Version vom 14.12.2008

Dipl.-Phys. Jörg Michael
Adalbert-Stifter-Str.11, 30655 Hannover
astro.joerg@googlemail.com

<sic!> bis </sic!> : nicht ändern

Technische Verfahren zum Personen- und Adressdatenabgleich

Zusammenfassung

Bei einem Personen- und Adressdatenabgleich hat man immer wieder mit den verschiedensten Fehlern zu tun. In diesem Artikel werden die in der Praxis vorkommenden Fehlerarten ("normale" Tippfehler, Ablesefehler, phonetische Fehler und "semantische" Fehler, sowie Besonderheiten bei arabischen, russischen und asiatischen Namen) erläutert.

Mehrere frei verfügbare Verfahren für unscharfen, fehlertoleranten Personen- und Adressdatenabgleich werden diskutiert.
Dazu zählen insbesondere die Hannoveraner Phonetik, Funktionen zur Anrede- und Vornamenskorrektur anhand eines europa- und weltweiten Vornamenlexikons, die phonetisch erweiterte Levenshtein-Funktion und darauf aufbauende Verfahren für unscharfe Datenbank- und Dublettensuche.

Inhaltsverzeichnis

1. Vorkommende Fehler

Welche Fehler kommen beim Namen und Anschriften vor?
Kurz gesagt, alle möglichen.
Im Folgenden soll eine kleine Übersicht gegeben werden.

1.1. "Normale" Tippfehler

1.1.1. einfacher Tippfehler:
- Comarstr. statt Colmarstr.
- Dörfstr. statt Dorfstr.

1.1.2. einfacher Buchstabendreher:
- Joesph statt Joseph
- Geothestr. statt Goethestr.

1.1.3. komplexere Tippfehler:
- Rimmenthal statt Rinnenthal
- Birrothstr. statt Billrothstr.

1.1.4. Doppelter Buchstabendreher:
- "Bitte mit Anageb der Telefonnummer."

1.2. Fehler durch fehlerhaftes Ablesen

1.2.1. einfache Fehler bei Handschrift:
- Gutenau statt Gatenau
- Georg-Elfer-Str. statt Georg-Elser-Str.
- "n" vs. "u", z.B.: Hansu vs. Hausu

1.2.2. "doppelte" Fehler bei Handschrift:
- Rast statt Riest
- "lz" vs. "br", z.B.: Stobrenau statt Stolzenau
- "cl" vs. "d", usw.

1.2.3. Fehler bei Druckschrift:
- ni <-> m
- H <-> M, usw.

1.2.4. doppelte Tippfehler:
- Foellerweg statt Forellenweg
- Industirstr. statt Industriestr.
- Platzstr. statt Pfalzstr.

1.3. Phonetische Fehler (durch Hören entstanden)

1.3.1. "einfache" phonetische Fehler:
- Kibitzweg statt Kiebitzweg
- Hämmergasse statt Hemmergasse
- Braunenbergstr. statt Braunbergstr.
- Ennestr. statt Ennestr.

1.3.2. "mehrfache" phonetische Fehler:
- Konschieta statt Conchita
- Safalzstr. statt Saar-Pfalz-Str.

1.4. Beispiele für "semantische" Fehler

Durch meine Programm zur Korrektur von Postanschriften (siehe Kapitel 7) habe ich gelernt, dass es zusätzlich zu den bereits genannten Fehlerarten noch verschiedene Arten "semantischer" Fehler gibt.

Die folgenden Beispiele geben eine Auswahl davon wieder.

1.4.1. Beispiele für fehlerhafte Straßen:
- Abtweg statt Abtstr. (nicht Astweg)
- Bachstr. statt Johann-Sebastian-Bach-Str.
- Prenskerstr. statt Benjamin-Preusker-Str. (enthält "u" <--> "n")
- Heimatshausen statt Heimathshausener Str.
- Herzfelder Weg vs. Herzfelder Str., Wohngebiet Herzfelder Weg
- Holzhausen vs. Holzhausen bei Theisenberg,
  Holzhausener Str., Holzhauser Str.

1.4.2. Beispiele für "pseudo-phonetische" Fehler:
- Markstr. statt Marktstr.
- Rankestr. statt Lampestr.
- Partner statt Vater

1.4.3. Beispiele für "grob fehlerhafte" Anschriften:
- Liebigstr., 60329 München (korrekt: 80538 oder Frankfurt)
- Stockholmer Str., 18107 Berlin (korrekt: 13359 oder Rostock)
- Heinrich-Mann-Str., 19053 Berlin (korrekt: 13156 oder Schwerin)
- Bernhardstr., 01187 Berlin (korrekt: 10715 oder Dresden)

1.5. Anmerkung zur "statistischen Unabhängigkeit" von Fehlern

Die Wahrscheinlichkeit für eine fehlerhaft eingegebene Anschrift liegt meiner Erfahrung nach bei unter 2%, wobei die Hälfte davon keine echten Fehler sind, sondern unterschiedliche Schreibweisen darstellen, etwa "Dorfstraße" statt "Dorfstr.". Damit verbleiben noch ca. 1% echte Fehler.

Wäre die Annahme der statistischen Unabhängigkeit von Fehlern immer erfüllt, dann müsste die Wahrscheinlichkeit für vier Tippfehler bei der Postleitzahl bestenfalls in der Größenordnung von 1 zu 100 Millionen liegen. Ein derartiger Fehler dürfte bei einer Million Anschriften also garnicht oder höchstens einmal vorkommen.

Wie man an den Beispielen aus Abschnitt 1.4.3. ersehen kann, stimmt dies leider überhaupt nicht. Tatsächlich kommen solche Fehler ca. 1000-mal öfter vor als es der "theoretischen" Wahrscheinlichkeit entspricht.

Man muss sogar davon ausgehen, dass die verschiedenen Arten "semantischer" Fehler jeweils einer eigenen Statistik unterliegen, die für jede Fehlerart separat ermittelt werden muss. Da die absoluten Fehlerhäufigkeiten ziemlich gering sind, braucht man dafür eine sehr große Stichprobe.

2. Besonderheiten bei ausländischen Namen

2.1. Besonderheiten bei arabischen Namen

In der arabischen Schrift (ebenso in der hebräischen Schrift)
werden die Vokale weggelassen <sic!> nd nr d Knsntn fgeschrbn </sic!>.

Dies führt beim Namen "Mhmd" u.a. zu folgenden Varianten:
Mohamad, Mohamed, Mohammad, Mohammed, Mouhamad, Mouhammad,
Muhammad, Muhamed, Muhammed (die Liste ist nicht vollständig)

2.2. Besonderheiten bei russischen Namen

2.2.1. Transkription aus dem kyrillischen Alphabet
Die Transkription vom kyrillischen Alphabet in das lateinische Alphabet
ist nicht immer eindeutig. Beispiele:
kyrillisch "B" -> lateinisch "V" oder "W"
kyrillisch "C" -> lateinisch "S" oder "SS"
bulgarisch "Bukmop" -> lateinisch "Viktor" oder "Wiktor" (oder englisch "Victor")

Wenn ein Name (wie etwa "Wassilij") drei solche Buchstaben enthält,
sind teilweise bis zu zwölf verschiedene Transkriptionen möglich.

