# Error-tolerant finite-state lookup for trademark search

Andreas Eisele[1] and Tim vor der Brück[2]

[1] Computational Linguistics, Saarland University, D-66123 Saarbrücken
eisele@coli.uni-sb.de
[2] German Meteorological Service, Kaiserleistraße 35, D-63067 Offenbach
tim.brueck-vor-der@dwd.de

**Abstract.** Error-tolerant lookup of words in large vocabularies has many potential uses, both within and beyond natural language processing (NLP). This work[3] describes a generic library for finite-state-based lexical lookup, originally designed for NLP-related applications, that can be adapted to application-specific error metrics. We show how this tool can be used for searching existing trademarks in a database, using orthographic and phonetic similarity. We sketch a prototypical implementation of a trademark search engine and show results of a preliminary evaluation of this system.

## 1 Introduction

Many applications of NLP have to deal with some kind of deviation of observed input from the theoretically correct form. Sources for such deviations may include human performance (typing) and competence (spelling) errors as well as technical inaccuracies of transmission and recognition, especially in cases involving speech or character recognition. An ideal NLP system should be able to guess a correction of deviating parts of the input and find an interpretation that is compatible with the user's intent.

Besides correction of errors, there are many other applications where the focus is to find terms that are similar to a given query. These include retrieval of person or product names from databases, or documents from text repositories[4] as well as cross-lingual applications, e.g. for finding likely transcriptions of names in different writing systems or for the alignment of words and phrases in translation memories.

The concept of similarity depends strongly on the input modalities and other properties of the application, as will be explained in more detail in Sect. 4. In

[3] The work described here was done while the authors were at Language Technology Lab, German Research Center for Artificial Intelligence (DFKI GmbH), Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany

[4] See http://www.google.com/jobs/britney.html for a list of 593 query types for which the google search engine proposed "Britney Spears" as correction. More than 23% of the queries were misspelled.

order to be able to address the wide range of applications in a generic way, we implemented a library, called SILO, for error-tolerant or similarity-based lookup from a set of strings which is given in the form of a finite-state device.

This papers focuses on a innovative use of SILO in a specialized search engine for trademarks, where the goal is to test whether a potential brand or product name conflicts with existing trademarks that look or sound similar.

## 2 Requirements for trademark search

The main purpose of trademark law is to protect the public from being confused or deceived about the origin and quality of a product. This is accomplished by the trademark owner preventing competitors from using a mark that the consuming public is likely to confuse with theirs, whether because it is identical (such as another computer manufacturer calling themselves "Apple") or sufficiently similar (such as a soft drink called "Popsi", to mimic "Pepsi"). See [Wil98] for an easy-to-read introduction to the subject.

When a company registers a new trademark at one of the national or international authorities, it must check whether this or a similar trademark already exists in the same industry sector[5]. Infringement of an existing trademark may cause very high costs, including the need to destroy products already manufactured, or to suspend expensive marketing campaigns. Manual search for existing trademarks has several obvious disadvantages, including the high cost of manual labour, potential incompleteness of the results and the need to redo the search after a change in the database. Therefore an automatic approach would be much preferable. Trademark search needs to be aware of several types of similarity, which can be character-based, phonetic or semantic. The most obvious form of similarity between two terms consists in the use of letters in a similar order. Especially the insertion or omission of repeated characters is very hard to notice (compare Alibert vs. Allibert) and therefore leads to a high confusability of the names.

Two trademarks can be pronounced similarly even if the orthography is quite different. For instance, the German optician Fielmann could be spelled as Viehlman, Philmahn, etc. without noticeable difference in pronunciation. Given our machinery for error-tolerant lookup, one might be tempted to translate knowledge about phonetic similarity between strings of characters into substitutions for the error metric. However, the large resulting number of substitutions would be inconvenient to specify and would make lookup unnecessarily slow. Instead, it appears much more adequate to transcribe strings into phonetic representations and determine similarity scores on that level. Fortunately, we could make use of existing software from the MARY project [ST01] for the transcription into phonetic representations, assuming phonetic rules for German or

[5] This is in contrast to the case of patents, where the authorities are responsible for the evaluation of novelty before registration.

