Fault-tolerant Fulltext Search for Large Multilingual Scientific Text Corpora

Wolfram M. Esser
Chair of Computer Science II
University of Würzburg
Am Hubland, 97074 Würzburg, Germany
+49-(0)931-888-6601
eMail: esser@informatik.uni-wuerzburg.de

Abstract

In the work reported here, we present a new way of performing fault-tolerant fulltext retrieval on large text corpora, such as scientific encyclopedias. The *weighted pattern morphing* (WPM) technique introduced in this paper overcomes disadvantages of both the popular edit distance measure and the Soundex code approaches, yet keeps their flexibility. This algorithm handles phonetic similarities; common typing errors such as omission or transposition of letters, and inconsistent usage of abbreviations and hyphenation. After showing how WPM can be implemented efficiently, we present a novel method of how the weights of the internal penalty matrix can be automatically adjusted for better results. Though the described technique can be applied without prior knowledge of actual user patterns, re-examination with a large number of online-user's patterns proves the portability of this fine-tuning approach. We further show how shifting the penalty matrix from one language to another can be accomplished.
The described WPM technique is integrated into a large commercial pharmaceutic encyclopedia CDROM, an online dermatological encyclopedia, and an online-reference encyclopedia of parasitology research, thus also proving its "road capability".

Keywords

Fuzzy matching, approximate information retrieval, fault-tolerant fulltext search, q-gram, n-gram, weighted pattern morphing, WPM, multilingual retrieval

1. Introduction

Users of CD-based, or online-encyclopedias, often face problems which arise from the fact these text corpora are written by a vast number of contributing authors. Although thoroughly edited, multiple (correct) spelling variants of scientific terms in these electronic text corpora remain in the final release. Some examples of these variants are multiple language terms with Greek, Latin or localized spelling; different application of hyphens, and inconsistent usage of abbreviations. Typical typographical errors such as omissions, inserting, and swapping of letters further complicate information recall from these texts by means of a fulltext retrieval system. In addition, many users of scientific texts often only know the pronunciation of a specific term, but not necessarily the correct spelling.

The first implementation of our fault-tolerant approach was during the compilation of the annual electronic version of *Hagers Handbuch der Pharmazeutischen Praxis* (Hager's Handbook of Pharmaceutic Practice) – the standard encyclopedia for German-speaking pharmacists and pharmacologists – in collaboration with scientific publisher Springer Verlag. The printed version of "Hager's Handbook" consists of twelve volumes totaling about 12,300 pages, and written by over 600 contributing authors. The current third electronic version, *Hager-ROM 2003* (Blaschek et al. 2003) was released in June of 2003. In this CDROM-based version, an approximate 200 MB XHTML text led to a 61 MB raw text, after layout tag removal.

To make this amount of information accessible to the end-user, a *q*-gram based fulltext retrieval system was built into the reading software. This q-gram search engine served as a fast, non fault-tolerant back-end for the WPM based front-end, where the front-end adds optional fault-tolerance to the retrieval system. Because of the WPM architecture, and the separation of front-end and back-end, this approach may add fault-tolerance to almost any non fault-tolerant text retrieval system like databases, or web search engines.

Further applications of the fault-tolerant approach are: fulltext retrieval systems of P. Altmeyer's "*Springer Enzyklopädie Dermatologie, Allergologie, Umweltmedizin*" (Springer's Encyclopedia of Dermatology, Allergology and Environmental Medicine), where the online version provides free access for physicians and medical students on (Altmeyer & Bacharach-Buhles 2002). The adaptation of WPM to an English text corpus was performed while integrating the technique into the combined online version of Springer's *Encyclopedic Reference of Parasitology* and the *Parasitology Reference Journal*.

In section 2 of this paper an overview is given of related work, and the different approaches based on phonetic codes and edit distance measures by other research groups. The disadvantages of these commonly used techniques are discussed, and we show how these can be circumvented with our new method. In section 3 we introduce our algorithm of weighted pattern morphing (WPM), which is, essentially, a matrix-driven, recursive replacement of substrings of the original user pattern.