2.2.2. Weißrussland und Ukraine
In Weißrussland und der Ukraine werden manche Namen sogar im Kyrillischen
anders geschrieben als in Russland und sehen dann aus wie Tippfehler.
Manchmal sind die Namenspaare nicht einmal ähnlich.

Beispiele:

Russland	Ukraine	Belarus (Weißrussland)
Aleksandr, Alexander	Oleksandr	Aliaksandr
Nikolai, Nikolaj	Mykola	Mikalaj
Oleg	Oleh	Aleh
Olga	Olha	Volha
Sergei, Sergej	Serhij	Siarhei, Siarhej
Tatjana, Tatyana	Tetjana	Tatsiana
Vladimir, Wladimir	Volodymyr	Uladzimir ("U" !!)

2.3. Besonderheiten bei asiatischen Namen

Im asiatischen Raum ist es üblich, dass der Familienname vor dem
Vornamen steht. Bei der Berichterstattung wird diese Reihenfolge
häufig beibehalten.

Beispiele:

China :	Deng Xiaoping	(Deng ist der Nachname)
Korea :	Hwang Woo-Sook	(Hwang ist der Nachname)
Vietnam:	Nguyen Thi Thu Thanh	(Nguyen ist der Nachname)

2.4. Die häufigsten Tippfehler bei Vornamen

Die häufigsten Tippfehler bei Vornamen sind:
- Guiseppe statt Giuseppe
- Hava statt Havva
- Ayshe (oder Aysche) statt Ayse
- Dimitri (bei russischen Namen) statt Dmitri
- Ranghild statt Ragnhild

2.5. Tippfehler oder nicht?

Könnten Sie auf Anhieb sagen, welche der folgenden Namen korrekt sind?

Necaattin
Necaddin
Necaettin
Necaittin
Necatdin
Necbettin
Necmetdin
Necmetin
Necmetlin
Necnettin
Nedjimedin
Nedjmedin
Negjmidin
Negjnedin
Negmettin
Nehadin
Nejdettin
Nejmadin
Nejmeddin
Nejmettin
Nejmittin
Nemettin
Neredin
Nerettin
Netzadin
Netzatin
Netzmentin
Nexhemdin
Nexhemedin
Nexmedin
Nezedin
Nezmedin
Nezmettin
Nezmoddin
Nezreddin
Nigmettin
Nizamethin
Nizametin
Nizamuddin
Nizmattin
Nizzamettin
Njazedin

3. Automatische Anredebestimmung

3.1. Europa- und weltweites Vornamenlexikon

Die Wahrscheinlichkeit für eine falsch eingegebene Anrede liegt meiner Erfahrung nach bei ca. 0,5 % und kann bei seltenen Namen ausländischer Herkunft noch darüber liegen. Eine automatische Anredeprüfung macht daher Sinn.
Hierfür ist es hilfreich, zu wissen, dass seit letztem Jahr ein europa- und weltweites Vornamenlexikon mit mehr als 40.000 Namen frei verfügbar ist [Michael 2007a].

Zur Anredebestimmung dient dabei die Funktion "get_gender".

Diese Funktion hat u.a. folgende symbolische Returnwerte (als Makros):

```
  IS_FEMALE           Mädchenname
  IS_MALE             Jungenname
  IS_UNISEX_NAME      Unisex-Name
  IS_MOSTLY_FEMALE    Unisex-Name, der meistens weiblich ist
  IS_MOSTLY_MALE      Unisex-Name, der meistens männlich ist
  NAME_NOT_FOUND      unbekannter Name
```

3.2. Statistik und Qualität des Vornamenlexikons

In der aktuellen Version 1.2 (vom 30.11.2008) sind mehr als 3.000 Namen dazugekommen. Damit sieht die Statistik des Vornamenlexikons jetzt folgendermaßen aus:

```
Anzahl Vornamen         :   44566
   Mädchennamen         :   16919
   Jungennamen          :   17740
   Unisex-Namen         :    9907

Namen aus Albanien        :  1748
Namen aus Arabien/Persien :  2039
Namen aus Armenien        :   653
Namen aus Azerbaijan      :   764
Namen aus Belgien         :  1518
Namen aus Bosnien/Herzeg. :  1216
Namen aus Bulgarien       :  1793
Namen aus China           :  7334
Namen aus Dänemark        :  1139
Namen aus Deutschland     :  2635
Namen aus Ostfriesland    :  2589
Namen aus Estland         :  1121
Namen aus Finnland        :   873
Namen aus Frankreich      :  1821
Namen aus Georgien        :   293
Namen aus Griechenland    :   778
Namen aus Großbritannien  :  2231
Namen aus India/Sri Lanka :  1457
Namen aus Irland          :   876
Namen aus Island          :  1302
Namen aus Israel          :  1067
Namen aus Italien         :  2871
Namen aus Japan           :  1386
Namen aus Kasachstan/Usbek.,etc.:   862
Namen aus Korea           :  1377
Namen aus Kosovo          :  1002
Namen aus Kroatien        :   895
Namen aus Lettland        :   710
Namen aus Litauen         :   845
```

```
Namen aus Luxemburg       :   564
Namen aus Malta           :   526
Namen aus Mazedonien      :   851
Namen aus Moldavien       :   405
Namen aus Montenegro      :   621
Namen aus den Niederlanden:  3303
Namen aus Norwegen        :  1064
Namen aus Österreich      :  1753
Namen aus Polen           :   359
Namen aus Portugal        :  1041
Namen aus Rumänien        :  1971
Namen aus Russland        :   490
Namen aus Schweden        :   974
Namen aus Schweiz         :  2464
Namen aus Serbien         :   727
Namen aus Slovakei        :   477
Namen aus Slovenien       :   636
Namen aus Spanien         :  1645
Namen aus Türkei          :  1807
Namen aus Tschechien      :   478
Namen aus Ukraine         :   531
Namen aus Ungarn          :   371
Namen aus U.S.A.          :  3916
Namen aus Vietnam         :   308
Namen aus Weißrussland    :   508
```

Die Qualität des Vornamenlexikons ist mittlerweile in vielen Fällen besser als bei länderbezogenen Vornamenlisten in Wikipedia, die von einem "native speaker" erstellt wurden.

3.3. Korrektur fehlerhafter Vornamen

Die Erfahrung hat gezeigt, dass gegebene Vornamen, die NICHT im Lexikon vorhanden sind, mit großer Wahrscheinlichkeit Tippfehler darstellen.

Dafür wurde in Version 1.2 die Funktion "find_similar_name" neu geschaffen. Sie dient dazu, zu einem gegebenen Vornamen die wahrscheinlichste Korrektur zu ermitteln.

