graphemen meist stimmhaft realisiert. Es tendiert zu einer Aussprache als Approximant oder sogar als [ʊ].

[64] In Mangold (2000, S. 84) werden Hinweise auf Kontexte gegeben. Vor allem die Stimmhaftigkeit des Folgelautes scheint die Auslautverhärtung zu steuern.

[65] Es existieren jedoch noch weitere Schreibweisen, z. B. <sz>,<cz>,<sc>,<sh> bei denen ohne Wissen über den Sprachursprung nicht klar ist, ob Sie als [ʃ], [ç], [s] oder [tç] ausgesprochen werden. Diese sollten ebenfalls entsprechend untersucht werden.

ist es nun auch vertretbar, komplexere Vergleichsoperationen durchzuführen, um falsche Suchergebnisse zu filtern oder um eine Reihenfolge zu generieren. Bei Kukich (1992) wird diese Vorgehensweise auch als Partitionierung der Datenbank bezeichnet. Partitionierungsmerkmal ist somit der Schlüssel der jeweiligen Verfahren für die Phonetische Suche.

Als Optimum hierfür dürfte ein gewichtetes Edit-Distance-Verfahren gelten, wie es unter anderem in Kapitel 4.4 vorgeschlagen wurde. Jedoch ist der dafür aufzuwendende Forschungsaufwand für deutschen Namen recht groß, da Probleme mit dem Graphem-Phonem-Alignment gehandhabt werden müssen. Bei genauer Betrachtung wäre eine Art automatische Transkription notwendig, wie sie ein Graphem-nach-Phonem-Konverter erledigt. Aufgrund der Fehlertoleranz von Algorithmen wie z. B. Levenshtein-Damerau wäre hier ein gewisser Fehlergrad durchaus zu vertreten, so dass die Anforderungen an die Genauigkeit der Transkription nicht so hoch wären, wie z. B. in TTS[66]-Systemen. Auch müsste die Berücksichtigung von Akzent und Intonation gar nicht oder nur mit sehr einfachen Modellen erfolgen. Dennoch wurde im Rahmen dieser Arbeit darauf verzichtet, ein entsprechendes Modell zu entwickeln.

Anstatt dessen sollen drei unaufwendig zu errechnende Merkmale vorgestellt werden, die eine Filterung der Suchergebnisse erlauben. Ziel ist es dabei, möglichst viele falsche und möglichst wenige richtige Treffer zu eliminieren. Untersucht werden Silbenanzahl, wortinitiale Grapheme und besondere vokalische Information.

Um das dafür verwendete Hilfsmittel vorzustellen, soll eine kurze Einführung in die regulären Ausdrücke gegeben werden. Diese werden ebenfalls in mehreren der selbst erstellten Implementationen von Algorithmen verwendet. Sie stellen eine Möglichkeit für die Beschreibung von Zeichenketten dar.

8.1 Reguläre Ausdrücke

Reguläre Ausdrücke[67] stellen ein System für das Beschreiben von Zeichenketten dar. Sie können für die Erkennung von Mustern und Kontexten verwendet werden. Weitergehende Beschreibungen der verschiedenen Formen finden sich in Friedl (2002). Eine formale Beschreibung und die Anwendbarkeit auf ausgewählte Aspekte der Sprachverarbeitung findet sich in Karttunen u. a. (1996).

Um einen Eindruck für die Syntax zu vermitteln, sollen hier ein paar Beispiele für reguläre Ausdrücke gegeben werden. So beschreibt Ausdruck /[aeoiu]{2}/ eine Abfolge von zwei Zeichen der in den eckigen Klammern angegebenen Vokalen. Der Ausdruck

```
/k[ae][ijy]h?s+(ar?|er)/
```

[66] Text-to-Speech
[67] englisch: Regular Expressions

beschreibt mögliche Schreibvarianten des Namens <Kayser>. Dabei sorgt das Fragezeichen für ein optionales Erscheinen des vorhergehenden Zeichens. Die runden Klammern gruppieren Unterausdrücke. Das | steht für die Entweder-Oder-Alternative. So steht am Ende von passenden Zeichenketten entweder ein <a> oder <ar> oder ein <er>.

Reguläre Ausdrücke eignen sich hervorragend für die Implementierung von phonologischen Regeln, da die Funktion beider recht ähnlich ist. Leider haben sich historisch bedingt mehrere Typen von regulären Ausdrücken entwickelt. Diese sind in der Mächtigkeit größtenteils äquivalent, unterscheiden sich jedoch in der Syntax. In den folgenden Verfahren werden die regulären Ausdrücke im Stil von Perl verwendet. Zum einen ist die Beispiel-Implementation in dieser Sprache verfasst, zum anderen besitzt Perl eine hervorragende Integration von regulären Ausdrücken. Für andere Programmiersprachen existiert über die PCRE Library[68] die Möglichkeit, diese Ausdrücke direkt zu übernehmen.

8.2 Silbenanzahl

Einige der Verfahren, insbesondere die mit fest kodierter Schlüssellänge, liefern für kurze Anfragen viele falsche Ergebnisse. Wie sich nach kurzer Analyse zeigte ist ein Teil dieser Suchergebnisse offensichtlich zu lang. Erste Überlegungen für ein Kriterium zur Filterung könnte somit ein Vergleich der Zeichenkettenlänge sein. So berichtet Kukich (1992) für das Englische, dass sich ähnlich ausgesprochene Wörter in der Länge der orthographischen Repräsentation meist unwesentlich unterscheiden. Da sich die Ähnlichkeit jedoch an der Aussprache orientieren sollte, wird im Folgenden versucht, die Anzahl der Silben zu bestimmen.

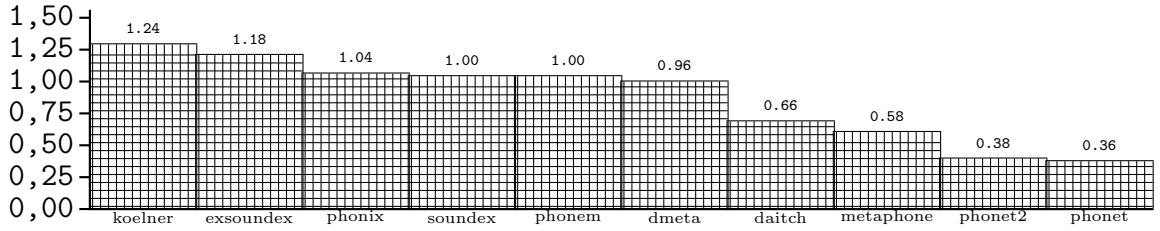

Abbildung 39: Mittlere Anzahl von fehlerhaften Filterungen bei den 50 häufigsten Namen bei Annahme von Gleichheit der Silbenanzahl.

Von den aus der Literatur bekannten Verfahren für eine Silbentrennung bot sich aufgrund der Einfachheit das Sonoritätsprinzip an. Ein entsprechender Algorithmus, ähnlich dem in Spencer (1996, S. 83ff) beschriebenen, wurde implementiert und ist im Anhang zu finden. Eine detailliertere Übersicht über die für die Silbentrennung nach Sonoritätsprinzip relevante Forschung gibt Niklfeld (1995). Komplexere Verfahren für eine Silbentrennung, wie z. B. clusterbasierte Ansätze in Müller (2000b) oder den schon in Kapitel 3.6.9

[68] siehe hierzu http://www.pcre.org

erwähnten Artikel von Belhoula (1993), wurden nicht weiter verfolgt, da für das Zählen der Silbenanzahl das Sonoritätsprinzip auszureichen scheint.