English.[6] We can then look for similarities on the phonetic level, using a much smaller set of possible confusions. Current leading trademark search engines like Eucor [EUC04] or Compumark [Com04] also use orthographic and phonetical similarity. However, technical information about the inner workings of these engines is rather sparse.

Words can also be semantically similar, i.e. they can be near synonyms, one can be a special case of the other, or they can be translations of each other. Although one could try to tackle semantic similarity with a resource like Wordnet, coverage may not be sufficient for the very large vocabulary used in the trademark domain. The current implementation does not take semantic similarity into account.

The proper treatment of trademarks composed of multiple words requires the assignment of weights to the constituents. Ideally, one would like to give a high weight to specific words and names, whereas generic or descriptive parts of the names should have a lower weight. Attempts to infer good weights from the frequencies of words in the database have had mixed results so far, so the current implementation supports the interactive specification of weights for the parts of multi-word trademarks.

## 3 Finite-state-based lexical lookup

Our implementation is based on a generic toolkit that aims at a much broader scale of tasks related to robust and multilingual natural language processing. Especially when languages are richly inflected or involve compositional morphology, a full enumeration of all possible word forms gives rise to prohibitive storage requirements or is plainly impossible.

In order to be useful even in these cases, the implementation of the lookup is based on a representation of the relevant data in the form of finite-state acceptors (FSA) or transducers (FST) in the style of [KK94] and [Kar94]. FSAs can be seen as improved versions of letter trees or tries [Fre60] to regular languages [HU79], in which the possibility to share trees of suffixes can make the representation exponentially more compact and even encode infinite vocabularies. FSTs describe regular relations between strings in a declarative way. Their mathematical properties are simple and well understood, and they can be used for generation of surface forms as easily as for morphological analysis. They generalize finite-state acceptors to multiple tapes, where transitions between states simultaneously affect several levels of string representation. Applications for morphological lookup use one of these tapes for the surface string, and another for the underlying linguistic analysis. Since FSTs factor out independent sources of variation, exponential or infinite numbers of forms with all morphological

[6] A proper assessment of trademark similarity in a multilingual context needs to address potential confusions of many different types, such as: "Does word X (which is really from language Y), when pronounced in language Z, sound similar to word U (which is really from language V),...", which is further complicated by neologisms and trademarks composed from multilingual pieces.

readings can be represented very compactly. The clear separation between compilation of the FST and transduction of strings nicely accommodates different requirements for off-line and on-line processing of the linguistic specification and makes it easy to embed morphological processors for different natural languages into larger systems.

The compiled FST representations can then be interpreted by existing implementations of exact and error-tolerant finite-state lookup implemented at the German Research Center for Artificial Intelligence (DFKI). In order to accommodate different requirements for functionality, speed and compatibility, there are two implementations of the lookup routine that can work with the same binary representations. The Java implementation focuses on simplicity and reliability and can be included in multi-threaded applications. The C implementation is significantly faster and allows for the search of a set of most similar strings under a given distance metric, as described below. Programming interfaces to several host languages exist. Using this generic framework, robust morphological analysis and generation can be embedded quite flexibly into various platforms for NLP and into other applications.

## 4 Lookup with application-specific error tolerance

The concept of similarity most suitable for robust lookup and error correction depends strongly on the sources of deviations such as the modalities of textual input. If text is typed, confusions of keys nearby on the keyboard are much more likely than others, and the keyboard layout also influences the likelihood of swapping adjacent characters. If text is decoded from document images by an OCR system, similarities in visual appearance are important; typical confusions involve the sets `{I,l,1}`, `{e,c}`, `{m, rn}`, `{d,cl}`, and so on. Phonetic similarity plays a crucial role both for the correction of spelling errors and of errors in the result of automated speech recognition.

For the application in trademark search, the latter two kinds of similarity are also immediately applicable, as will be explained below. Keyboard layout, however, does not matter in this case.