Die Funktion wertet dazu neben der Wortähnlichkeit (gemessen durch die phonetische Levenshtein-Distanz; siehe Kapitel 5) auch die im Lexikon zu jedem Namen verfügbaren länder-bezogenen Häufigkeits-Angaben aus.

So gibt es beim Tippfehler "Ursla" zwei Korrekturen, die von der Ähnlichkeit her gleichwertig sind, nämlich "Ursela" und "Ursula".
Da der Name "Ursula" aber mehr als 100-mal so oft vorkommt, stellt er auch mit mehr als 100-mal größerer Wahrscheinlichkeit die richtige Korrektur dar.

Die Laufzeit liegt auf heutigen Rechnern in der meisten Fällen in der Größenordnung von ca. 1 Millisekunde.

3.3.1. Welche Namen sind korrigierbar?

Tippfehler zu "Wolfgang":
Es gibt keinen ähnlichen Namen. Daher kann jeder Tippfehler leicht als solcher erkannt werden.

Tippfehler zu "Barbara":
Zumindest im deutschsprachigen Raum gibt es keinen ähnlichen Namen.

Tippfehler zu "Maria":
Viele ähnliche Namen. Daher gibt es viele Möglichkeiten, durch einen Tippfehler einen anderen gültigen Vornamen zu erzeugen.

3.3.2. Varianten und ähnliche Vornamen von "Maria"
(vorneweg steht jeweils das Anredekürzel:
F=Frau, M=Mann, ?=Unisex-Name, ?F=meistens weiblich).

a) Varianten von "Maria":

F Maria
F Mária
F María

F Maaria
F Mariah
F Mariea
F Marija
F Mariya
F Marja
F Marya
F Maryia

F Maria da Conceição
F Maria da Glória
F Maria de Fátima
F Maria de Jesus
F María de la Concepción
F María del Carmen
F María del Consuelo
F María del Mar
F María de los Ángeles
F Maria de Lourdes
F María del Pilar
F María del Rosario
F Maria de Lurdes
F Maria do Carmo
F Maria do Rosário

b) ähnliche Vornamen zu "Maria":

F Amrei
F Aria
F Daria
F Dária
F Faria
F Maira

F Mara
F Marca
F Marcia
F Márcia
F Marga

?F Mari
F Mariam
? Marian
M Marían

M Marián
M Marías

F Marica
F Marie
F Marii
F Marij
F Marika
F Marila
M Marin
F Marín
F Marina

M Mario
M Mário
F Maris
M Maris
M Marís
F Marita
F Marla
F Marna
F Marta
F Márta
F Marva
F Marwa
F Marzia
F Miria

3.3.3. "Standardisierung" arabischer und russischer Vornamen
Eine weitere sinnvolle Anwendung der Funktion "find_similar_name" besteht in der "Standardisierung" arabischer und russischer Vornamen (sozusagen als "Pseudo-Phonetik").

Hierzu braucht man einfach "nur" den ersten Vornamen der Ergebnisliste nehmen.

Beispiele:
a) Das Suchergebnis für "Mouhammed" ist:
"Mohamed; Mohammad; Mohammed; Mohamad; Muhammad; Muhammed; Muhamed; Mouhamed; Mouhammed"

b) Das Suchergebnis für "Siarhey" ist:
"Sergey; Siarhey"

3.4. Schnittstellen für utf-8 und Unicode

Die Funktionen "get_gender" und "find_similar_name" sind unter ähnlichen Namen (mit Suffix "_utf8" bzw. "_unicode") auch für utf-8 und Unicode verfügbar [Michael 2007a] [Unicode und utf-8].

4. Hannoveraner Phonetik (Programm "phonet")

4.1. Einleitung

Phonetische Verfahren dienen häufig dazu, in Datenbanken besser nach gleich oder ähnlich klingenden Namen suchen zu können. "Phonetisch" bedeutet dabei, dass aus einem Quellstring ein Ergebnisstring ermittelt wird und das Umwandlungsergebnis idealerweise nur von der Aussprache abhängt, nicht aber von der Schreibweise eines Namens. Die Namen "Meier", "Meyer", "Maier" und "Mayer", "Mair" sollen also alle auf denselben Ergebnisstring abgebildet werden.

4.2. Ältere Verfahren

Das wahrscheinlich bekannteste phonetische Verfahren ist Soundex [Soundex] [Wilde und Meyer 1988]. Es handelt sich dabei um ein relativ einfaches, für die englische Sprache entwickeltes Verfahren (das dort auch gut genug funktioniert). Dabei wird der erste Buchstabe unverändert übernommen und alle nachfolgenden Konsonanten unter Weglassung der Vokale gruppenweise zu Zahlen umgewandelt. Damit haben "Levin", "Lippmann", "Lopian" und "Lubnow" denselben Soundex-Code, nämlich "L15".

Aufgrund der größeren Bedeutung der Vokale in der deutschen Sprache ist dieses Verfahren hierzulande nicht so gut anwendbar. Deutlich besser ist da schon die Kölner Phonetik, die als Soundex-Variante mit Anpassungen für die deutsche Sprache entwickelt wurde und schon eine Reihe (wenngleich einfacher) Kontextabhängigkeiten berücksichtigt [Postel 1969] [Kölner Phonetik].

Wegen der immer wieder vorkommenden Ausnahmen stößt aber auch die Kölner Phonetik leicht an ihre Grenzen. Dazu muss man sich nur einmal klar machen, dass zum Beispiel der Buchstabe "T" manchmal wie ein "Z" ausgesprochen wird. In der neuen Rechtschreibung wurde dies beim Wort "Potenzial" korrigiert, aber "Tradition" schreibt sich immer noch traditionell mit "t". Eine "gute" Phonetik sollte daher auch diese Kontextabhängigkeiten berücksichtigen.

4.3. Das Programm "phonet"

Das Ziel bei der Entwicklung von "phonet" [Michael 1999] bestand darin, genau dies zu erreichen und einer "idealen Phonetik" nicht nur möglichst nahe zu kommen, sondern dabei auch noch gleichzeitig eine möglichst hohe Programmgeschwindigkeit zu erreichen.

Es sieht so aus, dass dies einigermaßen gut gelungen ist, denn die Resonanz auf die Veröffentlichung hat klar gezeigt, dass damit ein neuer Quasistandard für deutschsprachige Phonetik geschaffen wurde.

Mittlerweile existiert auch eine Java-Version von "phonet":
siehe https://opensource.softmethod.de/trac/opensource
nebst Click auf "phonet4j".

4.4. Architektur des Programms

Die Architektur von "phonet" besteht aus drei Schichten:
a) Auf der "unteren" Ebene findet sich die Syntax für die phonetischen Regeln sowie der zugehörige Parser.
b) Die zweite Ebene wird durch die phonetischen Regeln repräsentiert.
c) Die dritte Ebene schließlich wird durch die Funktion "check_rules" dargestellt.