Bei der Implementierung stellten sich jedoch schnell erste Fragen. Zum einen bereitet wiederum die Phonotaktik Probleme, zum anderen ist die Vergabe der Sonoritätswerte fehleranfällig. Ersteres Problem wurde dadurch gelöst, dass bestimmte Zeichenkombinationen wie z. B. <ch> und <sch> zusammengefasst wurden. Dadurch lassen sich jedoch Wörter wie z. B. <Verkehrschaos> nicht mehr an der richtigen Position trennen. Dies ist zwar unschön, aber für den Anwendungszweck ausreichend, da lediglich die Anzahl der Silben gezählt werden muss. Aus diesem Grund wurde auch auf eine Maximierung des Onsets verzichtet.

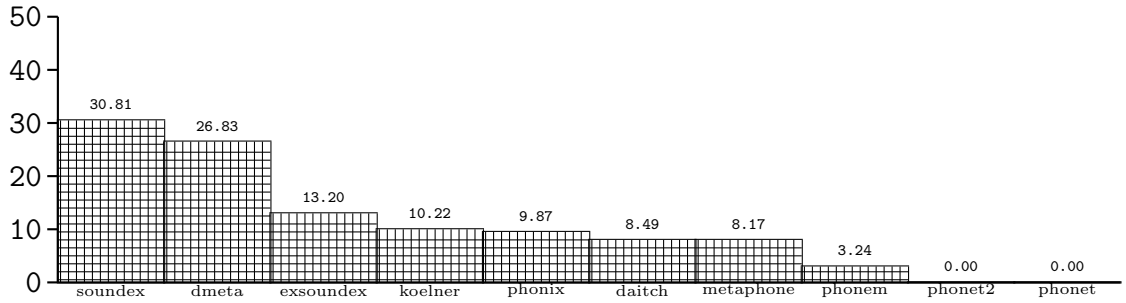

Abbildung 40: Prozentualer Anteil der durch Überprüfung der Silbenanzahl gefilterten Suchergebnisse bei den 50 häufigsten Namen. Insbesondere Soundex und Double Metaphone profitieren sehr von dieser Maßnahme.

Die zweite Problematik zeigt ein prinzipielles Problem des Sonoritätsprinzips, wenn versucht wird, es auf Grapheme anzuwenden. Namen wie <Michael> oder <Gabriel> besitzen Vokalkombinationen, die nicht, wie sonst üblich, in einer Silbe zusammengefasst werden, sondern beide einen Silbenkern darstellen. Bei Namen wie diesen sind Fehler beim Zählen der Silbenanzahl zu erwarten.

Aus diesem Grund wurden für den Vergleich der Silbenanzahl zwei Methoden gegenübergestellt. Zunächst wurde untersucht, wie die Anzahl der Fehler zu bewerten ist, wenn die Silbenanzahl gleich ist. In Abbildumg 39 findet sich die durchschnittliche Anzahl an fehlerhaften Filterungen in den 50 häufigsten Namen mit diesen Rahmenbedingungen. Da hier relativ viele Fehler auftreten, wurde in einem zweiten Schritt eine Toleranz von einer Silbe erlaubt. Mit dieser Toleranz wurde unter den 50 und 100 häufigsten Namen kein als korrekt eingestuftes Suchergebnis gelöscht. Die Wirkung der Filterung auf nicht korrekte Suchresultate wird in Abbildung 40 gezeigt. Die Wirkung der Filterung ist hier etwa um den Faktor zwei schwächer als bei der Annahme der Gleichheit der Silbenanzahl, jedoch sind keine negativen Auswirkungen auf die Anzahl der richtigen Suchresultate zu bemerken.

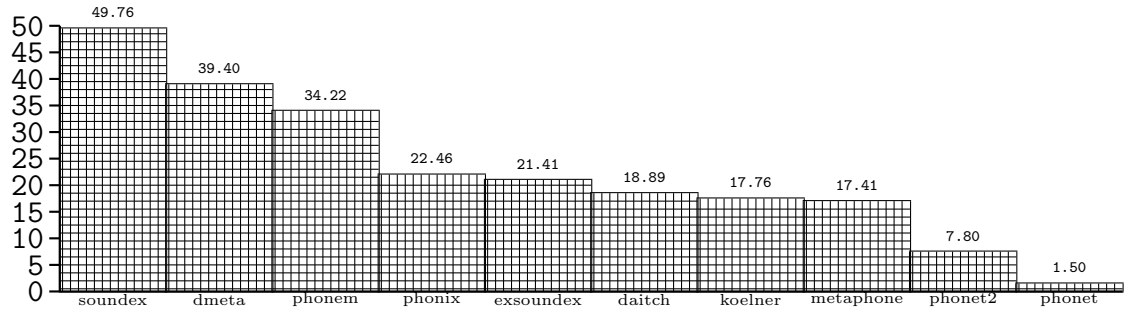

Abbildung 41: Prozentualer Anteil der durch Überprüfung der Länge gefilterten Suchergebnisse bei den 50 häufigsten Namen.

8.3 Länge der Zeichenkette

Als weiteres Merkmal für die Übereinstimmung von Zeichenketten in Bezug auf Ihre Aussprache kann im Deutschen auch die Länge der Zeichenkette benutzt werden. Sie ist einfach zu berechnen und scheint auf die Suchresultate ähnlich zu wirken, wie das Zählen von Silben. Es wurde für die Abbildungen 41 und 42 die Länge der Suchzeichenkette als Filterkriterium verwendet. Die Länge der Suchresultate durfte dabei um ein Drittel von der Länge der Suchanfrage abweichen.

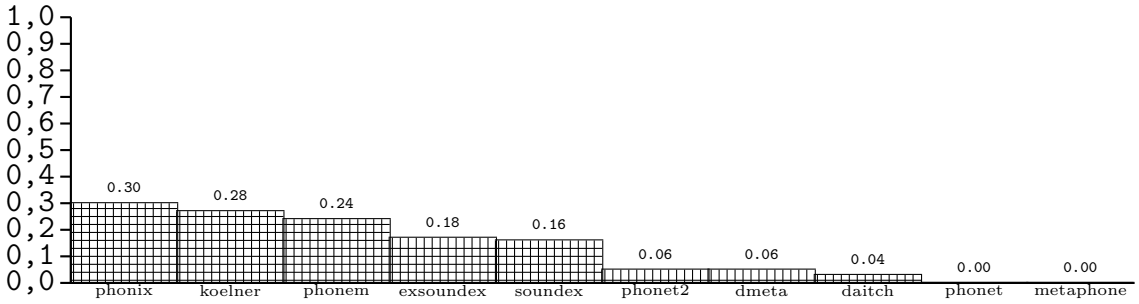

Abbildung 42: Durchschnittliche Anzahl der durch Überprüfung der Länge fälschlicherweise gefilterten Suchergebnisse bei den 50 häufigsten Namen.