As we want to support cases where the vocabulary is specified as a finite-state device, we needed a way to incorporate application-specific error tolerance into a finite-state lookup algorithm. Such mechanisms have been described in the literature [OG94,Ofl96,SM01], but these approaches are based on uniform costs for all kinds of errors, i.e. the error model is built into the search algorithm and cannot be parametrized according to the needs of the application. [KCG90,BM00] describe the use of application-specific probability distributions for the correction of typing and spelling errors, but the method they give for efficient lookup does not seem to be immediately applicable to infinite vocabularies, encoded in cyclic FSMs. Our error-tolerant lookup follows roughly the approach of [Ofl96], but furthermore allows to specify the likelihood of deviations (such as typing/spelling/OCR errors or phonetic similarity) in an application-specific error metric.

Error metrics consist of parameters that specify the cost of generic edit operations such as deletion, reversal, substitution, and insertion, where the latter can be further differentiated according to the place of the insertion (initial vs. inner vs. final position). Furthermore, we allow to list specific substitutions in the form of 4-tuples $\langle w_i, w_{d_i}, w_{c_i}, s\rangle$ with $w_i, w_{d_i}, w_{c_i} \in \Sigma^*, s \in R^+$, where $\Sigma$ is the set of characters, $w_i$ and $w_{d_i}$ stand for possible substrings of the query and the target string that are matched with error cost $s$, and where $w_{c_i}$ specifies a (potentially empty) left context, to which this replacement is constrained. Whereas this definition may not satisfy all wishes for generality or elegance, it has proven flexible enough to deal with many important classes of phenomena. For the trademark search, the possibility to make edit costs dependent on the position within a word and to penalize the insertion of whitespace that would break words have proven especially useful.

The lookup happens in two steps. In a preprocessing step, substrings of the given query are enriched with potential alternatives according to the list of 4-tuples, whenever the left context given in the substitution matches. Conceptually, this transforms the given query into a weighted graph, specifying a set of variants of the query modulo the substitutions. The main lookup routine now performs a backtracking search for compatible paths through the expanded query graph and the lexicon FSM, where the use of generic edit operations is taken into account as well as the possibility and costs of picking variants of the query.

While traversing the pair of FSMs, costs for the edit operations or variant branches are cumulated, and branches that exceed a given upper limit (tolerance) are abandoned. In this way, the backtracking search enumerates all matches that are possible within the initial tolerance. This limit is increased by iterative deepening, until a pre-specified minimal number of matches has been found.

## 5 Trademark search algorithm

The following section specifies the principles discussed in the previous section in a more formal and detailed manner.

### 5.1 Similarity comparison

The lexicon which is used for similarity comparison is stored in a finite state machine, which is a popular, efficient and memory-saving mechanism to store large numbers of strings. In order to simplify the presentation and because it does not make a difference for the purpose of the paper, we restrict the examples to the special case of letter trees, such as in Fig. 1, which are finite state machines where each node has maximally one ingoing arc. The word entries of the lexicon are the concatenation of the letters from the tree root to a leaf node.

In the example in Fig. 1 the lexicon would consists of the words

- ab
- au

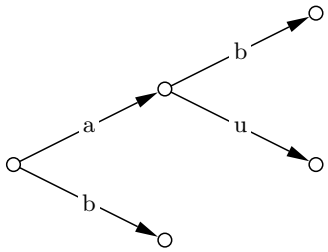


**Fig. 1.** letter tree

– b

The input word consists of letters of some alphabet $\Sigma$ such that $w \in \Sigma^*$. The similarity algorithm now returns all entries of the lexicon $d$ for which similarity to some input $w$ is below some given threshold $t$.

$$similar\_words_d(w,t) = \{(w_d, sim) | w_d \in d \wedge sim_{total}(w_d, w) < t\}$$

This is done by recursively examining all routes of the tree from the root until it reaches a leaf node or the similarity is already higher than the given threshold using a depth-first search. The search can be aborted if the threshold is exceeded since the cost (similarity value) of choosing some arc depending of some letter of the input $w$ is always positive or zero. The similarity function is defined as follows

$$sim : \Sigma^* \times \Sigma^* \times \Sigma^* \rightarrow \Re_{+0} : sim(w_i, w_{d_i}, w_{c_i}) = s$$

with

– $w_i$: some part of the input
– $w_{d_i}$: concatenations of arcs in the tree
– $w_{c_i}$: context: some part of the input directly in front of $w_i$:
  $w = aw_{c_i}w_ib$ where $a, b \in \Sigma^*$.