Die letzte Funktion ist für die Qualität des Programms von entscheidender Bedeutung, denn aufgrund der hohen Anzahl an phonetischen Regeln und der Vielzahl ihrer gegenseitigen Abhängigkeiten wäre eine manuelle Prüfung von vornherein zum Scheitern verurteilt.

4.5. Phonetische Regeln

a)
"phonet" bietet zwei verschiedene Regelsysteme mit insgesamt ca. 1.000 Regeln, welche durch Angabe der Option "strict" oder "fuzzy" benutzt werden.
Der Name "Mayer" z. B. wird bei "strict" zu "MEIA" und bei "fuzzy" zu "NEIA" umgewandelt

Die Option "strict" benutzt man, wenn man nur solche Namen sucht, die wirklich gleich klingen, während man die Suche mit der Option "fuzzy" etwas unschärfer gestalten kann.

b)
Syntax für phonetische Regeln

Die Syntax für phonetische Regeln sieht wie folgt aus:

```
<suchstring>  <1.regel>  <2.regel>
```

Syntax für Suchstrings:

```
<wort> [<->..] [<] [<0-9>] [^[^]] [$]
```

Manche der Regeln (insbesondere diejenigen für Vorsilben ("^^")) stellen eine Art verkappter Silbentrennungsalgorithmus dar.

c)
Die Reihenfolge der phonetischen Regeln beinhaltet gleichzeitig eine Priorität, denn es wird grundsätzlich die erste passende Regel genommen. Spezialfälle müssen daher vor den "allgemeinen" Regeln stehen.

4.6. Weitere Leistungsmerkmale des Programms

a)
Sehr hohe Geschwindigkeit dank zweier miteinander kombinierter Hash-Verfahren.

b)
Beim Design der phonetischen Regeln wurde darauf geachtet, dass sich der Informationsverlust möglichst gleichmäßig auf die einzelnen Buchstaben verteilt.

So wurde bei der Entwicklung des Programms auch untersucht, ob es sinnvoll ist, die Buchstaben "V", "W" und "F" bzw. die Buchstaben "C", "K" und "Z" "gleich zu machen", wie es einige einfache Programme (etwa das Programm "PHONEM" aus [Wilde und Meyer 1988]) tun.

Nach kurzer Untersuchung wurde aber schnell klar, dass der Informationsverlust bei diesen Buchstaben damit überproportional groß werden würde. Diser Ansatz wurde daher verworfen, und somit musste insbesondere für die Umwandlung der Buchstaben "C" und "V" ein besonders detailliertes Regelwerk erarbeitet werden.

c)
Die phonetischen Regelwerk enthält auch eine Reihe von Regeln für neue bzw. alte Rechtschreibung sowie für wichtige Fremdwörter, Vornamen und Ortsnamen.

In der aktuellen Version 1.5 (vom 30.11.2008) sind zusätzlich einige Dutzend Regeln
für wichtige ausländische, insbesondere russische, Vornamen hinzugekommen.
"Siarhei" wird damit auf denselben Namen abgebildet wie "Sergej".

4.7. Phonetische Regeln für den Buchstaben "N"

Als Beispiel seien nachfolgend die phonetischen Regeln für den Buchstaben "N" aufgelistet:

```
"NACH^^",          "NACH",    "NAK",
"NADINE^$",        "NADIN",   "NATIN",
"NAIV--",          "NA",      "NA",
"NAISE$",          "NESE",    "NEZE",
"NAUGENOMM------", "NAU ",    "NAU ",
"NAUSOGUT$",     "NAUSO GUT", "NAUZU KUT",
"NCH$",            "NSH",     "NZ",
"NCOISE$",         "SOA",     "ZUA",
"NCOIS$",          "SOA",     "ZUA",
"NDAR$",           "NDA",     "NTA",
"NDERINGEN------", "NDE ",    "NTE ",
"NDRO(CDKTZ)-",    "NTRO",    null,
"ND(BFGJLMNPQVW)-", "NT",     null,
"ND(SßZ)$",        "NS",      "NZ",
"NÇOISE$",         "SOA",     "ZUA",
"NÇOIS$",          "SOA",     "ZUA",
"ND'S$",           "NS",      "NZ",
"ND´S$",           "NS",      "NZ",
"NEBEN^^",         "NEBN",    "NEBN",
"NENGELERN------", "NEN ",    "NEN ",
"NENLERN(ET)---",  "NEN LE",  "NEN LE",
"NENZULERNE---", "NEN ZU LE", "NEN ZU LE",
"NE(LMNRST)-3^",   "NE",      "NE",
"NEN-3",           "NE",      "NE",
"NETTE$",          "NET",     "NET",
"NGU^^",           "NU",      "NU",
"NG(BDFJLMNPQRTVW)-", "NK",   "NK",
"NH(AUO)-$",       "NI",      "NI",
"NICHTSAHNEN-----", "NIX ",   "NIX ",
"NICHTSSAGE----",  "NIX ",    "NIX ",
"NICHTS^^",        "NIX",     "NIX",
"NICHT^^",         "NICHT",   "NIKT",
"NINE$",           "NIN",     "NIN",
"NON^^",           "NON",     "NUN",
"NOTLEIDE-----^",  "NOT ",    "NUT ",
"NOT^^",           "NOT",     "NUT",
"NTI(AIOU)-3",     "NZI",     "NZI",
"NTIEL--3",        "NZI",     "NZI",
"NTYNA",           "NTINA",   "NTINA",
"NT(SßZ)$",        "NS",      "NZ",
"NT'S$",           "NS",      "NZ",
"NT´S$",           "NS",      "NZ",
"NYLON",           "NEILON",  "NEILUN",
"NY9^",            "NÜ",      null,
"NSTZUNEH---",     "NST ZU ", "NZT ZU ",
"NSZ-",            "NS",      null,
"NSTS$",           "NS",      "NZ",
"NZ(BDFGKLMNPQRSTVWX)-", "NS", null,
"N(SßZ)$",         "NS",      null,
```

4.8. Vergleich mit Soundex und Kölner Phonetik

Soundex und die Kölner Phonetik arbeiten wesentlich unschärfer als die Hannoveraner Phonetik. Dies spiegelt sich auch in der jeweiligen Trefferzahl wider.

Bei der Datenbank-Suche nach einem Namen findet im Durchschnitt:

```
- Soundex          9,6 unterschiedliche Namen
- Kölner Phonetik  3,7 unterschiedliche Namen
- phonet (strict)  1,2 unterschiedliche Namen
- phonet (fuzzy)   1,4 unterschiedliche Namen
```

Der hohe (und eher schlechte) Wert für Soundex darf insofern nicht überraschen, weil dieses Verfahren für den englischen Sprachraum entwickelt worden ist.