8.4 Filterung nach wortinitialen Graphemen

Da sich in der Literatur (insbesondere bei Erikson (1997)) Hinweise darauf fanden, dass bestimmte Merkmale wie der erste Buchstaben (S. 29) oder gar die erste Silbe (S. 49) eine besondere kognitive Bedeutung haben, wurde versucht, ein genaueres Modell für die Kodierung der ersten Grapheme zu entwickeln, als es bei den meisten Verfahren der Fall ist. Ausgenutzt wurde, dass sich der Kontext des Wortanfangs mit einer relativ geringen Anzahl von Regeln beschreiben lässt. Aufbauend auf der Implementierung von Daitch-Mokotoff wurden die Grapheme vor dem ersten Vokal in mehrere Kategorien kodiert. Eine Auflistung findet sich in Abbildung 49.

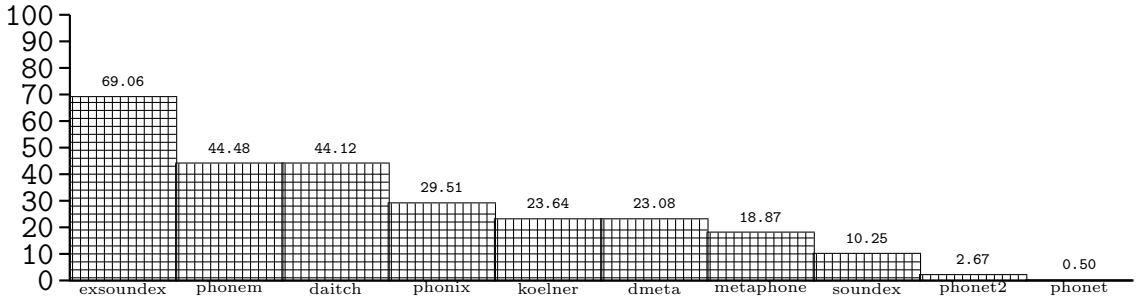

Abbildung 43: Prozentualer Anteil der durch genauere Untersuchung der initialen Grapheme gefilterten Suchergebnisse bei den 50 häufigsten Namen. Vor allem Soundex und Phonem profitieren von einer genaueren Betrachtung der ersten Grapheme.

Das Ergebnis in den Abbildungen 43 und 44 zeigt, dass die meisten Algorithmen hier noch einiges an Spielraum lassen und in diesem Kontext viel zu ungenau arbeiten.

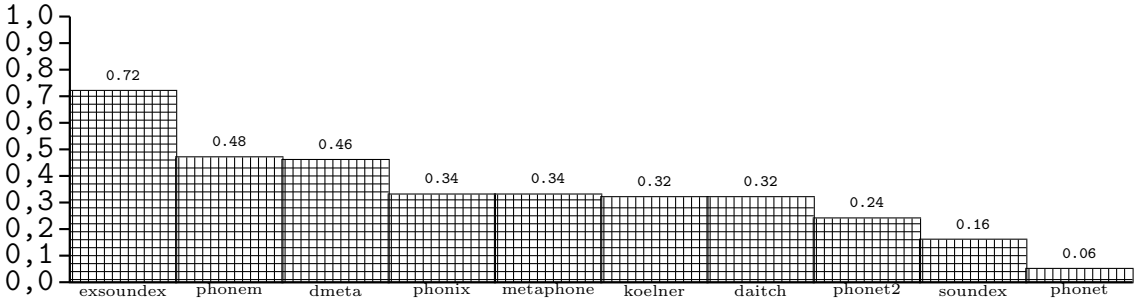

Abbildung 44: Durchschnittliche Anzahl der durch Überprüfung der initialen Grapheme fälschlicherweise gefilterten Suchergebnisse bei den 50 häufigsten Namen.

8.5 Benutzung der vokalischen Information

Wie vorher bereits mehrfach erörtert, verwerfen viele der Verfahren die Information, die durch vokalische Grapheme getragen wird. Als einfaches Experiment wurde ein regulärer Ausdruck entwickelt, der alle Schreibweisen der Vokale /i/ und /a/ erkennt. Gefiltert wurden die Suchresultate, die nicht auf die gleiche Weise auf den Ausdruck matchen, wie es bei der Suchanfrage der Fall ist.

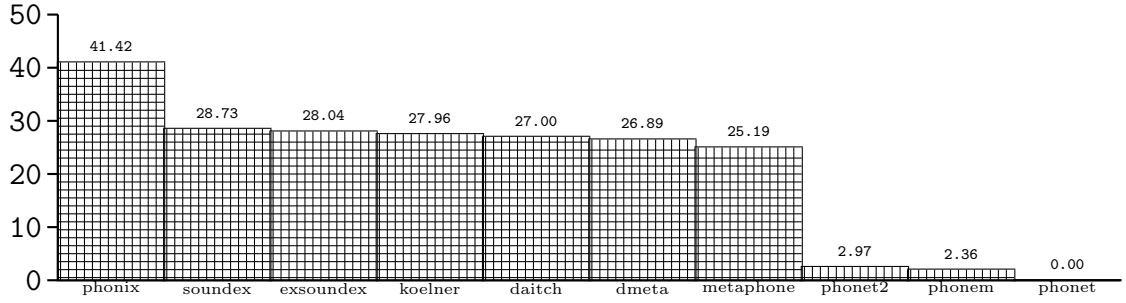

Abbildung 45: Prozentualer Anteil der durch den Abgleich der Graphemvarianten des /a/ bei den 50 häufigsten Namen.

Wenn z. B. nach <Schwarz> gesucht wird, ist <Schwartz> ein Suchresultat, welches ein <a> enthält. Da aber das ebenfalls gefundene <Schwecht> kein <a> enthält, kann es gefiltert werden. Andererseits kann in der Suchabfrage <Simon> kein /a/ realisiert werden. Dementsprechend sind hier Suchergebnisse mit <a>, wie z. B. <Suhrmann> meist nicht erwünscht, obwohl auch ähnliche Namen wie z. B. <Siman> existieren. Hier könnte überlegt werden, ob das <o> ein Phon erzeugen könnte, das dem /a/ ähnlich ist. In den fünfzig häufigsten Namen gab es kein Indiz dafür, so dass hier vorerst darauf verzichtet wurde.

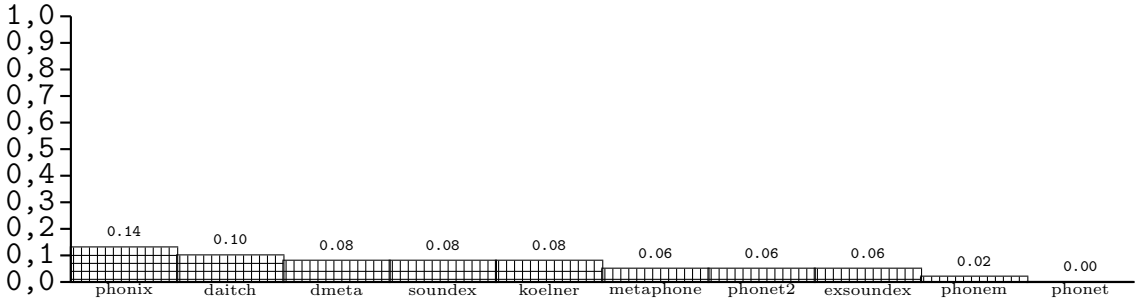

Abbildung 46: Durchschnittliche Anzahl der durch den Abgleich von Graphemvarianten des /a/ fälschlicherweise gefilterten Suchergebnisse bei den 50 häufigsten Namen.