Since the backbone of the WPM algorithm is the content of the so-called penalty weight matrices, we focus in section 4 on the description of the means of defining

Published 2004-10-06 in JoDI,
Journal of Digital Information (ISSN: 1368-7506)

rules and weights for these matrices. Finally, in section 5, we discuss some usage examples within the context of a large scientific encyclopedia.

2. Related Work

The paper we present here is taking our previous work presented in (Esser 2004) some important steps further: While in our ECIR'04 paper we wanted to find out reasonable base parameter settings for our algorithm (e.g., total number of alterations of the start pattern), in this paper we focus on how to adjust the hundreds of rule weights automatically – even without knowledge of actual future user search patterns. After improving the rule weights in this way we prove the better quality of these new weights with real user patterns. Further, in this paper we have examined the usability of our approach on a large English text corpus.

In this section we will summarize different approaches that lead to fault-tolerant – or fuzzy – search algorithms. The first search algorithms generated a canonical phonetic code for every word: if a searchpattern and a target word have the same code, they are regarded as phonetically similar. In contrast to this sound oriented measure exists a class of approaches which is more spelling oriented. These approaches are grouped around V. Levenshtein's edit distance measure.

2.1 Phonetic Codes

The now famous Soundex algorithm, patented in 1918 by Robert Russel, was intended to *"provide an index wherein names are entered and grouped phonetically rather than according to the alphabetical construction of the names"*((Russel 1918), and see (Zobel & Dart 1995) for details). Although this relatively simple phonetic approach has been the subject of widespread criticism, it has also triggered numerous improved successors, such as: Gadd's Phonex (Gadd 1990), Philips' Metaphone and Double Metaphone (Philips 2000), and Phonetex (Hodge & Austin 2001b), to name but a few.

There exist, however, hybrid algorithms such as Editext, and Idapist (Zobel & Dart 1996) that combine the edit distance measure's excellent capacity to correct spelling mistakes (see below), with the ability of phonetic codes to group equal sounding word variants.

However, all these approaches work only for word lists where codes for every single word are generated during preprocessing. The code for the user pattern is generated afterwards and then compared to the pre-calculated word-codes. These techniques have been successful for fuzzy surname matching, and for approximate search word lists drawn from larger texts.

Yet with this class of algorithms, it is impossible to rank strings that have the same code: strings are either similar (same code) or not (different code). The German "`sch`", for instance, sounds like the English "`sh`", however, the Soundex algorithm cannot manage multi-character strings as its code groups consist only of single letters.

Furthermore, phonetic codes are not applicable if one wants to perform a fulltext search that can retrieve every substring from a given text corpus. User patterns do not necessarily start and end at word boundaries: they can start near the end of a word, then span several words, and end in the middle of another word.

So, if one wants to avoid a full-word-only search system, one has to avoid phonetic codes. More flexible fulltext search systems usually use string distance measures, where in most cases, the edit distance measure is preferred.

2.2 Edit Distance Based Approaches

In (Navarro et al. 2001), a taxonomy for approximate fulltext searching is specified. This taxonomy also distinguishes three major classes of approaches: *neighborhood generation*, *partitioning into exact search* and *intermediate partitioning.*

Neighborhood generation generates all patterns $P' \in U_k(P)$ that exist in the text, where `editdistance`$(P, P') \leq k$ for a given k (edit distance see below). These neighbor patterns are then searched with a normal, exact search algorithm. This approach works best with suffix trees and suffix arrays, however, $U_k(P)$ can become quite large for long patterns P and greater values of k, which can be problematic.

Partitioning into exact search carefully selects parts of the given pattern that must appear unaltered in the text; then searches for these pattern parts with a normal, exact search algorithm, and finally, checks whether the surrounding parts of the text are close enough to the original pattern parts.