This function is partial and does not have to be defined for all possible input words. However it should at least be defined for all $w_i, w_{d_i} \in \Sigma$. The similarity for the whole word is the sum of the similarities for the word parts.

$$sim_{total}(w, w_d) := \min\{\sum_{i=1}^{n} sim(w_i, w_{d_i}, w_{c_i}) \mid w_d = w_{d_1} \bullet ... \bullet w_{d_n} \in d \wedge w = w_1 \bullet ... \bullet w_n \wedge w = v \bullet w_{c_i} \bullet w_i \bullet ... \bullet w_n\}$$

This basic algorithm is already quite useful for similarity determination. But there are also some cases where no comparison would be possible. Consider the input string "Pateck Phillip" and the entry "Patek Phillip" in the database. The algorithm described above could only determine the similarity between both strings if the value of $sim(ck, k, \varepsilon)$ (or alternatively $sim(c, \varepsilon, \varepsilon)$) would be explicitly defined.

Therefore additionally to the algorithm stated above we introduce the possibility to leave out one character of the input string without changing the state in the tree. Furthermore we have to define how much the costs determining the similarity value between the input string and a tree node is raised by this action. Especially in connection with trademark names it makes a difference whether characters are left out at the beginning, in the middle or at the end of the input sentence e.g. "Gosun" would be considered less similar to the mark name "SUN" than "Sungo".

Similar to this case it can also be useful to change the state in the tree without going to another character in the input sentence. Generally we can define a second similarity function as follows

$$sim' : Op \times Loc \rightarrow \Re_{0+}$$

where

$$Op \in \{Jump, Stay\}, Loc \in \{Beginning, Middle, End\} \ .$$

So the first similarity function $sim$ can be extended in the following way:

$$\begin{aligned} sim(w, \varepsilon, w_c, loc) &= sim'(Stay, loc) \\ sim(\varepsilon, w_d, w_c, loc) &= sim'(Jump, loc) \ \forall w, w_d, w_c \in \Sigma^* \end{aligned}$$

Note that the parameter list of $sim_{total}$ has to be extended for an additional parameter indicating the current location inside the query term.

### 5.2 Weighted Merge

The proposed algorithm tends to return too small similarity values when used for trademarks which consists of several words. Consider the trademark SUN and the two strings "Gesund" and "Ge sun d". Although the latter two strings both contain the word sun and are written using the same letters in the same sequence only the last one would conflict with the trademark SUN because by using a blanks before and after the substring "sun" it becomes obvious that "sun" is part of "Ge sun d".

Therefore the base algorithm was extended for multi-word trademarks to compare not only the whole expression but also every word separately with the words from the database. Now we use the individual rankings of the word for word comparison to compute a global ranking. To do this we use the condition

that if every word-for-word similarity calculation for some search string would result in the same similarity value then the global similarity should be equal to this value.

Additionally there should be a possibility to use weights for the words of the input strings, e.g. in the upper example string "ge sun d", "sun" could be weighted more (or less) important than the token "ge".

Let $sim_i$ denote the similarity of the $i$'th token with some word $w_d$ from the database. $sim_i$ is constraint from 0 (total identical) to $\infty$ (maximal difference) Let $w_i$ denote the weight of the $i$'th token in the input string $w$. For better computation we first shift the rating $sim_i$ in the interval from 0 (least similar) to 1 (identical). The conversion formula is easily given by $sim'_i = 1/(sim_i + 1)$. The global similarity is calculated by

$$g = \Big( \sum_{i=1}^{\sharp Tokens(s)} w_i * sim'_i \Big) / \sum_{i=1}^{\sharp Tokens(s)} w_i \ .$$

Afterwards the similarity must be reconverted to the scale from 0 to infinity which is done by: $g' = (1/g) - 1$.