Für die Erkennung des /a/ wurde somit der Ausdruck

```
/a[^eiyju]|er|a$|r$|á|à|â/
```

verwendet. Demnach enthält ein Name ein /a/, wenn es eine Teilzeichenkette <a> hat, auf das kein <e>, <i>, <y>,<i> oder <u> folgt. Auch wenn am Ende der Zeichenkette ein <er>, ein einzelnes <a> oder <r> steht, ist ein Matching mit dem Ausdruck erfolgreich. Die Varianten mit Accent sind der Vollständigkeit halber ebenfalls aufgenommen worden. Sie treten nicht in Kombination mit anderen Vokalen auf und können somit ohne Einschränkungen als /a/ bewertet werden.

Abbildung 46 zeigt, dass dieses Vorgehen in einigen wenigen Fällen noch Probleme bereitet. Allerdings dürfte der Erfolg in Abbildung 45 bestätigen, dass vokalische Information im Deutschen relevant ist.

Analog zum /a/ wurde für das /i/ der reguläre Ausdruck

```
/[^aei][\374iy][^j]|ii|[^qae]ue/
```

für die Detektion von Graphemen verwendet, die ähnlich einem /i/ ausgesprochen werden können. Der angegebene Ausdruck deckt sicherlich nicht alle Fälle ab, in denen ein dem /i/ ähnlicher Laut beschrieben wird, allerdings scheint er bei den fünfzig häufigsten Namen auszureichen. Die Resultate der Filterung werden in den folgenden Abbildungen 47 und

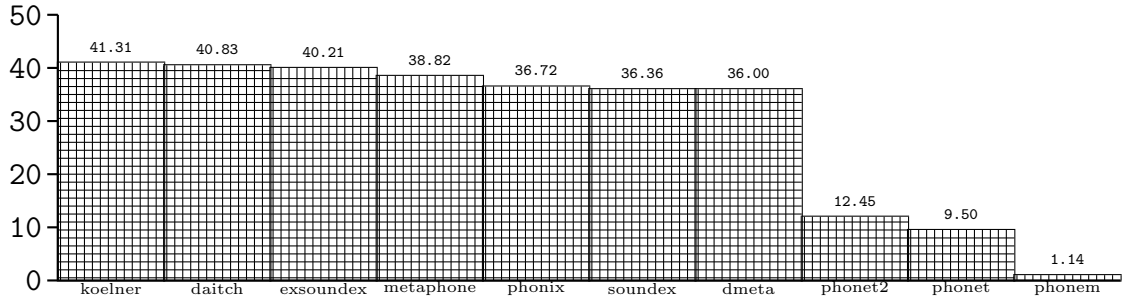

Abbildung 47: Prozentualer Anteil der durch Abgleich der Graphemvarianten des /i/ gefilterten Suchergebnisse der 50 häufigsten Namen.

48 dargestellt. Demnach ist der reguläre Ausdruck für das /a/ für die Filterung ähnlich zuverlässig, jedoch werden weniger richtige Suchresultate entfernt.

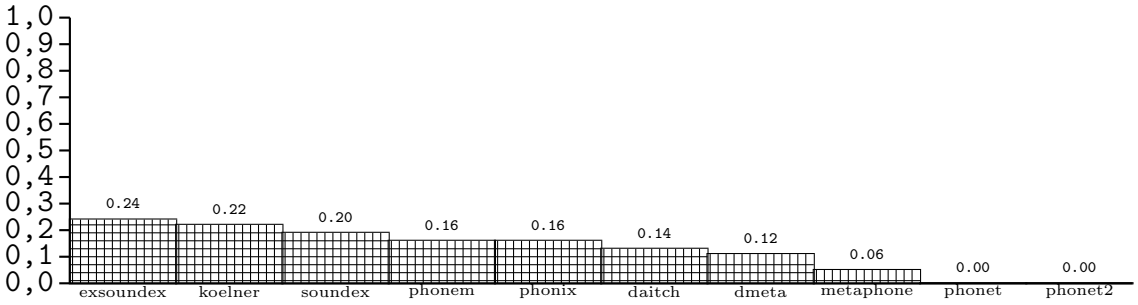

Abbildung 48: Durchschnittliche Anzahl der durch Abgleich von Graphemvarianten des /i/ fälschlicherweise gefilterten Suchergebnisse bei den 50 häufigsten Namen.

9 Schlussbemerkungen

In dieser Arbeit wurde ein Überblick über ausgewählte, relevante Verfahren für eine Phonetische Suche gegeben. Darüber hinaus wurde ein Blick in den Bereich der Methoden für den Vergleich zweier Zeichenketten vorgenommen. Die Verfahren für die Suche wurden anhand eines umfangreichen Korpus auf Ihre Anwendbarkeit für häufige deutsche Familiennamen untersucht. Dabei wurde sowohl die Anzahl von korrekten Suchergebnissen, als auch die Fehlerquote beachtet. Anschliessend wurden aus der Auswertung und vorhandener Literatur gewonnene, ausgewählte Aspekte der Graphem-Phonem-Beziehungen im Deutschen besprochen. Schliesslich wurden mehrere Vorschläge für eine Verbesserung der Fehlerquote der Verfahren für Phonetische Suchen vorgenommen und auf Ihre Anwendbarkeit überprüft. Somit wurden drei einfach zu implementierende Methoden vorgestellt, die die Fehlerquote immens verbessern. Für eine noch bessere Filterung und die Möglichkeit, die Suchergebnisse nach Ähnlichkeit zu sortieren, bleibt die Vermutung, dass die bereits angesprochenen phonetisch/phonologischen motivierten Gewichtungen für Edit-Distance-Algorithmen noch besser für diesen Zweck geeignet sind. Voraussetzung dafür wäre allerdings ein frei verfügbarer, konfigurierbarer und in der Praxis einsetzbarer Graphem-nach-

Phonem-Umsetzer sowie eine Überprüfung der Anwendbarkeit von vorhandenen Ähnlichkeitsmaßen für Phoneme, wie sie z. B. in Vieregge (1985) besprochen werden.

10 Abbildungen

Graphemkombinationen am Wortanfang	Code
ü,y,i,ue,j	i
a,ä,ö,oe,o	a
ou,u	u
b,p	p
br,pr	pr
bl,pl	pl
f,pf,ph,v,w	f
d,t	t
ck,cha,ca,g,k,q,x	k
qur,qr,cr,chr,gr,ghr,kr,khr	kr
gl,cl,chl,ckl,kl	kl
gm,gn,ghm,cm,cn,km,khm,kn	km
c,ch	*
drz,drs,ds,cs,cz,csz,czs	s
l	l
m,n	m
r	r
chm,chn,scm,scn,schm,schn,shm,shn,sm,sn,zm,szm,szn,zn,tsm,tsn,tchm	sm
sh,sc,s,rs,rz,d,szcz,szcs,tch,ttch,ttsch,trz,trs,tsch,tsh,tts,ttsz,tc,tz,ttz,tzs,tsz,ts,z	s
szt,sht,szt,st,shd,szd,sd,scht,schd,shrzt,rst,st	st

Abbildung 49: Tabelle der im Grapheme, die auf den jeweiligen Code abgebildet wurden. Dieser musste gleich sein. Ist der Code ein * kann sowohl ein sch oder auch ein k als gleich bewertet werden.