Intermediate partitioning, as the name implies, is located between the above approach classes. Firstly, parts of the pattern are extracted, and neighborhood generation is applied to these small pieces. Because these pieces are much smaller, and thus have fewer errors than the full pattern, their neighborhood will be smaller. Secondly, an exact search is performed on the generated pattern pieces, which also checks if the surrounding text forms a search hit.

Several approaches have been developed in order to combine the speed, and flexibility, of q-gram indices with fault-tolerance. Given the structure of q-gram indices, direct neighborhood generation is not possible within reasonable time. Assuming that the error is in the text itself, Jokinen and Ukkonen show in (Jokinen & Ukkonen 1991) how an approximate search with a q-gram index structure can be achieved by *partitioning into exact search.* Although using the same basic approach, a different algorithm was developed in (Navarro & Baeza-Yates 1998) given that they assumed the error occurred not in the text, but within the pattern. Myers demonstrates in (Myers 1994), an *intermediate partitioning* approach to the approximate search problem on a q-gram index.

All the above methods are based on the definition of Levenshtein's *edit distance* (Levenshtein 1965). This metric calculates the distance between two strings by counting, and then adding, the minimal number of atomic actions *insert*, *delete* and *substitute* of single symbols (Stephen 1994).

Although the metric provides a mathematically well-defined measure for string similarities, it cannot, from a human point of view, satisfactorily model the similarity between natural language fragments.

Given, for instance, the special characteristics of the Hager text corpus, the use of the edit distance measure did not seem appropriate. This is mainly due to the fact that the edit distance processes letters individually (regardless of any context information), and does not provide the means of preferential string substitution: $A \rightarrow B$ versus $A \rightarrow C$, where $|A| \geq 1$, $|B| \geq 1$, $|C| \geq 1$; and $|A|$, $|B|$ and $|C|$ may be arbitrarily different. For example:
`editdistance`("`kalzium`", "`calcium`")=2 and
`editdistance`("`kalzium`", "`tallium`")=2, are the same – despite the fact, that every human reader would rate the first two strings much closer together than the second pair of strings. (note: $|X|$ denotes the length of string X in characters).

Because the edit distance is more suited to modeling random typing mistakes or transmission errors, we needed a way to approximate patterns where the differences between text and pattern are less "random". Instead, these differences arise from the fact that a great number of authors may use different, but correct, spellings of the same scientific term. We also wanted to address cases where the scientific term has been spelt phonetically and, therefore, incorrectly. Our technique of *weighted pattern morphing* is described in the following section.

3. Weighted Pattern Morphing

In this section we present the structure and algorithms of our fault-tolerant front-end, based on the *weighted pattern morphing* (WPM) approach. The results of experiments that led to reasonable parameter settings for our fault-tolerant search engine are also discussed below.

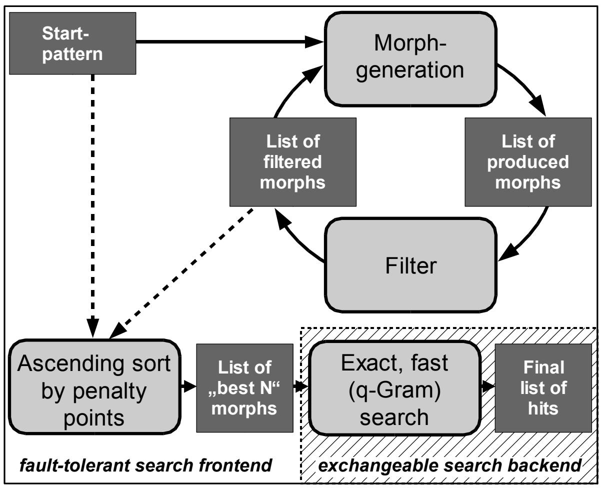

Figure 1. Workflow of weighted pattern morphing

As observed in section 2.2, the edit distance metric, used by most available approximate text retrieval algorithms, is not appropriate when one is trying to model a more human oriented string similarity. WPM overcomes these disadvantages with a simple, but powerful idea described in the following two steps:

1.) Examine searchpattern P for all substrings $p_{i,j}$ $(1 \leq i \leq j \leq |P|)$, which are part of a phoneme group G (where $G=\{g_1,g_2,\ldots,g_z\}$ and $p_{i,j}=g_k$ $(1 \leq k \leq z)$).