### 5.3 Automatically determined weights

If some query term consists of more than one word, weights can be assigned to each singular word. Sometimes such multi-word expressions includes class names (e.g. "Cafe" in "Cafe Karlo") or corporation abbreviations like ("Deutsche Telekom AG" or "IBM Corp."). A different trademark should not be considered similar to IBM Corp. only because the other company trademark also includes the word "Corp". One approach we followed here is doing a frequency analysis to count how often a word appears. The probability is high that a word like 'AG', 'GmbH' or 'Corp' appears quite often. So if a word is appearing often, the weight used for this word should be low (near to zero). Unfortunately, this approach turned out to be too simple and leads to wrong results in some situations. So it is assumed that also the position of the word should be considered, so 'AG' or 'Corp' should only be weighted low if it appears at the end of the expression.

## 6 Implementation and first results

The program for trademark comparison is implemented as a web server application with Java Server Pages and Servlets. The trademarks are stored in a MySQL database and are accessed via JDBC. The user can chose between different views, including a compact view with only one score per result or a detailed view where scores of textual similarity, phonetical similarity for English and German and weighted combinations of metrics are given. Hyperlinks allow to obtain more detailed information on each trademark in a separate frame (see Fig. 2 and 3).

During the construction of the trademark database, phonetic transcriptions of the trademarks are generated according to German and English pronunciation

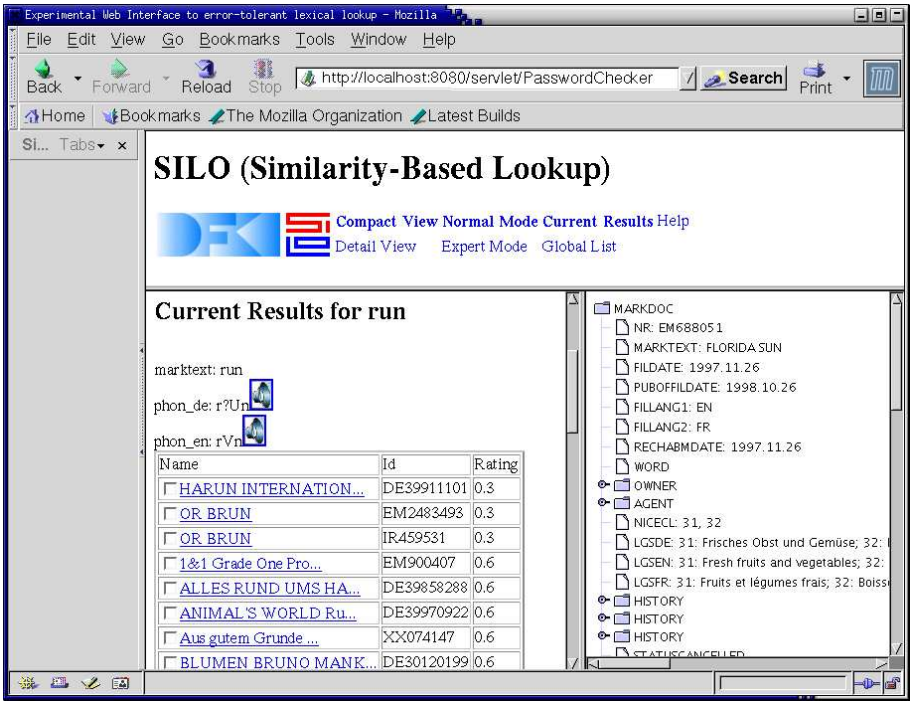


**Fig. 2.** Silo

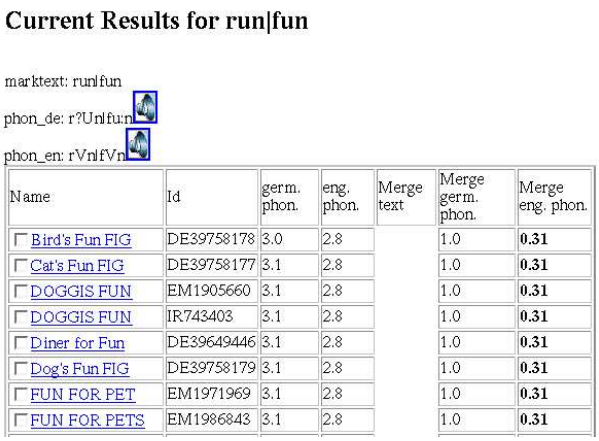


**Fig. 3.** Silo - Detailed View

rules[7], and the resulting strings are stored in additional fields of the trademark database. Finite-state encodings are then generated both for the relevant orthographic and phonetic representations found in the database.