5. Die phonetisch erweiterte Levenshtein-Funktion

5.1. Einleitung

Für einen fehlertoleranten Anschriftenvergleich braucht man als zentralen Kern zunächst einmal ein Ähnlichkeitsmaß für die Unterschiede zweier Strings.

Bekannte Verfahren sind hierbei:
a) Trigramme [Rapp 1997]
b) Die Levenshtein-Funktion [Levenshtein 1965] [Ebner 1989]

Bei einem Vergleich mittels Trigrammen wird die Anzahl von Tripeln benachbarter Buchstaben bestimmt, welche in beiden Wörter gleichzeitig vorkommen.

Die Levenshtein-Funktion zählt, grob gesagt, die Unterschiede zwischen zwei Strings. Bei der Berechnung wird dabei iterativ vorgegangen und eine so genannte Distanzmatrix bestimmt, an deren Position (i,j) jeweils die Distanz der aus i bzw. j Buchstaben bestehenden Teilstrings steht. Am Ende der Rechnung findet sich "rechts unten" in der Matrix die gesuchte Levenshtein-Distanz.

Beispiel: Abt vs. Ast:

		A	S	T
	0	1	2	3
A	1	0	1	2
B	2	1	1	2
T	3	2	2	1

Da sich beider Wörter in genau einem Buchstaben unterscheiden, ergibt sich, wie es auch sein sollte, die Levenshtein-Distanz 1.

5.2. Vergleich von Trigrammen und Levenshtein-Funktion

Wie man an der Beschreibung vielleicht schon erahnen kann, verhalten sich Trigramme und Levenshtein-Funktion bei einem Stringvergleich unterschiedlich.
Während Trigramme in erster Linie die Ähnlichkeiten zweier Strings messen, zählt die Levenshtein-Funktion vor allem die Unterschiede.

Weitere Unterschiede ergeben sich bei Abkürzungen, denn die Levenshtein-Funktion kann auf einfache Weise auf die üblichen Wildcards '*' (inklusive '.') und '?' erweitert werden [Michael 1994].
Eine Erweiterung auf Phonetik ist ebenfalls gut machbar.
Bei Trigrammen macht beides konzeptionelle Schwierigkeiten.

Nicht zuletzt besteht bei der Levenshtein-Funktion die Möglichkeit, durch einen Rückgabewert in Hundertstel-Fehlern eine deutlich bessere Granularität zu erreichen [Michael 2007b]. Damit hat man die Chance, bei ähnlich klingenden Buchstaben beispielsweise 0,3 Fehlerpunkte zurückzugeben. Wenn man nur ganze Zahlen zu Verfügung hat, müsste man sich demgegebenüber immer für 0 oder 1 entscheiden.

5.3. Erweiterung der Levenshtein-Funktion um Trigramme und eine native Phonetik

Um die Vorteile der verschiedenen Verfahren nutzen zu können, habe ich
in der aktuellen Version 1.2 (vom 30.11.2008) des Anschriftenvergleichs
die Levenshtein-Funktion mit Trigrammen und einer regelbasierter nativen
Phonetik für die deutsche Sprache kombiniert.
Die Trigramme bewirken dabei eine bessere Berücksichtigung von Ähnlichkeiten.

Bedeutsamer dürfte jedoch der Einschluss einer nativen Phonetik sein.
"Nativ" bedeutet hierbei eine echte Integration einer regelbasierten Phonetik.
Der besondere Vorteil liegt dabei in einer erneut verbesserten Granularität. So
geben Meyer, Meier, Maier und Mayer jeweils leicht unterschiedliche Fehlerpunkte.

Um diese Verbesserung auch nach außen zu dokumentieren, habe ich das
Programm "lev100.c" (aus dem Artikel "unscharfer Anschriftenvergleich")
in "lev100ph.c" umbenannt.

5.4. Beispiele für einen Stringsvergleich mittels phonetischer Levenshtein-Funktion

a) Rast vs. Riest:

		'R'	'I'	'E'	'S'	'T'
	0.00	1.00	2.00	2.05	3.05	4.05
'R'	1.00	0.00	1.00	1.05	2.05	3.05
'A'	2.00	1.00	1.00	1.05	2.05	3.05
'S'	3.00	2.00	2.00	2.00	1.05	2.05
'T'	4.00	3.00	3.00	3.00	2.05	1.05

Trigramme = 0
Levenshtein-Distanz = 1.05

Da die Buchstabenfolge "IE" fast wie "I" klingt, sollte die Levenshtein-Distanz
nur ein klein wenig größer sein als 1, was auch der Fall ist.

b1) Konschieta vs. Conchita:

		'C'	'O'	'N'	'C'	'H'	'I'	'T'	'A'
	0.00	1.00	2.00	3.00	4.00	4.30	5.30	6.30	7.30
'K'	1.00	0.30	1.30	2.30	3.30	3.60	4.60	5.60	6.60
'O'	2.00	1.30	0.30	1.30	2.30	2.60	3.60	4.60	5.60
'N'	3.00	2.30	1.30	0.30	1.30	2.30	3.30	4.30	5.30
'S'	4.00	3.30	2.30	1.30	1.30	2.30	3.30	4.30	5.30
'C'	5.00	4.00	3.30	2.30	1.30	2.30	3.30	4.30	5.30
'H'	5.00	4.30	3.30	2.30	2.30	0.40	1.40	2.40	3.40
'I'	6.00	5.30	4.30	3.30	3.30	1.40	0.40	1.40	2.40
'E'	6.20	5.50	4.50	3.50	3.50	1.60	0.45	1.40	2.40
'T'	7.20	6.50	5.50	4.50	4.50	2.60	1.45	0.45	1.45
'A'	8.20	7.50	6.50	5.50	5.50	3.60	2.45	1.45	0.45

Trigramme = 1
Levenshtein-Distanz = 0.45

b2) Wie b1), aber mit Fehlerlimit 2:

```
            'K'   'O'   'N'   'S'   'C'   'H'   'I'   'E'   'T'   'A'
      0.00  1.00  2.00  ----  ----  ----  ----  ----  ----  ----  ----
'C'   1.00  0.30  1.30  ----  ----  ----  ----  ----  ----  ----  ----
'O'   2.00  1.30  0.30  1.30  ----  ----  ----  ----  ----  ----  ----
'N'   ----  ----  1.30  0.30  1.30  ----  ----  ----  ----  ----  ----
'C'   ----  ----  ----  1.30  1.30  1.30  ----  ----  ----  ----  ----
'H'   ----  ----  ----  1.60  1.60  1.80  0.40  1.40  1.45  ----  ----
'I'   ----  ----  ----  ----  ----  ----  1.40  0.40  0.45  1.45  ----
'T'   ----  ----  ----  ----  ----  ----  ----  1.40  1.40  0.45  1.45
'A'   ----  ----  ----  ----  ----  ----  ----  ----  ----  1.45  0.45
```

Trigramme = 1
Levenshtein-Distanz = 0.45

Man beachte:
Ein Vergleich von "Konschieta" und "Conchita" mittels Trigrammen würde einen sehr großen Unterschied ergeben, weil nur ein einziges passendes Trigramm gefunden wird (nämlich "CHI").