11 Anhang

11.1 Implementation der „Kölner Phonetik"

```
# Implementation of the Algorithm as described in H.J. Postel, Die Koelner Phonetik
# Ein Verfahren zu Identifizierung von Personennamen auf der Grundlage der Gestalt-
# analyse. IBM-Nachrichten 19 (1969), 925-931
#
# As the original papers does only a description of the method, that is
# ambiguous on many topics, the following implementation tries to achieve
# a model, that at least matches the examples given in the paper.
#
# Martin Wilz <martin@wilz.de> 2005-01-07
#
```

```
package Koelner::Phonetik;

require 5.000;
$VERSION = '1.0';

use strict;

#Umlauts are given by their octal code according to ISO-8859-1
our $vowels = "aeioujy\374\351\366\344";

sub new
{
    my $proto = shift;
    my $class = ref($proto) || $proto;
    my $self  = {};

    bless ($self, $class);
    return $self;
}

sub generateKey
{
    my ($self,$input)=@_;

    my @text=split /\s*/,$input;

    my $key="";

    #handle begin
    my ($char,$nextchar)=@text;

    #encode first char with special rules for word beginning
    if (index ($vowels,$char) >= 0)
    {
$key="0";
    }
    else
    {
if (($char eq "c") and index ("agjkiqruxq",$nextchar) >= 0)
```

```
{
    $key="4";
}
else
{
    my $advance=0;
    ($key,$advance)=$self->getcode ($text[0],$text[1]);
    shift @text if $advance;
}
    }
    shift @text;

    #loop over the rest of the string
    while (@text)
    {
my ($code,$advance) = $self->getcode ($text[0],$text[1]);
#print "$text[0],$text[1],$code\n";
$key .= $code;

shift @text;
shift @text if $advance;
    }

    my $firstchar=substr $key,0,1;

    #remove doubles, vowels function as separators, as they are encoded
    #as "0"

    $key=~tr///cs;

    #remove vowel separators
    $key=~tr/0//d;

    #reconstruct first char if it is a vowel
    $key=$firstchar.$key if $firstchar eq "0";

    return $key;
}

#function for encoding a character. The next character is used as context information
```

```
#the function returns the encoded character and a flag that shows if a further
#character should be skipped

sub getcode
{
    my ($self,$char,$nextchar)=@_;

    $nextchar||= ":";

    return ("1",0) if index ("bp",$char) >= 0;
    return ("8",0) if index ("dt",$char) >= 0 and index ("csz",$nextchar) >= 0;
    return ("2",0) if index ("dt",$char) >= 0 ;
    return ("3",0) if index ("fvw",$char) >= 0;
    return ("3",1) if index ("p",$char) >= 0 and $nextchar eq "h";
    return ("4",0) if index ("gqk",$char) >= 0;
    return ("4",0) if ($char eq "c") and index ("aouhkq",$nextchar) >= 0;
    return ("48",1) if ($char eq "c")  and $nextchar eq "x";
    return ("48",0) if $char eq "x";
    return ("5",0) if $char eq "l";
    return ("6",0) if $char eq "m" or $char eq "n";
    return ("7",0) if $char eq "r";

    #the handling of sc was ambiguous in the paper and deducted from the example
    #encodings given in the paper. So the c is consumed after an s.
    return ("8",1) if $char eq "s" and  $nextchar eq "c";

    return ("8",0) if ("c" eq $char) and index ("aoukxq",$nextchar) == -1;
    return ("8",0) if index ("sz\337",$char)>=0;

    #vowels return 0 to function as separators
    return ("0",0) if index ($vowels,$char)>=0;

    #default: return empty string and advance only one char
    #for example ignore h
    return ("",0);
}
1;
```

11.2 Implementation des Daitch-Mokotoff-Algorithmus

```
# Implementation of the Daitch-Mokotoff-Algorithm as described by
```

```
# http://www.avotaynu.com/soundex.html
#
# Martin Wilz <martin@wilz.de> 2005-01-07
#

package Daitch::Mokotoff;
require 5.000;
$VERSION = '1.0';

#encodings for the beginning of the word
our %initial= (ai=>0,aj=>0,ay=>0,au=>0,a=>0,chs=>5,ei=>0,ej=>0,ey=>0,e=>0,
      eu=>1,h=>5,ia=>1,ie=>1,io=>1,iu=>1,i=>0,oi=>0,oj=>0,oy=>0,
      schtsch=>2,schtsh=>2,schtch=>2,shtch=>2,shch=>2,shtsh=>2,
      stch=>2,stsch=>2,sc=>2,strz=>2,strs=>2,stsh=>2,st=>2,
      szcz=>2,szcs=>2,szt=>2,shd=>2,szd=>2,sd=>2,ui=>0,
      uj=>0,uy=>0,u=>0,ue=>0,y=>1,zdz=>2,zdzh=>2,zhdzh=>2,
      zd=>2,zhd=>2,"\374"=>0,"\351"=>0,"\366"=>0,"\344"=>0
)
;
#encodings before a vowel
my %beforevowel=(ai=>1,aj=>1,ay=>1,au=>7,ei=>1,ej=>1,ey=>1,eu=>1,
 oi=>1,oj=>1,oy=>1,
 h=>5,ks=>5,sht=>2,scht=>2,schd=>2,
 ui=>1,uj=>1,uy=>1
 );

#encodings for any other context
my %other=(b=>7,chs=>54,ch=>"5:4",ck=>"5:45",cs=>4,cz=>4,csz=>4,czs=>4,c=>"5:4",
   drz=>4,drs=>4,ds=>4,dsh=>4,dsz=>4,dz=>4,dzh=>4,dzs=>4,d=>3,dt=>3,
   fb=>7,f=>7,g=>5,j=>"1:4",ks=>54,kh=>5,k=>5,l=>8,mn=>66,m=>6,nm=>6,n=>6,
   p=>7,pf=>7,ph=>7,q=>5,rz=>"94:4",rs=>"94:4",r=>9,
   schtsch=>4,schtsh=>4,schtch=>4,sch=>4,shtch=>4,shch=>4,shtsh=>4,
   sht=>43,scht=>43,schd=>43,sh=>4,stch=>4,stsch=>4,sc=>4,strz=>4,
   strs=>4,stsh=>4,st=>43,szcz=>4,szcs=>4,szt=>43,shd=>43,szd=>43,sd=>43,
   sz=>4,s=>4,"\337"=>4,tch=>4,ttch=>4,ttsch=>4,th=>3,trz=>4,trs=>4,tsch=>4,tsh=>4,
   tts=>4,ttsz=>4,tc=>4,tz=>4,ttz=>4,tzs=>4,tsz=>4,ts=>4,t=>3,
   v=>7,w=>7,x=>"5:54",zdz=>4,zdzh=>4,zhdzh=>4,zd=>43,zhd=>43,
   zh=>4,zs=>4,zsch=>4,zsh=>4,z=>4
   );
```

```
our @initial;
our @order;

sub new
{
    my $proto = shift;
    my $class = ref($proto) || $proto;
    my $self  = {};

    #sort keys of hashes by length for faster access in the encoding methods
    @initial= sort {length $b <=> length $a} (keys %initial, keys %beforevowel,keys %ot
    @order= sort {length $b <=> length $a} (keys %other, keys %beforevowel);

    bless ($self, $class);
    return $self;
}

sub generateKeys
{
    my ($class,$name) = @_;

    $name=~s/\s+//g;

    my $pos = 0;
    my @codes;

    my $match = " ";

    #encode first character
    foreach my $char (@initial)
    {
if (substr ($name,0,length $char) eq $char)
{
    $match = $char;

    if ($pos == 0 and $initial{$match})
    {
push @codes,$initial{$match};
    }
    elsif ((substr ($name,$pos+length $match,1) =~/[aeiouy\374\351\366\344]/)
```

```
    and ($beforevowel{$match}))
    {
push @codes,$beforevowel{$match};
    }
    elsif ($other{$match})
    {
push @codes,$other{$match};
    }
    else
    {
push @codes,0;
    }
    last;
}
    }
    $pos += length $match;

    #encode the rest of the input string
    while ($pos < length $name)
    {
$match = " ";
foreach my $multichar (@order)
{
    if (substr ($name,$pos,length $multichar) eq $multichar)
    {
$match = $multichar;

#check for vowel after the matched string and apply beforevowe rules

if ((substr ($name,$pos+length $match,1) =~/[aeiouy\374\351\366\344]/)
    and ($beforevowel{$match}))
{
    push @codes,$beforevowel{$match};
}
elsif ($other{$match})
{
    push @codes,$other{$match};
}

last;
```

```
    }
}
push @codes,0 if $match eq " ";
$pos += length $match;
    }

    #The codes array now contains the encoded values. These may have to be
    #split up into multiple keys. The needed processing is contained in the
    #Format and CreateVariations function.

    return Format (@codes);
}

sub Format
{
    my @codes = @_;
    my @variations = CreateVariations (@codes);

    #decode and format internal data structures
    foreach my $variation (@variations)
    {
$_=join "", @$_;

s/(.)\1/$1/g;

substr ($_,1) =~ s/0//g if ($_>0);

$_.="0" x (6-length $_);
    }

    return @variations;
}

sub CreateVariations
{
    my @codes = @_;

    if (grep {/:/} @codes)
    {
my $i;
```

```
my @codes2 = @codes;

for ($i = 0;$i < @codes;$i++)
{
    last if $codes[$i]=~/:/;
}

#the colon is used as separator for multiple encoding possibilies
my @options = split ":" , $codes[$i];
my @results;

foreach (@options)
{
    $codes[$i] = $_;
    #call recursively
    push @results, CreateVariations (@codes);
}
return @results;
    }

    return \@codes;
}
1;
```