When a query is entered into the system, transcriptions are generated for the query term in all relevant languages. For the set of representations obtained in this way, similarity searches are performed in the respective parts of the database, and the results are merged into one unified ranking according to the lowest distance across all types of similarities that are computed.

The system also includes an interface to the MBROLA-based speech synthesizer of the Mary system[ST01]. This was motivated by imperfections in the phonetic transcription for unknown words, which however constitute an important subset of the trademark vocabulary. As the system can provide audible feedback, users who are not familiar with the encoding of the phonetic transcription are still able to spot mistakes in the automatically computed transcriptions of the query, and can alleviate these errors by entering "hints" in the form of variants of the query that lead to pronunciations that are closer to the intended outcome.

A first round of evaluation has been performed by BOEHMERT & BOEHMERT, a law firm specialized in trademarks and intellectual property rights from which this project originated. Results of the search engine were compared with the outcome of manual searches and of search engines by commercial service providers. The recall of our implementation turned out to be very promising, compared to the alternatives. The precision of the results was a bit lower than that of manual search agents, which apparently make use of certain "intuitive" notions of relevance that are difficult to capture formally. However, so far the distance metrics used in the experiments have been rather simple, and we hope to further improve the precision (i.e. shorten the result lists without losing too many relevant hits) by a somewhat more careful design of the metrics. More details of the evaluation are given in the following section.

## 7 Evaluation

Section 7.1 gives an evaluation of the basic algorithm which shows how the system performed with respect to precision, recall and f-measure, which trademarks were missing, and compares this with the performance of a human researcher, and also gives an interpretation of the results. Section 7.2 shows results of the weighted merge extension to the algorithm, which we did ourselves and also discusses these findings.

[7] Transcriptions for more languages are under preparation

### 7.1 Results of the basic algorithm

*QUERY STRING: PERFECT FIT*

| Value | Silo | human researcher |
|---|---|---|
| Precision | 0.02 | 0.125 |
| Recall | 1.0 | 1.0 |
| F-Value | 0.03 | 0.22 |

*QUERY STRING: CREMISSIMA*

| Value | Silo | human researcher |
|---|---|---|
| Precision | 0.31 | 0,5 |
| Recall | 1.0 | 0.33 |
| F-Value | 0.47 | 0.19 |
| Missing trademark | - | KäsEmilia<br>Carnissimo<br>PETISSIMO<br>ledissimo |

*QUERY STRING: LAITANA*

| Value | Silo | human researcher |
|---|---|---|
| Precision | 0.06 | 0.75 |
| Recall | 0.5 | 0.5 |
| F-Value | 0.1 | 0.6 |
| Missing trademarks | Lactina<br>Lacsana | MULTANA<br>altina |

*QUERY STRING: CURLIES*

| Value | Silo | human researcher |
|---|---|---|
| Precision | 0.2 | - |
| Recall | 0.42 | - |
| F-Value | 0.15 | - |
| Missing trademarks | JERKIES<br>URVIS<br>Cultaris<br>LIS<br>Lis<br>ULIS<br>FORLYSE<br>CORALISE<br>BURGYS<br>CHRYSALIS<br>CEREALIS | |

These examples show that SILO had quite good results in the recall but sometimes poor results in precision. The fact that precision scores are not very good is based partly on technical issues. SILO requires the user to select the number of trademarks it should return and always returns that number of trademarks even if only one single trademark in the database is really similar. To overcome this problem one could only select trademarks up to a predefined maximal distance. But such a limit would have to be determined using psychological experiments to determine the level of similarity people consider as significant for this application. Such data could in principle be collected using a suitable extension of the system for collecting user feed-back. However, such a module has not yet been implemented.