5.5. Levenshtein-Ähnlichkeit zweier Strings

Die subjektive Ähnlichkeit zweier Strings richtet sich neben der Levenshtein-Distanz natürlich auch noch nach den Stringlängen. So haben die Stringspaare "Abt und Ast" die Levenshtein-Distanz 1. Das Gleiche gilt aber auch für "A." und "D." sowie für "Christine" und "Christiane".

Um ein Ähnlichkeitsmaß für Strings zu erhalten, muss man die Levenshtein-Distanz also noch mit den jeweiligen Stringslängen und der Anzahl und Verteilung eventueller Wildcards und Abkürzungen ('*', '?' und '.') in Beziehung setzen.

Sinnvoll ist es dabei, aus den Stringlängen und den Wildcards eine Schwelle für den maximal zulässigen Fehler zu bestimmen, der dann als Referenzmarke für den Vergleich mit der Levenshtein-Distanz dient. Für Strings ohne Wildcards liegt eine sinnvolle höchstzulässige Fehlerzahl ungefähr bei einem Viertel bis einem Drittel der Stringlänge, denn wenn man noch mehr Fehler zulässt, werden auch "Meier" und "Müller" als ähnlich bewertet.

Beispiele:
Das Programm "lev100ph.c" berechnet für "Konschieta" und "Conchita" eine Ähnlichkeit von 90,3 %. Für "Rast" und "Riest" ergibt sich demgegebenüber ein deutlich kleinerer Wert von nur 44,2%.

6. Unscharfer Anschriftenvergleich (Programm "addr")

6.1. Einleitung

Einfache Verfahren für fehlertoleranten Anschriftenvergleich beruhen darauf, für jedes Feld je nach Fehlerzahl bzw. Übereinstimmung eine bestimmte, feste vorgegebene Punktzahl zu vergeben und am Schluss durch Addition der Einzelwerte die Gesamtpunktzahl zu bestimmen.

Für "kleinere" Datenbanken in der Größenordung bis ca. 300.000 Anschriften reicht dies sogar vollkommen aus
(siehe zum Beispiel die Programme in [Michael 1994]).

Bei größeren Datenbeständen braucht man differenziertere Verfahren, und dann muss man insbesondere den Informationsgehalt der einzelnen Anschriftenfelder genauer berücksichtigen.

6.2. Gewichtungen

Der Informationsgehalt eines Datenbankfeldes (und damit auch die Gewichtung) ergibt sich aus der "Breite" der Verteilung.
Dann es macht bei einer Suche einen großen Unterschied, ob in einer Datenbank im Wesentlichen seltene oder im Wesentlichen häufige Nachnamen zu finden sind.

Mathematisch formuliert: Der Filterfaktor des häufigsten Eintrags ist entscheidend.
Im Programm "addr" aus [Michael 2007b] wird folgende Formel verwendet:

```
punkte = 10 * log(filterfaktor)
```

Als "log" wird dabei der dekadische Logarithmus genommen. Die Normierung ist also dergestalt, dass 100 Punkte rechnerisch einem Filterfaktor von 1 zu 10 Milliarden entsprechen.

Da die Verteilungen von Land zu Land unterschiedlich sind, sind auch die Gewichtungen für jedes Land unterschiedlich. Das Programm "addr" enthält Vorgaben für ein Dutzend verschiedene Länder, die in Abhängigkeit vom Nationalitätenkennzeichen benutzt werden.

Für Deutschland werden folgende Gewichtungen benutzt:

```
Anrede   :   4   Punkte
Vorname  :  18   Punkte
Nachname :  20,7 Punkte
Straße   :  37,4 Punkte
PLZ      :  28,7 Punkte
Land     :   3   Punkte
Geb.-tag :  39,7 Punkte
Kundennr.:  50   Punkte
```

Die Gewichtung für den Nachnamen ergibt sich daraus, dass der häufigste Nachname "Müller" ist und ungefähr jeder 117-te Deutsche diesen Namen hat.

Das Geburtsjahr nimmt eine Sonderrolle ein, denn je nach Datenbestand (z.B. Einwohnermeldeämter oder Schulen und Universitäten) sieht die "Breite der Verteilung" eventuell ganz anders aus. Der obige Werte ist für "mittlere" Werte ausgelegt und sollte daher ggf. geändert werden.

6.3. Abschläge beim Fehlen von Informationen

Die Gewichtung eines Feldes gibt "nur" die maximal mögliche Punktzahl an. Das Vorhandensein von Wildcards und Abkürzungen ('*', '?' und '.') ist gleichbedeutend mit dem Fehlen bzw. Nichtvorhandensein von Informationen und führt daher zu Abschlägen bei der erreichbaren Maximalpunktzahl.

Denn die Namen "Chr." und "Christian" können zu den gleichen Personen gehören - oder auch nicht. Ein Abschlag führt dazu, dass zwar komplette Übereinstimmung festgestellt wird, die Maximalpunktzahl für dieses Feld jedoch deutlich reduziert wird.

Wenn ein Feldeintrag leer ist, wird die Maximalpunktzahl für dieses Feld auf Null gesetzt. Dies kann man beispielsweise im Trace-Protokoll aus Kapitel 6.5. begutachten.

6.4. Besonderheiten beim Vor- und Nachnamen

Bei der Bedeutung der Wortendungen gibt es deutliche Unterschiede zwischen Vor- und Nachnamen. So ist bei Vornamen der letzte Buchstabe wichtig, denn er bildet sehr häufig die Unterscheidung zwischen weiblichen und männlichen Vornamen, etwa bei "Inge" und "Ingo" oder bei "Maria", "Marie" und "Mario".

Bei Nachnamen hingegen ist das Wortende häufig wenig unterscheidungskräftig, insbesondere wenn der letzte Buchstabe ein 'e' oder 's' ist (wie etwa bei "Schulz" und "Schulze" oder bei "Meyer" und "Meyers").

Wenn der Nachname wie ein gültiger Vorname aussieht, sind außerdem sehr leicht Namensdreher zwischen Vor- und Nachname möglich (beispielsweise "Günther, Otto" versus "Günther Otto").

Familienangehörige stellen eine Anschriftensuche gelegentlich vor erhöhte Probleme.
Problematisch sind insbesondere Ehepartner mit ähnlichen Geburtstagen und sehr ähnlichen Vornamen (z.B. "Christian" und "Christina").
Zur Verbesserung der "Trennschärfe" werden daher "Überkreuzvergleiche" gemacht, um die Unterschiede stärker hervorzuheben. Dies reicht normalerweise aus, um beide Datensätze als unterschiedlich zu klassifizieren.