11.3 Implementation von PHONEM

```
# Implementation of the PHONEM substitutions, as described in
# Georg Wilde and Carsten Meyer, Doppelgaenger gesucht -
# Ein Programm fuer kontextsensitive phonetische Textumwandlung
# ct  Magazin fuer Computer & Technik 25/1998
#
# The original variant was implemented as X86-Assembler-Funktion. This implementation
# does not try to mimic the original code, though it should achieve equal results.
# As the original software used for building the original implementation was not
# available, there was no testing for correctness, other than the examples given
# in the article
#
# Martin Wilz <martin@wilz.de> 2005-01-09

package PHONEM;
```

```
#Hash with encodings for combinations of two characters
#the § is probably to avoid later matching of UE in names like bauer ?
our %doublechars=(SC=>'C', SZ=>'C', CZ=>'C', TZ=>'C', SZ=>'C', TS=>'C',
  KS=>'X', PF=>'V', QU=>'KW', PH=>'V', UE=>'Y', AE=>'E',
  OE=>'Ö', EI=>'AY', EY=>'AY', EU=>'OY', AU=>'A§', OU=>'§ ');

sub PHONEM
{
my $string = uc shift;

#iterate over two character substitutions
foreach my $index (0..((length $string)-2))
{
    if ($doublechars{substr $string,$index,2})
    {
substr ($string,$index,2) = $doublechars{substr $string,$index,2};
    }
}

#single character substitutions via tr
#umlauts are still lower case, since they are not converted by uc
$string =~tr/ZKGQäüIJflFWPT§àáéèúuôoiî/CCCCEYYYSVVBDUAAEEUUOOYY/;

#delete forbidden characters by using the complementary operator
$string =~tr/ABCDLMNORSUVWXYö//cd;

#remove double chars
$string =~tr/ABCDLMNORSUVWXYö//s;
return $string;
}
1;
```

11.4 Implementation der Silbentrennung

```
# Syllabification algorithm tailored for german words as suggested by
# multiple authors as for Example
# Spencer, 1996,  Phonology: theory and description
# ISBN 0-631-19233-6
#
# Martin Wilz <martin@wilz.de> 2004-12-10
```

```
package Text::German::Syllable;

#hash of sonority values
our %sonority = ("-"=>-1,k=>2,x=>2,c=>2,ck=>2,g=>2,q=>1,p=>2,t=>1,h=>1,
 z=>1,s=>1,"\337"=>1,ch=>1,sch=>1,r=>5,l=>4,
 b=>1,d=>1,v=>3,ph=>3,f=>2,w=>3,m=>4,n=>4,
j=>5,y=>8,"\374"=>8,i=>8,ih=>8,u=>8,uh=>8,e=>9,eh=>9,
"\351"=>9,"\366"=>9,"\351h"=>9,"\366h"=>9,o=>9,oh=>9,
"\344"=>10,"\344h"=>10,a=>11,ah=>11);

our @multichar;

sub new
{
    my $proto = shift;
    my $class = ref($proto) || $proto;
    my $self  = {@_};
    our @multichar=sort {length $b <=> length $a} grep {length $_>1} keys %sonority;

    bless ($self, $class);
    return $self;
}

#replace uncommon characters with accents by aequivalent ones
sub convertChars
{
    my $text=shift;
    $text=~tr/\'\140/\-\-/;
    $text=~tr/\350\352\353/eee/;
    $text=~tr/\347/c/;
    $text=~tr/\340\341\342\343/aaaa/;
    $text=~tr/\362\363\364/ooo/;
    $text=~tr/\372\371\373/uuu/;
    $text=~tr/\361/n/;
    $text=~tr/\354\355\356/iii/;
    $text=~tr/\304/e/;
    $text=~s/~//g;
    return $text;
}
```

```
# counts syllables in input text by using splitBySonority.
sub countSyllables
{
    my ($self,$text)=@_;

    #handle foreign characters
    $text=convertChars($text);

    my $out=$self->splitBySonority($text);

    $out =~s/\-\-/-/g;
    $out =~s/o(h?a)/o-$1/g;  #handle oa as syllable boundary

    my @out =split '-',$out;
    return scalar @out;
}

#split text by maximising onsets. As we are using graphemes, some of them have to be
#multiple characters long e.g. <sch> or <ph>
sub splitBySonority
{
    my ($self,$name)=@_;
    my $pos = 0;
    my @units;

    #build array of graphemic units. these are graphemes, that are not
    #separatable and would result in wrong minima, if matched as single graphemes
    while ($pos < length ($name))
    {
my $match;

#first try to match
foreach my $m (@multichar)
{
    $match = $m, last if ((substr $name,$pos,length ($m)) eq $m);
}

#use current char, if multiple characters do not match
$match ||= substr $name,$pos,1;
```

```
push @units,$match;

$pos += length ($match);
    }

    # assign the characters their values. build string with comparison results for
    # every two units. this allows pattern matching to be used for finding local minima
    my $signs = join "",map {($sonority{$units[$_-1]} <=> $sonority{$units[$_]})+1} (1.

    my @tmp;
    my $cnt = 0;

    #use patterns in difference array to find onset minima and prepend a "-" as marker

    foreach my $unit (@units)
    {
push @tmp,$unit;
push @tmp,"-" if (substr ($signs,$cnt++) =~/^21*0/;
    }

    return join "",@tmp;
}

#Optional call into this module. This delivers a string, where syllable boundaries
#have been moved to obtain valid onsets for german words.
sub syllabify
{
    my ($self,$text) = @_;

    #handle foreign chars
    $text=convertChars($text);
    $out=$self->splitBySonority($text);

    #correct onset, if invalid combinations are found. These rules are a first
    #attempt and have to be evaluated

    $out =~s/\-\-/-/g;
    $out =~s/\-g([mn])/g-$1/g;
    $out =~s/\-(s?ch)(ptkbdg)/$1-$2/g;
    $out =~s/\-([dt]+z?)(s?ch|c?k)/$1-$2/g;
```

```
    $out =~s/\-(ph|f+|b+|g+|p+|[dt]+z?|c?k|l+|ch|\337|st)([mn])/$1-$2/g;
    $out =~s/\-([dt]+z?)(l|w|g)/$1-$2/g;
    $out =~s/\-ch(s|h)/ch-$1/g;
    $out =~s/\-st(t|f|ph)/st-$1/g;
    $out =~s/\-s(w|n+|m+)/s-$1/g;
    $out =~s/\-([mn])r/$1-r/g;
    $out =~s/\-nm/n-m/g;
    $out =~s/\-mn/m-n/g;
    $out =~s/o(h?a)/o-$1/g;  #kein gueltiger Diphtong im Deutschen->Silbengrenze

    return $out;
}
1;
```