One can further see in the statistics that SILO sometimes does not recognize words which appear in the middle of some trademark in the database. This problem could be removed by finetuning or automatically learning the costs for leaving out letters at the beginning or at the end.

### 7.2 Evaluation of Weighted Merge

This section describes how the evaluation for the "weighted merge"-algorithm was done. First the 50 most similar words to the given query were retrieved using both the "weighted merge" and the basic algorithm. Now we subjectively selected from this set the 10 words most similar to the query and compared how many of them were contained in the 10 topmost ranking entries of the respective result lists found by either algorithm. To ensure that the selection was not influenced by the initial ranking within the result lists, these lists were first shuffled, which made sure that the similarity rating the algorithms assigned to the terms were not visible. Since the number of terms found by the algorithm used for the evaluation and of the manually selected terms are both 10, recall, precision and f-measure all have the same value. Therefore only the value of the recall is given in the tables.

*QUERY STRING: High Meyer*

| Value | Normal | Weighted Merge |
|---|---|---|
| Recall | 0.1 | 0.1 |
| Missing trademarks | HAI | HAI |
| | PRO-FIT Meyer | PRO-FIT Meyer |
| | MEYER FIG | MEYER FIG |
| | Heinrich Liesmeyer | Heinrich Liesmeyer |
| | Metzger Meier Fleischermeister | Metzger Meier Fleischermeister |
| | KEINE FEIER OHNE MEYER | KEINE FEIER OHNE MEYER |
| | High Tech mit gutem Gewissen | High Tech mit gutem Gewissen |
| | hy | hy |
| | Herrman Meyer | Herrman Meyer |

*QUERY STRING: Maut Champion*

| Value | Normal | Weighted Merge |
|---|---|---|
| Recall | 0.7 | 0.7 |
| Missing trademarks | HAI | HAI |
| | Frolic Champion FIG | Frolic Champion FIG |
| | Mr. Champ | Mr. Champ |
| | UEFA European Football Champion | UEFA European Football Champion |

*QUERY STRING: Power Flip*

| Value | Normal | Weighted Merge |
|---|---|---|
| Recall | 0.3 | 0.2 |
| Missing trademarks | Flip | PowerVit |
| | Preis Power FIG | Preis Power FIG |
| | STOPY FLIPS | STOPY FLIPS |
| | UEFA European Football Champion | UEFA European Football Champion |
| | POWER | POWER |
| | NP National Power FIG | power POOL FIG |
| | Power Pur FIG | Power Pur FIG |
| | | PowerFlakes |

This experiment shows that the "weighted merge"-algorithm can retrieve terms that would not be found using only the standard algorithm (e.g. "Flip" for the query term "Power Flip". Actually this term does not even appear at the first 50 terms found by the ordinary algorithm)). The "Weighted Merge"-algorithm is especially useful if only one word of the query term is appearing in the database. On the other hand the overall performance of the "weighted merge"-algorithm is not significantly better than that of the ordinary algorithm. We expect that the results can be improved by additional fine-tuning of the weighting scheme.

## 8 Other applications of SILO

The advantages of FST-based morphology motivated the transformation of existing resources for 5 European languages [PR95,BLMP98] into FST representations, where e.g. over 6.5 million different analyses of German full forms could be represented in only 1.2 MB. In the case of a resource currently under construction for Arabic [SE04], the restoration of missing vowels, tightly integrated into the lookup operation, comes as a free extra feature without additional implementation work.[8]

[8] As long as we want to have *all* readings. Selecting the *correct* interpretation in ambiguous cases is of course much more difficult.

Using the generic framework on which SILO is built, robust morphological analysis and generation can also be embedded quite flexibly into various platforms for NLP. Several design studies and demonstrators have been built, including on-line correction of typing errors in a dialogue system, OCR correction for financial documents transmitted via FAX, and the correction of typing errors in a database of job offers. Current activities include the integration of SILO into the platforms LKB [Cop01] or PET [Cal00] for deep, HPSG-based syntactic analysis and generation, which are used in the project Deep Thought [CESS04], and many related activities.