6.5. Ein Anschriftenvergleich im Detail

Um die Arbeitsweise beim Anschriftenvergleich besser nachvollziehen zu können, bietet es sich an, zwei Adressen mit aktivierter Trace-Option zu vergleichen, etwa die beiden fiktiven Anschriften:

a) Herr Karl-Heinz Müller, Hauptstr. 147, PLZ = 12345, geboren 14.5.1967
b) Herr Karl Mueller, Hauptstraße 147, PLZ = 12345, geboren " ".05.67

Das Trace-Protokoll sieht folgendermaßen aus:

```
first address:
gender = 'M', customer number = ''
first name = 'Karl-Heinz', family name = 'Müller', c/o name = ''
street = 'Hauptstr. 147', city = '12345'', country = ''
birthday (yyyy-mm-dd) = '1967-05-14'
phone = '', mobile = '', email = ''
```

```
IBAN code = '', bank account = ''

second address:
gender = 'M', customer number = ''
first name = 'Karl', family name = 'Mueller', c/o name = ''
street = 'Hauptstraße 147', city = '12345'', country = ''
birthday (yyyy-mm-dd) = '67-05- '
phone = '', mobile = '', email = ''
IBAN code = '', bank account = ''

min_points = 0
Note:  country for first address is not set;
       country will be set to default country ("D").

Note:  country for second address is not set;
       country will be set to default country ("D").

set weights according to country:  "D"

gender:  result = GENDER_IS_EQUAL,
gender:  points = 4,  max_points = 4,  diff = 0,  empty_diff = 0

family name:  points = 20.7,  max_points = 20.7,  diff = 0,  empty_diff = 0
total for "nam":  points = 24.7,  max_points = 24.7,  diff = 0,  empty_diff = 0

lev_2_name: diff = 0
first name:  points = 11.99,  max_points = 18,  diff = 1,  empty_diff = 0
total for "nam":  points = 36.69,  max_points = 42.7,  diff = 1,  empty_diff = 0

total points for "nam" (after "extra checks with name"):
      points = 36.69,  max_points = 42.7,  diff = 1,  empty_diff = 0

ZIP code/city:  points = 28.7,  max_points = 28.7,  diff = 0,  empty_diff = 0
total points for "place":  points = 28.7,  max_points = 28.7,  diff = 0,  empty_diff =
0

country:  points = 0,  max_points = 0,  diff = 0,  empty_diff = 0
total points for "place":  points = 28.7,  max_points = 28.7,  diff = 0,  empty_diff =
0

street:  points = 14.96,  max_points = 14.96,  diff = 0,  empty_diff = 0
total points for "place":  points = 43.66,  max_points = 43.66,  diff = 0,  empty_diff
= 0

day of birth:  points = 0,  max_points = 1.85,  diff = 0,  empty_diff = 1.85
month of birth:  points = 10.7,  max_points = 10.7,  diff = 0,  empty_diff = 0
year of birth:  points = 14.2,  max_points = 14.2,  diff = 0,  empty_diff = 0
total points for birhday:  points = 24.9,  max_points = 26.75,  diff = 0,  empty_diff =
1.85

normalized total points = 97.63

final result:
points = 97,  min_points = 0,  is_family_member = 0
```

(Wie es auch sein muss, werden beide Anschriften als sehr ähnlich bewertet.)

6.6. Unscharfe Datenbanksuche

Das Programm "addr" bietet Funktionen für eine unscharfe Datenbanksuche an. Bei einer unscharfen Datenbanksuche sollte man Fehler in allen Fehlern zulassen. Eine ausreichend schnelle Suche ist jedoch nur dann gewährleistet, wenn man Datenbank-Indexe nutzen kann.

Um diese teilweise widersprüchlichen Anforderungen "unter einen Hut zu bekommen", sollte man für das Programm "addr.c" folgende Indexe erzeugen:
a) Postleitzahl und Familienname
b) Vollständiger Geburtstag und Vorname
c) Kunden-, Vertrags- oder Patientennummer.

Anschließend kann man das Modul "dbselect.c" für eine unscharfe Datenbanksuche nutzen. Man muss dabei noch die Mindestpunktzahl für einen "Treffer" angeben. Empfohlene Werte liegen im Bereich von 80 bis 90.

6.7. Unscharfe Dublettensuche

Bei unbereinigten Datenbanken sind 4 - 5 % Dubletten durchaus normal.
Das Programm "addr" bietet daher benfalls Funktionen für eine unscharfe Dublettensuche an. Um schnell zu sein und gleichzeitig die verschiedenen Arten von Fehler "zu erwischen", arbeitet das Programm "addr" daher mit mehreren Unload-Dateien.

Zur Erzeugung der bis zu sieben Unload-Dateien kann z.B. das Modul "dedupl" benutzt werden:

```
dedupl  -unload  <unload_file_1>  [ .. <unload_file_7> ]
```

Vor der eigentlichen Dublettensuche muss man die Unload-Dateien noch sortieren. Nach dem Sortieren stehen Dubletten mit großer Wahrscheinlichkeit hintereinander in der Datei, so dass man die Dubletten durch einfaches sequentielles Lesen mit dem Modul "dedupl" ermitteln kann:

```
dedupl -search_duplicates <sorted_unload_file> <dest_file>  [ <min_points> ]
```

Für eine intiale Dublettensuche reichen die ersten ein oder zwei Unload-Dateien aus, weil darin erfahrungsgemäß bereits die meisten "echten" Dubletten enthalten sind.

7. Automatische Anschriftenprüfung und -korrektur

7.1. Einleitung

Auch ohne nähere Analyse sollte klar sein, dass eine automatische Korrektur fehlerhafter Anschriften erheblich schwieriger und komplexer ist als eine simple Ähnlichkeitsprüfung, denn für eine gute Korrektur muss man ggf. unter verschiedenen ähnlichen Anschriften diejenige ermitteln, die unter Berücksichtigung von Tippfehlern, phonetischen Fehlern, Ablesefehlern, semantischen Fehlern sowie den ja auch gelegentlich vorkommenden Änderungen von Straßennamen, Postleitzahlen oder Ortsnamen an besten passt.

7.2. Qualität

Der Autor von "phonet" und "addr" hat ein solchen Programm zur Prüfung und Korrektur von deutschen Postanschriften entwickelt.

Wie Sie sicherlich nicht anders erwarten, sind Qualität und Geschwindigkeit des Programms sehr gut.

Bei einem Vergleich mit einer kommerziell verfügbaren Standardanwendung war dieses Programm nicht nur schneller, sondern auch bei der Fehlerkorrektur klar überlegen. Es ist geplant, das Programm kommerziell zu nutzen.

7.3. Eigenschaften der Fehlerkorrektur

Ein besonderes Leistungsmerkmal dieses Programms besteht darin, dass nicht nur Einzelfehler korrigiert werden können (das ist noch einfach), sondern auch Doppel- und (bei hinreichend hoher Ähnlichkeit) sogar Dreifachfehler (d.h. Straße, PLZ und Ort sind falsch) ebenfalls korrigiert werden können.