Literatur

1 Adda-Decker u. a. 2000

ADDA-DECKER, M. ; ADDA, G. ; LAMEL, L.: Investigating text normalization and pronunciation variants for German broadcast transcription. In: *Proceedings ICSLP*. Beijing, Oct 2000, 266-269

2 Adda-Decker u. Lamel 2000

ADDA-DECKER, M. ; LAMEL, L.: Modeling Reduced Pronunciations in German. In: *Proceedings Workshop on Phonetics and Phonology in ASR, PHONUS 5*. Saarbrücken, March 2000, 145-159

3 Belhoula 1993

BELHOULA, Karim: Rule-Based grapheme-to-phoneme conversion of names. In: *Proceedings Eurospeech* (1993), S. 881–884

4 Black u. Llitjos 2001

BLACK, Alan W. ; LLITJOS, Ariadna F.: Knowledge of Language Origin Improves Pronunciation. In: *Proceedings of the European Conference on Speech Communication and Technology Aalborg, Denmark* (2001), Dezember 07. http://www-2.cs.cmu.edu/ awb/papers/ES01lts.pdf

5 Bosch u. Wolters 1997

BOSCH, Antal Van D. ; WOLTERS, Maria: Automatic Phonetic Transcription of Words Based On Sparse Data. In: *Proceedings Workshop on Empirical Methods in Natural Language Processing at the European Conf. on Machine Learning* (1997). ftp://ftp.cs.unimaas.nl/pub/ecml97/wolters-ftp.ps.gz

6 Breuer u. Abresch 2003

Breuer, Stefan ; Abresch, Julia: Unit selection speech synthesis for a directory enquiries service. In: *Proceedings of the 15th International Congress of Phonetic Sciences* (2003)

7 Christiansen u. Torkington 1998

Christiansen, Tohm ; Torkington, Nathan: *Perl Cookbook: Solutions and examples for Perl programmers*. Cambridge, USA : O'Reilly & Associates, Inc., 1998. – xxxiv + 757 S. – ISBN 1–56592–243–3

8 Conway 2000

Conway, Damian: *Object Oriented Perl: A comprehensive guide to concepts and programming techniques*. Conneticut, USA : Manning Publications Co., 2000

9 Damper u. a. 1999

Damper, Robert I. ; Machand, Yannik ; Adamson, Martin J. ; Gustafson, Kjell: Evaluating the pronunciation component of text-to-speech systems for english; a performance comparison of different approaches. In: *Computer Speech and Language* 13 (1999), 155–176. `http://eprints.ecs.soton.ac.uk/archive/00000441/01/eval.pdf`

10 Erdgeist 2002

Erdgeist: Reverse-Engineering für Ortsfremde. In: *Die Datenschleuder* 77 (2002)

11 Erikson 1997

Erikson, Klas: Approximate Swedish name matching - survey and test of different algorithms. In: *Nada report* (1997)

12 Friedl 2002

Friedl, Jeffrey E. F.: *Mastering Regular Expressions*. Second. Cambridge, USA : O'Reilly & Associates, Inc., 2002. – xxii + 460 S. – ISBN 0–596–00289–0

13 Gadd 1988

Gadd, T. N.: Fisching for Werds'. Phonetic Retrieval of written text in Information Retrieval Systems. In: *Program* 22 (1988), Nr. 3, S. 222–237

14 Gadd 1990

Gadd, T. N.: 'PHONIX: The Algorithm. In: *Program* 24 (1990), Nr. 4, S. 363–366

15 Gusfield 1997

Gusfield, Dan: *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997

16 Hyyrö 2003

Hyyrö, Heiki: A bit-vector algorithm for computing Levenshtein and

Damerau edit distances. In: *Nordic Journal of Computing* 10 (2003). citeseer.ist.psu.edu/537930.html

17 Karttunen u. a. 1996

Karttunen, L. ; Chanod, J.-P. ; Grefenstette, G. ; Schille, A.: Regular expressions for language engineering. In: *Nat. Lang. Eng.* 2 (1996), Nr. 4, S. 305–328. – ISSN 1351–3249

18 Knuth 1973

Knuth, Donald E.: *The Art of Computer Programming*. Bd. 3: *Sorting and Searching*. Addison-Wesley, 1973

19 Kohler 1999

Kohler, Klaus: Handbook of the International Phonetic Association. (1999), S. 86–89

20 Krech 2002

Krech, E.-M.: Neukodifizierung der deutschen Standardaussprache. Zur Orthoepieforschung an der Universität Halle. In: Braun, A. (Hrsg.) ; Masthoff, H. R. (Hrsg.): *Phonetics and its Applications*. 2002, S. 506–515

21 Krech 1998

Krech, E.M.: Gegenwärtiger Stand und neueste Ergebnisse bei der Erforschung der deutschen Standardaussprache. In: u.a., B. J. K. (Hrsg.): *Festschrift für G. Heike*. 1998, S. 227–241

22 Kukich 1992

Kukich, Karen: Techniques for Automatically Correcting Words in Text. In: *ACM Computing Survey* 24 (1992), Nr. 4, S. 377–439

