Fuzzy matching as a technique for the name correction

Petr Matousek ⋆
matousp@dcse.fee.vutbr.cz

Abstract This paper introduces methods that can be used to correct those errors in electronic mail addresses which occur when a recipient name in the header of an e-mail message is misspelled or mistyped. The target mail delivery system should automatically detect the error and correct it. We use the term "fuzzy match" to mean the best approximate match in this context. By combination of various techniques – simple string comparison, approximate matching up to 2 allowed mistakes, soundex matching etc. – we implement fuzzy matching to achieve good results in a reasonable time. This paper covers an analysis of applied techniques and their drawbacks (i.e. problems with search speed, search imprecision and misinterpretation). It also gives some ideas for the future improvement. Following this strategy a program called mailrouter was designed which demonstrates practical employment with required results ⋆⋆.

Key Words: Fuzzy match, name correction, approximate match, soundex, name recognition

1 Introduction

Mistyped e-mail addresses are a common problem in every mail system. It arises when a user does not know the correct name of the addressee or when he unintentially makes a typo. The problem is more visible in institutions where people from different countries are working. One of such large international organisations is CERN - The European Laboratory for Particle Physics. Scientists from different language domains (European, Russian, Chinese, Japanese) need to communicate with each other via e-mail. The standard format of e-mail addresses at CERN is *Firstname.Surname@cern.ch* – the so called Generic E-mail Address. The transcription of non-english alphabets to English can, however, produce various forms of the same name (e.g. Iouri, Yuri, Jouri).

Of course, this problem does not appear only in electronic mail. However, the electronic mail system is not a human: it misses the additional knowledge that every postman has. How can we add this knowledge and in what way should it be processed ? Is it ever possible to describe this knowledge ?

At first we need to define what type of mistakes we have to face and how to eliminate them completely or partially. In the following text we introduce methods that are more suitable for this task. These methods require additional knowledge to be added and we need to define weights evaluating different kinds of knowledge. These and other topics are discussed in the following sections.

2 Problem specification

A person who does not think *twice* will realize that the issue is not easy. Many systems search a simple database of names and build the result on a match by using a Soundex algorithm

⋆ Department of Computer Science and Engineering, FEI VUT Brno, Božetěchova 2, CZ-612 66 Brno

⋆⋆ The project was realized at CERN, Geneva, Switzerland as a part of technical student programme

(explained later). Or they simply return an error message without doing any fuzzy match. Shall we ever deal with this problem ?

The requirements are as follows:

- find *"the best possible match"* in reasonable time;
- refuse wrong addresses;
- if more than one entry in database are matched a list of possible recipients is returned;
- keep the list of recipients small (to protect private data and to avoid mis-use by commercial mails); and
- correct the address and forward the mail to the sender only in the case of unique exact match (if an address is incomplete but not mistyped - e.g. P.Matousek or Matousek instead of Petr.Matousek).

2.1 The best possible match

The term *best possible match* raises the question of how to measure the **distance** between an input name and the correct value. The relevant terminology can be introduced by the following way.

- If the searched name corresponds precisely to a database entry we call it **an exact match**. The distance for an exact match is zero.
  - If only one entry matches exactly (unique match) we found *the exact match*.
  - If more than one matches exactly (ambiguous names like Smith) we have *an ambiguous match*.
- If the name differs from any database value (*approximate match*) we count the number of **differences**. A difference can be an insertion, a deletion or a substitution of a character - this approach is specified in the agrep implementation (see [1]). The distance corresponds to the number of differences.

The best match is a subset of possible matched entries (exactly or approximately matched) having minimal distance.

For example: Difference 1 means that a match includes one extra character (insertion) or one character missing (deletion) or one character changed (substitution). Be aware that the distance (difference) between the names Matousek and Matuosek is 2 (two substitutions !).

In the section about the Soundex algorithm we will define **the soundex distance** which is different from the distance as defined above.

2.2 Correction versus rerouting

When an approximate or an exact match was found and we succeeded to correct the address, we should reroute the message to the sender.

Did we really find the correct address ?

Even if the distance is 0 (exact match) who is able to say what the sender's intention was ? If the recipient is Harvey - did the sender send mail to John.Harvey (without specifying a firstname) or to Harvey.Newman (omitting the surname) or is it another mistake ?

It is a policy problem. What will happen if someone receives an e-mail letter not meant for him ? Although the e-mail is proclaimed not to be confidential service, lots of people send private and confidential information.

To quote an e-mail expert: *One mail mis-forwarded to the wrong recipient is worse than thousands of undelivered mails being returned back to the sender*.

Taking these principles into account we decided to inform the sender about incorrect email address and give him the list of *the best matched entries*. Only if the address is matched exactly and uniquely (in case that one part is missing or shortened) the mail is automatically forwarded to the sender (e.g. Matousek@cern.ch or P.Matousek@cern.ch would be rerouted to Petr.Matousek@cern.ch).