2.) Attempt to replace all $p_{i,j}$ which are members of the same phoneme group G, with all g_l $(l \neq k)$.

Table 1. Two example penalty weight matrices: phoneme group "c/g/k..."(top) and numbers (bottom)

	c	**g**	**k**	...
c	–	–	1	...
g	10	–	10	
k	1	15	–	
...	...			

	1	**one**	**2**	...
one	1	–	–	...
1	–	1	–	
two	–	–	1	
...	...			

More general as with the edit distance, here $|p_{i,j}| \geq 1$, $|g_l| \geq 1$ (including $|p_{i,j}| \neq |g_l|$) is possible. A pattern P', in which at least one substitution took place is called a *morph* of P, and a single substitution of $p_{i,j}$ to g_l is called *submorph* $p_{i,j} \rightarrow g_l$.

As the interchangeability of members of the same phoneme group ought to be different, the concept of *penalty weights* was introduced. These penalty weights were stored in two-dimensional *submorph matrices,* where the rows represent the source strings g_k, and the columns the destination strings g_l (see examples in Table 1).

As the table demonstrates, not every possible submorph is allowed, and the matrix may be asymmetric to the diagonal. There exist, however, submorph tables for every common phoneme group such as "a/ah/aa/ar", "i/ie/y/ih/ii", "g/j", "c/g/k/cc/ck/kk/ch", among others. The possibilities of the edit distance can be approximated by submorphs: $\varepsilon \rightarrow$"?" (insert any char); $c \in \Sigma \rightarrow$"?" (substitute a char c); $c \in \Sigma \rightarrow \varepsilon$ (delete), where ε is the empty word, Σ the alphabet, and "?" is the one-letter wildcard for our search engine. Also defined are the more exotic submorphs: (e.g.) `solution`→`sol.`, `ac.`→`acid`, `5`→`five`, and special scientific characters (e.g.) α→`alpha`. These are often helpful in a biochemical and medical context, because abbreviations are used inconsistently by different authors (e.g.) in *HagerROM* the terms "`5-petalled`" and "`five-petalled`" occur).

Every morphed pattern P' is then recursively fed into the same morph algorithm where more submorphs are performed. To avoid recursion loops, the first index i_{min} where submorphs $p_{i,j} \rightarrow g_l$ start, is always increased for deeper recursion levels. Otherwise loops may occur at different submorph recursion levels as $u \rightarrow v$, $v \rightarrow w$, $w \rightarrow u$. On every recursion level, P is also fed unaltered into the next recursion, with only i_{min} increased. This is done to only allow submorphs towards the end of the pattern for some morphs. This leads to an upper bound for the computational time complexity of $\boldsymbol{O}(|P|r)$ for generating all morphs (if r is the total number of morph rules).

Because the recursion tree can become large, the total penalty S (as a sum of the penalty weights for all applied submorphs), and M (the total number of applied submorphs (=recursion depth)) are updated for every recursion call. Recursion backtracking is performed when either S or M pass configurable limits S_{max}, M_{max}, or when $i_{min} > |P|$. As S_{max}, M_{max} and i_{min} increase with every recursion level, the algorithm terminates within reasonable time limits (see section 5).

Unsurprisingly, the above algorithm generates morphs which do not belong to the text corpus. Pre-filtering of "useless" morphs is achieved by the introduction of the *hexagram filter* (see *Filter* in Figure 1). This was deemed necessary despite the high speed of the *q*-gram algorithm in finding out that a pattern has no hits in the text (see (Grimm 2001)).