23 Kunze 2003

Kunze, Konrad: *dtv-Atlas Namenkunde. 4.Auflage*. Deutscher Taschenbuch Verlag, 2003. – ISBN 3–42333–03266–9

24 Lait u. Randell 1993

Lait, A.J. ; Randell, B.: An Assessment of Name Matching Algorithms. (1993). http://homepages.cs.ncl.ac.uk/brian.randell/home.informal/Genealogy/NameMatching.pdf

25 Levenshtein 1965

Levenshtein, Vladimir L.: Binary codes capable of correcting spurious insertions and deletions of ones. In: *Problems of Information Transmission* 1 (1965), Nr. 1, S. 8–17

26 Lutz u. Greene 2003

Lutz, Richard ; Greene, Stephan: *Measuring phonological similarity: The case of personal names*. Hendon: Language Analysis Systems, 2003. http://www.las-inc.com

27 Mangold 2000

MANGOLD, Max (Hrsg.): *Duden - Das Aussprachewärterbuch, 4.Auflage*. Duden Verlag, 2000. – ISBN 3–411–04064–4

28 Michael 1988

MICHAEL, Jörg: Nicht wörtlich genommen - Schreibweisentolerante Suchroutinen in dBase implementiert. In: *ct Magazin fur Computer & Technik* 10 (1988), S. 126–131

29 Mokotoff 2003

MOKOTOFF, Gary: Soundexing and Genealogy. In: *Avotaynu - Publisher of Works on Jewish Genealogy* (2003). `http://www.avotaynu.com/soundex.html`

30 Mücke 1998

MÜCKE, Doris: Der j-Laut im Deutschen: Normierung und Gebrauchsnorm. In: U.A., B. J. K. (Hrsg.): *Festschrift für G. Heike*. 1998, S. 97–113

31 Müller 2000a

MÜLLER, Karin: Probalistische kontext-freie Grammatiken für Silbifizierung und Graphem-Phonem-Konvertierung. In: *AIMS Report* 6 (2000), Nr. 2. `http://staff.science.uva.nl/ kmueller/Onlinepapers/pcfg.ps.gz`

32 Müller 2000b

MÜLLER, Karin: Silbifizierung im Deutschen mit Hilfe einer lexikalisierten probabilistischen kontextfreien Grammatik. In: *AIMS Report* 6 (2000), Nr. 4. `http://staff.science.uva.nl/ kmueller/Onlinepapers/soell.ps.gz`

33 Navarro 2001

NAVARRO, Gonzalo: A guided tour to approximate string matching. In: *ACM Computing Surveys* 33 (2001), Nr. 1, 31–88. `citeseer.ist.psu.edu/navarro99guided.html`

34 Niklfeld 1995

NIKLFELD, Georg: Phonologische Datenstrukturen in einem Concept-to-Speech System: Segmentale und Prosodische Phonologie. In: *Austrian Research Institute for Artificial Intelligence* (1995)

35 Park u. Park 1992

PARK, Keith ; PARK, Tracy: *Family History UK*. Familiy History Club, 1992. – ISBN 1–873594–04–6

36 Patman u. Shaefer 2003

PATMAN, Frankie ; SHAEFER, Leonard: *Is Soundex Good Enough for You? On the Hidden Risks of Soundex-Based Name Searching*. Hendon: Language Analysis Systems, 2003. `http://www.las-inc.com`

37 Pfeifer u. a. 1995

Pfeifer, Ulrich ; Poersch, Thomas ; Fuhr, Norbert: Searching Proper Names in Databases. In: *Hypertext - Information Retrieval - Multimedia, Synergieeffekte elektronischer Informationssysteme, Proceedings HIM '95*. Konstanz : Universitätsverlag Konstanz, April 1995 (Schriften zur Informationswissenschaft), S. 259–276

38 Philips 1990

Philips, Lawrence: Hanging on the Metaphone. In: *Computer Language Magazine* 7 (1990), Dezember, Nr. 12, S. 38–. – ISSN 0749–2839

39 Philips 2000

Philips, Lawrence: The Double Metaphone Search Algorithm. In: *C/C++ Users Journal* 18 (2000), Juni, Nr. 6. – ISSN 1075–2838

40 Postel 1969

Postel, H-J.: Die Kölner Phonetik - Ein Verfahren zur Identifizierung von Personennamen auf der Grundlage der Gestaltanalyse. In: *IBM-Nachrichten* 19 (1969), S. 925–931

41 Raghavan u. a. 1989

Raghavan, V. V. ; Bollmann, P. ; Jung, G. S.: Retrieval system evaluation using recall and precision: problems and answers. In: *SIGIR '89: Proceedings of the 12th annual international ACM SIGIR conference on Research and development in information retrieval*, ACM Press, 1989. – ISBN 0–89791–321–3, S. 59–68

42 Scholz 2004

Scholz, Christoph P.: Lexikalische Entlehnungen aus dem Englischen. In: *Fachbereich Philosophie und Geisteswissenschaften, Freie Universität Berlin* (2004). `http://www.diss.fu-berlin.de/2004/234/index.html`

43 Schwartz u. Christiansen 1997

Schwartz, Randal ; Christiansen, Tom: *Learning Perl*. Second. Cambridge, USA : O'Reilly & Associates, Inc., 1997. – xxix + 269 S. – ISBN 1–56592–284–0

44 Siebs 1969

Siebs, Theodor ; Boor, Helmut de (Hrsg.) ; Moser, Hugo (Hrsg.) ; Winkler, Christian (Hrsg.): *Deutsche Aussprache*. Walter de Gryter & Co, 1969. – ISBN 3–11–000325–2

45 Spencer 1996

Spencer, Andrew: *Phonology: theory and description*. Blackwell Publishers Inc., 1996. – ISBN 0–631–19233–6

46 Telekom 2000

Das Telefonbuch. Für Deutschland. Frühjahr 2000. 2000

47 Thielen 1995

Thielen, Christine: An approach to proper name tagging in German. In: *Proceedings EACL SIGDAT.*

48 Vieregge 1985

Vieregge, Wilhelm: Ein Maß zur Reliabilitätsbestimmung phonetisch-segmentaler Transkriptionen. In: *Zeitschrift für Dialektologie und Linguistik* (1985), Nr. 52, S. 167–180

49 Vitale 1991

Vitale, Tony: An Algorithm for High Accuracy Name Pronunciation by Parametric Speech Synthesizer. In: *Computational Linguistics* 17 (1991), Nr. 3, 257-276. `citeseer.ist.psu.edu/vitale91algorithm.html`

50 Wilde u. Meyer 1988

Wilde, Georg ; Meyer, Carsten: Doppelgänger gesucht - Ein Programm für kontextsensitive phonetische Textumwandlung. In: *ct Magazin fur Computer & Technik* 25 (1988)

51 Wothke 1993

Wothke, K.: Morphologically based automatic transcription. In: *IBM Systems Journal* 32 (1993), Nr. 3

52 Zobel u. Dart 1996

Zobel, J. ; Dart, P. W.: Phonetic String Matching: Lessons from Information Retrieval. In: Frei, H.-P. (Hrsg.) ; Harman, D. (Hrsg.) ; Schäble, P. (Hrsg.) ; Wilkinson, R. (Hrsg.): *Proceedings of the 19th International Conference on Research and Development in Information Retrieval*. Zurich, Switzerland : ACM Press, 1996, S. 166–172