Beispiele:
(zwecks Anonymisierung sind die Hausnummern weggelassen):

Straße, PLZ und Ort falsch, aber korrigierbar:
a) Feuersicht, 31295 Stobrenau

Andere Fehlerarten (Korrekturvorschlag wird gemacht):
b) Neuhof, 35792 Löhnberg-Niedershausen
c) Maxstr., 83274 Traunstein
d) Klerstr., 38120 Braunschweig
e) Dorfstr., 07806 Neunhofen

8. Prüfziffernverfahren

Wo immer Zahlen zur Identifikation dienen (sei es als Kunden-, Konto- oder Patientennummer), bietet es sich an, Prüfziffernverfahren zu Fehlerprüfung und eventuellen -korrektur einzusetzen.

Wie die optimalen Verfahren für "einfache" Prüfziffern (also ohne Korrekturmöglichkeit) aussehen, hat J. Verhoeff bereits 1969 in seiner Doktorarbeit untersucht [Verhoeff 1969]. Das Ergebnis lautet:
Bei Modulo-Verfahren sollte man Modulo11 benutzen [Verhoeff 1969] [Michael 1996].
Die Alternative, also das "zweitbeste" Verfahren, basiert auf Diedergruppen (gesprochen Di-Eder) [Verhoeff 1969] [Michael 1997].

Natürlich kann man versuchen, noch besser zu werden. Das Verfahren, welches Faldum und Pommerening diskutieren [Faldum und Pommerening 2005], ist in der Tat beeindruckend gut, denn es können auch eine Reihe verschiedenartiger Tippfehler korrigiert werden.

Wenn zwei verschiedene "Codewörter" den Mindestabstand 3 haben, bedeutet ein doppelter Tippfehler aber auch, dass die so entstandene fehlerhafte Eingabe mit einiger Wahrscheinlichkeit den Abstand 1 zu einem anderen "korrekten" Codewort hat.

Bezüglich der Annahme einer "statistischen Unabhängigkeit" von Fehlern gilt das bereits in Kapitel 1.5 gesagte.

9. Literaturhinweise

[Ebner 1989] G. Ebner: Wort-Arithmetik, Phonetische Ähnlichkeiten mit der Levenshtein-Distanz errechnet, c't, Heft 07/1989, S. 192-208.

[Faldum und Pommerening 2005] A. Faldum, K. Pommerening: An optimal code for patient identifiers, Computer Methods and Programs in Biomedicine, vol. 79, S. 81-88 (2005).

[Kölner Phonetik] Kölner Phonetik: http://de.wikipedia.org/wiki/Kölner_Phonetik

[Levenshtein 1965] Vladimir I. Levenshtein: Binary Codes Capable of Correcting Deletions, Insertions and Reversals, Soviet Physics Doklady, vol. 10, S. 707-709 (1965).

[Michael 1994] J. Michael: Joker im Spiel, Erweiterung der Levenshtein-Funktion auf Wildcards, c't, Heft 03/1994, S. 230-239.

[Michael 1996] J. Michael: Mit Sicherheit, Prüfziffernverfahren auf Modulo-Basis, c't, Heft 07/1996, S. 264-268.

[Michael 1997] J. Michael: Blütenrein, Prüfziffernverfahren auf der Basis von Diedergruppen, c't, Heft 04/1997, S. 448-452.

[Michael 1999] J. Michael: Doppelgänger gesucht, Ein Programm für kontextsensitive phonetische Textumwandlung, c't, Heft 25/1999, S. 252-261.

[Michael 2007a] J. Michael: 40000 Namen, Anredebestimmung anhand des Vornamens, c't, Heft 17/2007, S. 182-183.

[Michael 2007b] J. Michael: Von Hinz und Kuntz, Ein Programmpaket zur fehlertoleranten Anschriftensuche, c't, Heft 20/2007, S. 214-219.

[Postel 1969] Hans-Joachim Postel: Die Kölner Phonetik, Ein Verfahren zur Identifizierung von Personennamen auf der Grundlage der Gestaltanalyse, IBM-Nachrichten, Band 19, S. 925-931 (1969).

[Rapp 1997] R. Rapp: Text-Detektor, Fehlertolerantes Retrieval ganz einfach, c't, Heft 04/1997, S. 386-392 (Artikel über Trigramme).

[Soundex] Soundex: http://de.wikipedia.org/wiki/Soundex

[Unicode und utf-8] Unicode und utf-8: siehe Wikipedia

[Verhoeff 1969] J. Verhoeff, Error Detecting Decimal Codes, Mathematical Centre Tracts, vol. 29 (Mathematisch Centrum, Amsterdam, 1969).

[Wilde und Meyer 1988] G. Wilde, C. Meyer: Nicht wörtlich genommen, "Schreibweisentolerante" Suchroutinen in dBase, c't, Heft 10/1988, S. 126-131
(Artikel über "Soundex" und eine dort vorgestellte Soundex-Variante namens "phonem").

#----------

Zur Person des Autors

Dipl.-Phys. Jörg Michael
Adalbert-Stifter-Str.11, 30655 Hannover
astro.joerg@googlemail.com

Physikstudium an der Technischen Universität Hannover.
Diplomarbeit in Quantenoptik. Abschlussnote: 1,8

Tätigkeiten
Langjährige Programmierung und Systemverwaltung im Bereich Finanzdienstleistung,
Programmierung in C und Cobol für Kunden- und Adressdaten-Verwaltung,
fehlererkennende und fehlerkorrigierende Verfahren jedweder Art,
elektronische Archivierung und Workflow.

Entwicklung eines Programms zur Prüfung und Korrektur von deutschen Postanschriften.
Es ist geplant, das Programm kommerziell zu nutzen.

Veröffentlichungen

J. Michael:
Joker im Spiel, Erweiterung der Levenshtein-Funktion auf Wildcards, c't, Heft 3/1994, S.230-239.

J. Michael:
Mit Sicherheit, Prüfziffernverfahren auf Modulo-Basis, c't, Heft 7/1996, S.264-268.

J. Michael:
Blütenrein: Prüfziffernverfahren auf der Basis von Diedergruppen, c't, Heft 4/1997, S.448-452.

J. Michael:
Doppelgänger gesucht, Ein Programm für kontextsensitive phonetische Textumwandlung,
c't, Heft 25/1999, S.252-261.

J. Michael:
Wie vor gut 100 Jahren das Rätsel der Saturnringe gelöst wurde,
Sterne und Weltraum, Juli 2000, S.590-591.

J. Michael:
40000 Namen, Anredebestimmung anhand des Vornamens, c't, Heft 17/2007, S.182-183.

J. Michael:
Von Hintz und Kunz, Ein Programmpaket zur fehlertoleranten Anschriftensuche,
c't, Heft 20/2007, S. 214-219.

Außerdem diverse Kurzartikel und Leserbriefe in "Sterne und Weltraum",
"Spektrum der Wissenschaft", "Sky and Telescope" und "New Scientist".