The hexagram filter is a trie structure with a maximum depth of six. It does not store actual offsets of hexagrams, but simply indicates whether a specific *q*-gram class ($q \leq 6$) exists in the text at all. The term *trie* was first used by Fredkin (derived from 'information re*trie*val') in (Fredkin 1960).

So when a morph P' is generated, the hexagram trie is traversed for every second overlapping hexagram which belongs to P'. If any of the morph's hexagrams are not part of the trie, P' as a whole cannot be part of text T, and is discarded. However, if all the hexagrams of P' are part of the trie, there is no guarantee that P' occurs in T.

The trie depth of six and the selection of every second hexagram was chosen after performing about 14,000 test searches with patterns drawn from the text. We observed a minimum running time at depth=6 and window-delta=2, hence the reason for choosing this value for all subsequent experiments (Esser 2004).

In accordance with their penalty sum, the best B morphs that passed the hexagram filter are then searched by the non fault-tolerant q-gram fulltext retrieval back-end.

Figure 2 gives an exemplary overview of the steps involved from turning the user pattern "`diamphenethide`" into fault-tolerant search results: while WMP applies all possible submorph rules to the start pattern, the hexagram filter deletes those morphs that can't be part of the text. The survivors are searched by the backend.

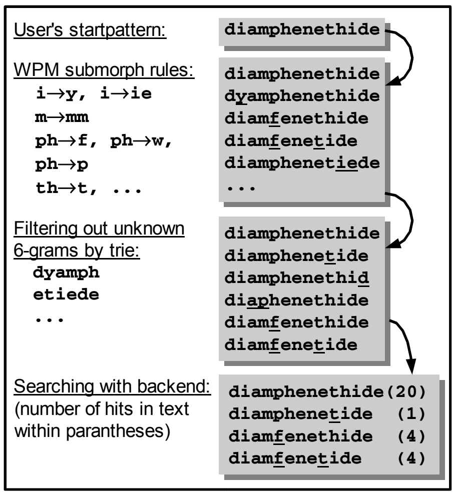

Figure 2. Example of WPM at work on Parasitology

Regarding storage space needs, the core WPM algorithm has only the need for storing the submorph rules with their penalties. Each rule consists of a source and a destination string (each with an average length of 3 chars) and an integer value for the weight, which in total sums up to a negligible amount of data.

Additionally the filter trie needs some storage space. Theoretical examination of trie storage space can be found for example in Knuth's standard work (Knuth 1998) where the trie data structure is covered in depth. The storage space for the hexagram trie used in our system is scalable from around 5% to 100% of |T| by adjusting precision of alphabet coverage. For larger texts we group together infrequent alphabet characters to one trie character to save storage space at a price of making the trie filter less restrictive.

4. Automatic Weight Adjustments

For German words combined with Greek and Latin terms, we manually identified about 25 different phoneme groups that lead to 369 submorphs. We believe that while machines do not have a real understanding of complex texts, and therefore, of the semantic similarity of certain words, it is impossible to automatically derive the above mentioned submorph rules. However, the desired weights can be obtained by a machine with the procedure described in this section. For a first prototype, the penalty weights for the above submorphs were adjusted manually by a native speaker.

For the generation of English morph matrices as required for the "Parasitology Encyclopedia"(see above) we relied on linguistic research publications like such as Mark Rosenfelder's "*Hou tu pranownse Inglish*" (Rosenfelder 2000). Though Rosenfelder focuses on rules to get from spelling to sound, we used this work to identify about 35 English sound groups, and their possible spelling which lead to morph matrices with about 900 submorphs (e.g. `ible`↔`able`, `z`↔`s`, `ea`↔`e`). Additional submorphs for numbers (`100`↔`hundred`), and domain specific abbreviations were added afterwards.