3 Fuzzy match techniques

As we mentioned before, *fuzzy match* means *the best possible match*. Fuzzy matching should correct every possible type of mistakes. In Table 1 a list of possible mistakes is specified.

mistake type	**example**	**correct name**
mis-typing	Mukkelson	Mikkelsen
mis-spelling	Iouri.Riabov	Yuri.Ryabov
swapping parts	Matousek.Petr	Petr.Matousek
using login names	enava	Elisa.Nava
initials	J.Smith	John.Smith
nicknames	Joe.Rothberg	Joseph.Rothberg
wrong delimiters	Nicola.Dambrosio	Nicola.D'Ambrosio
ambiguous names	F.Michaud	Frederic.Michaud, Jean-Francois.Michaud

Table 1: Common mistakes

In the following paragraphs we describe known techniques for text matching, their drawbacks, advantages and which of the specified mistakes they can cover.

3.1 String comparison

The simple solution is a searching an input name (a string) in the database. Using optimized algorithms like the KMP algorithm [3] or the Boyer-Moore algorithm [4] we can significantly speed up searching.

By string comparison (admitting regular expression matching in addition) we can correct problems of **swapping parts**, **wrong delimiters**, **initals** and **ambiguous names**.

The distance from a correct name is zero if matched. This does not mean we have found the best match (see Section *"Correction versus rerouting"*), unfortunately the exactly matched name can be wrong.

For example: Input *Abad* matches exactly user *Abad* but right recipient was user *Abada* with one mistake (deletion).

3.2 Approximate match based on spelling

For approximate matching we use the program **agrep**. Agrep searches a file for a string with approximate matching capabilities. It is fast and allows to specify the maximum number of errors for an approximate matching. Experimental results show that agrep is faster than e.g. grep, egrep etc. (see [1]).

Although it allows to set relative costs of mistakes (substitution, insertion, deletion), experiments give very similar results not depending on the cost. Cost setting may in particular cases mislead to imprecise matches.

Agrep is a fast tool; however, for short names it can be really slow and, moreover, if we permit several errors within a short string, it can match almost the entire database. For names shorter than 4 letters we don't allow any mistake for an approximate match (only an exact match is possible). Names having 4 letters are allowed to have one mistake, longer names can match database names with distances up to 2. *Allowing more mistakes will produce a lot of matches which have to be reduced by other techniques.* This would waste CPU time and slow down the fuzzy match.

Agrep can cover items of common mistakes not handled by string comparison (see Table 1) such as mistyping or partially misspelling errors. On the contrary agrep is not perfect because it can produce imprecise results.

For example: The name Kreiger approximately matches the names Krieger and Kreuzer both having the same distance 2. Of course, only the first one is correct. Nevertheless, agrep gives in many cases excellent results in comparison with other tools.

3.3 Approximate match based on sound

The approximate match based on sound uses the soundex algorithm as described by Donald Knuth in [5]. The algorithm is intended to hash words into a small space using a simple model that approximates the sound of the word when spoken by an English speaker.

Each word is reduced to a four-character string, the first character being and uppercase letter and the remaining three being digits. It groups names by sound rather than the spelling - see Table 2.

name	**code**
Euler, Ellery	E460
Gauss, Ghost	G500
Hilbert, Heilbronn	H416
Knuth, Kant	K530
Lloyd, Ladd	L300
Lukasiewicz, Lissajous	L222

Table 2: Example of Soundex coding

However, it has a few features which make it inefficient for approximate matching:

The soundex system defines a match only on names belonging to the same group (having the same code). Because it does not accept many common mistakes we re-defined the **soundex matching**. New rules for matching based on soundex algorithm follows:

feature	example
long names	ignores ending letters: Krizik, Krizikova = K622
names composed of several parts	code permutation required
not efficient for non-english spelling	Hrivnac, Harpman = H615
imprecise	Gauss,Ghosh = H615

Table 3: Soundex drawbacks

- The names match exactly if their *codes are equal* (original definition of soundex match).
- The names match exactly if their *codes differs in the first digit only* (E.g. uve = U100, uwe = U000, uffe = U100).
- The names match exactly if *two digits in their code are swapped*, no difference allowed (E.g. willjeman = W425, willemjan = W452).

The **soundex distance** is zero if *soundex matches exactly* or non-zero (infinitive) in any other case.

There are other sound-based system similar to soundex. A famous one is a metaphone that instead of coding by digits, uses letters. The metaphone algorithm does group names together more closely related than Soundex does. It has the same principle and consequently the same drawbacks as shown in Table 3.