## 9 Conclusion and outlook

We have shown how a generic toolkit for similarity-based lookup of strings in finite-state devices can be used for searching existing trademarks in a database, using orthographic and phonetic similarity. First evaluation results of the system look quite promising, but a lot of possibilities for fine-tuning the parameters of the system have not yet been explored. In order to support this kind of optimization, it would be very helpful if judgements of users about relevance or similarity of the proposed trademarks could be collected in a lightweight, semi-automatic way, so that machine learning methods could then turn the collected judgements into optimal weights for a significant set of tunable parameters.

## 10 Acknowledgements

We would like to thank Dr. Detmar Schäfer from the law firm BOEHMERT & BOEHMERT for the original idea, development data, and for valuable feed-back, Marc Schröder for the transcription of words into phonetic representations and many helpful advices, Stephan Busemann for keeping the project going and a tremendous amount of supporting work, and an anonymous reviewer for spotting errors in the draft version of this paper.

## References

[BLMP98] P. Bouillon, S. Lehmann, S. Manzi, and D. Petitpierre. Développement de lexiques à grande échelle. In *Actes du Colloque des journées LTT de TUNIS*, pages 71–80, 1998.

[BM00] E. Brill and R. C. Moore. An improved error model for noisy channel spelling correction. In *Proceedings of the ACL*. ACL, 2000.

[Cal00] U. Callmeier. PET — A platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering*, 6 (1):99 – 108, 2000.

[CESS04] Ulrich Callmeier, Andreas Eisele, Ulrich Schäfer, and Melanie Siegel. The DeepThought core architecture framework. In *Proceedings of LREC*, Lisbon, Portugal, 2004.

[Com04] Compumark, 2004. http://www.compumark.com.

[Cop01] A. Copestake. *Implementing Typed Feature Structure Grammars*. CSLI Lecture Notes. Center for the Study of Language and Information, Stanford, 2001.

[EUC04] Eucor, 2004. http://www.eucor.de.

[Fre60] E. Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

[HA01] V. J. Hodge and J. Austin. An evaluation of phonetic spell checkers. Technical report, Department of Computer Science, University of York, 2001. Technical report YCS 338.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.

[Kar94] L. Karttunen. Constructing lexical transducers. In *COLING-94*, pages 406–411, Kyoto, Japan, 1994.

[KCG90] M. D. Kernighan, K. W. Church, and W. A. Gale. A spelling correction program base on a noisy channel model. In *COLING-90*, volume II, pages 205–211, Helsinki, 1990.

[KK94] R. M. Kaplan and M. Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.

[Ofl96] K. Oflazer. Error-tolerant finite state recognition with applications to morphological analysis and spelling correction. *Computational Linguistics*, 22(1), 1996.

[OG94] K. Oflazer and C. Güzey. Spelling correction in agglutinative languages. In *4th ACL Conference on Applied NLP, Stuttgart*, Stuttgart, Germany, 1994. Association for Computational Linguistics.

[PR95] D. Petitpierre and G. Russell. MMORPH - the Multext morphology program. Technical report, ISSCO, CH-1227 Carouge, Switzerland, October 1995.

[SE04] A. Soudi and A. Eisele. Generating an Arabic full-form lexicon for bidirectional morphology lookup. In *Proceedings of LREC*, Lisbon, Portugal, 2004.

[SM01] K. U. Schulz and S. Mihov. Fast string correction with Levenshtein-automata. Technical report, CIS, Universität München, 2001. CIS-Bericht-01-127.

[ST01] M. Schröder and J. Trouvain. The german text-to-speech synthesis system MARY: A tool for research, development and teaching. In *4th ISCA Workshop on Speech Synthesis*, Blair Atholl, Scotland, 2001.

[Wil98] L. Wilson. *The Trademark Guide*. Allworth Press, New York, 1998.

[ZD95] J. Zobel and P. Dart. Finding approximate matches in large lexicons. *Software - Practice & Experience*, 25(3):331–345, March 1995.