After changing an algorithm's parameters, the developer needs some kind of measure to investigate whether the parameter change had positive, negative or zero influence on the quality of the algorithm's results. Our performance measure is explained in the following sub-section.

4.1 Performance Measuring Method

Because later desired user patterns remain an unpredictable factor at the time of the algorithm's design, measuring the performance of an approximate search approach is a difficult task. A sometimes used technique to find a measure for the performance of a fuzzy search technique is performing *precision & recall* tests.

So, for a given query term, someone chooses all R terms of the studied text corpus that would be relevant hits for the query. Since most approximation algorithms have one or more parameters which control the degree of fuzziness, two values are logged for the range of these fuzziness levels:

1.) *recall* ("completeness of the retrieval"): the percentage of how many of the relevant terms have been found.
2.) *precision* ("purity of the retrieval"): the proportion of how many of the total retrieved results are relevant.

Usually, as recall rises to 100%, precision drops. This is because of the increase of irrelevant terms that not only populate, but blur, the retrieval results.

Precision and recall measurement has three obvious disadvantages: one, it needs a large amount of time of an expert (or expert group) in the specific domain of the studied text corpus. This expert must identify potentially interesting queries; find and rank all (!) relevant variants that should belong to a complete retrieval result of this "hypothetical" query. This leads to problem two: as large text corpora cannot be browsed manually, the expert needs a retrieval tool to extract the set of relevant terms. But this tool is, yet again, a fuzzy retrieval system. So, every term that this fuzzy search cannot find, will not ever appear in the set of relevant terms.

Disadvantage three lies in the fact that even if a group of experts has deduced which retrieval results are relevant to a query, the end-user of the retrieval system might have a different opinion: "*An information need cannot be fully expressed as a search request that is independent of innumerable presuppositions of context - context that itself is impossible to describe fully, for it includes among other things the requester's own background of knowledge*" (Swanson 1988). Therefore, precision and recall measurement is never unbiased, and is always in danger of optimizing the studied algorithm towards the expert's opinion which might not necessarily fulfill the user's wishes.

Due to the problems described above we relied on a different performance measure which consists of the following steps:

1.) To obtain test patterns we extracted every word W from the given text corpus with $9 \leq |W| \leq 30$ (the lower bound of nine characters was the median of the length for about 20,000 searches users performed through the online search interface of Altmeyer's encyclopedia (Altmeyer & Bacharach-Buhles 2002)). Nine characters provide sufficient context to retrieve related words with one or two applied submorphs. The resulting list of words for Altmeyer's text corpus consisted of 62,395 unique test-words.
2.) These test-words where fed into the fault tolerant search algorithm. We counted, how often WPM could generate a different word that was part of the text corpus – called a "*valid morph target*". So it was regarded as "ground truth" that if a word of the text corpus could be morphed into a different word of the text corpus only by application of the morph rules, that both source and destination word mean the same thing and the version with more occurrences in the text is the rectified word. Manual random samples showed that this is true for the vast majority of word pairs.

For example:
`biotinilated`→`biotinylated` is counted as one valid morph target. But,
`biotinilated`→`iotinylated`, does not count as valid target morph since `iotinylated` is an incomplete word.

The more valid morph targets our algorithm generated, the better we rated its performance. Based on the above measure, we performed tests to improve the performance of WPM by fine-tuning the penalty matrices' weights. The details of the experiments, and the significant performance improvement we achieved are described in the following sub-section.

4.2 Morph Feedback for Weight Adjustment

To improve the penalty matrix weights, we fed all words W, where $9 \leq |W| \leq 30$, from the given text corpus into the fault-tolerant WPM algorithm. Everytime we got a valid morph target, we automatically logged which submorphs were involved in generating the valid target morph. After performing 62,395 unique searches, we observed that some morph rules were consistently useful, while others were used to a lesser degree, if at all. We ranked the list of submorphs according to the frequency of their successful application.