Soundex is not very good for approximate matching but can be used for final refinement where it excludes wrong results – see the Krieger-Kreuzer example mentioned above.

4 Implementation of fuzzy matching

In the previous section we introduced fuzzy match techniques. Unfortunately none of these techniques gives as precise results as we would expect. In Table 4 pros and cons are summarized.

technique	pros	cons
string comparison	fast	not for approximate match
agrep	approximate match, fast	imprecise for misspelling
soundex	correction of mis-spelling	inprecise for mistyping, not for non-english languages, omits ending letters

Table 4: Fuzzy match techniques

Making various tests we can say that a combination of the described methods can lead to quite good results. It is hard to find an optimal algorithm; hence, we were searching for a balance between the closest match and a certain level of approximation permiting to recover corrupted addresses.

If we permit more mistakes it will repair more addresses but for really wrong input values we get incorrect results.

For example: Blabla matches user Blaha.

The following heuristic procedure for fuzzy matching was designed and implemented at CERN:

1. e-mail address alignment (drop unwanted characters - see [6])
2. separation of firstname/surname parts
3. exact match (string comparison) searching the database of e-mail users
4. approximate match (agrep)
   (a) preselection (more mistakes allowed)
   (b) correction (refinement)
5. soundex correction (soundex)

Generally, this heuristic procedure describes a possible way how to define knowledge that human (in our case a postman) uses for the name recognition and correction. Or we can say the procedure shows how to transfer human knowledge to a program.

5 Experimental results

Definition of parameters for heuristic name correction was not simple. Many of parameters were set according mistake types (see Table 1). By testing of hundred names we found values that satisfied our requirement - the system corrects as much mistakes as possible in a real time (timeout is set to sixty seconds) while short and incorrect inputs are rejected in a first decision phase.

Table 5 shows what types of incorrect names are expected and what is the result of correction (process by our heuristic algorithm).

input	**corrected input**	**input**	**corrected input**
Helmut.Schonbacker	Helmut.Schonbacher	Deszco.Horvathd	Dezco.Horvath
Helmuth.Schonbacher	Helmut.Schonbacher	DidierChanat	Didier.Chanat
Javier.Cuevas	Javier.Cuevas.Maestro	Duennweber	Wolfgang.Dunnweber
Kluas.Plunkte.	Klaus.Pluntke	E.Katsoufi	Elias.Katsoufis
Gandi.Angelo	Roland.d'Angelo	Gennadi.Zinovyev	Gennady Zinovjev
Igor.Tyapkine	Igor.Tiapkin	Julian.Budagov	Ioulian.Boudagov
K.Chevreux.Thomas	Katharine.Thomas-Chevreaux	Susan.von-Wartburg	Suzanne.Von.Wartburg
d.oneil	Dugan.O'Neil	giancarlo.trinchero	Gian.Carlo.Trinchero

Table 5: Example of name correction

Neither by agrep algorithm nor by soundex algorithm we can obtain such results. Agrep algorithm correct many of above mentioned mistakes, for others it needs additional knowledge.

Soundex algorithm was designed in order to group names by their sound pronounced by english speaker. For other languages it gives inprecise output. Even for english its classification is not precise as shown in Table 2.

6 Conclusion

Proposed heuristic approach is an example which describes and exploits human knowledge for name recognition. It combines well-known techniques and approaches. Experimental results shows that our results are better in comparison with other algorithms.

Implementation for email addresses resolution gives an practical example of our approach. While e-mail address correction done “by hand” is unthinkable, we need an automatic technique. Fuzzy matching using string comparison and approximate matching based on sound and spelling makes the solution feasible. Of course we can never reach optimal correction. The system described above shows one possible way. Usage on real systems gives good results and reduces the number of messages refused because of incorrect addresses.

Acknowledgements

This project was performed at IT/DIS/MIS section at CERN. Thanks belongs to outstanding mail expert Dietrich Wiegand for his valuable advices and co-operation. The author also appreciate suggestion and comments of Arnaud Taddei and Judy Richards that improved several features of the program.

References

[1] Sun Wun, Udi Manber: Fast Text Searching With Errors. Dpt. of Computer Science, University of Arizona, Tuscon, June 1991.
[2] What is Soundex ? Infobahn Outfitter, Inc, 1997 (http://www.outfitter.com/genealogy/what-soundex.html).
[3] Boyer R.S., and J.S.Moore: A fast string searching algorithm, Communications of the ACM, 20 (October 1977).
[4] Knuth D.E., J.H.Morris, and V.R.Pratt: Fast pattern matching in strings, SIAM Journal on Computing, 6 (June 1977).
[5] Knuth D.E., The Art of Computer Programming, Volume 3, Mass.Addison-Wesley 1973.
[6] David H.Crocker: Standard for the format of ARPA Internet text messages, RFC 822, 1982.