The new submorph weights were then derived from the following ranking criteria: helpful submorphs received smaller penalty weights, while unused submorphs got the maximum penalty value. This procedure was called *morph feedback*.

Table 2 shows an excerpt of the usage statistic for both manual, and new weights, applied to submorphs after counting the frequency that each submorph produced a new valid target morph.

Table 2. Adjusting submorph weights by morph feedback

Listpos.	*#valid target morphs*	*From*	*To*	*Manual Weight*	*New Weight*
1	497	k	c	1	1
2	490	c	k	1	1
3	161	z	c	5	1
4	158	c	z	5	1
5	95	ie	y	10	1
6	94	ß	ss	1	1
7	86	y	ie	15	1
8	82	th	t	1	1
43	16	f	ph	5	5
79	4	ck	c	20	13
97	2	zwanzig[1)]	20	10	15
134	1	tz	c	10	16
369	0	w	ph	25	30

[1)] 'zwanzig' (German) = 'twenty' (English)

If w_i is the manually generated weight of rule r_i, then the new weight column of Table 2 is calculated as follows:

0.) Let μ_{max} be the maximum number of valid morph targets the best rule had (in Table: Pos.#1, rule k→c).
Let S_{max1}=20 and S_{max2}=30 be the maximum allowed penalty weights at tolerance level 1 and 2 (medium and high).
Let φ=0.9 be the parameter to flatten the curve and let μ_i be the valid target morphs of rule r_i.

1.) Set $\delta = \frac{\sqrt{\mu_{max}}}{S_{max1}} - \varphi$ as the stepping delta for all rules.

2.) Then the new weight $\bar{w}_i$ of rule r_i is calculated as:

$$\bar{w}_i = \begin{cases} S_{max2} & \text{if } \mu_i = 0 \\ S_{max1} - \frac{\sqrt{\mu_i}}{\delta} & \text{else} \end{cases}$$

We then compared the behavior of WPM through morph feedback with different weight sets (see Figure 3): 1. Random weights; 2. Manually generated weights; 3. Automatically optimized weights (based on manual!).

With stricter maximum allowed penalty sum S_{max} the manually generated weight set produces more valid target morphs than the set of random weights. The step structure of the graph of manual weights is due to the fact that not every arbitrary weight value was chosen at weight definition. Figure 3 also reveals that the technique of morph feedback based on the words from the text corpus, improved the performance of the algorithm significantly. More valid target morphs could be generated with less tolerance (smaller S_{max}) at earlier stages of the algorithm.

Up till now the complete process of adjusting and improving the morph weights happened in some sort of ″clean room preprocessing″ without any knowledge of what patterns the users want to search with the algorithm. Because the text corpus was source (for the test patterns) *and* destination (for validating the morph targets) of the morph feedback technique, it was unclear if the improved weights would also result in increased performance for patterns of "real world" users. To verify this we relied on the log-files we gained from the online version of Altmeyer′s Dermatology Encyclopedia.

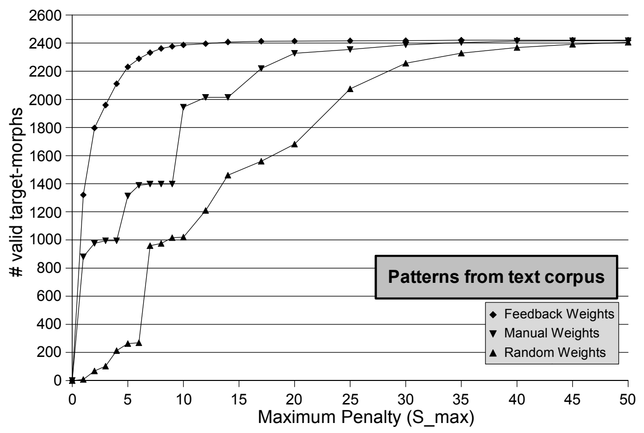

Figure 3. Altmeyer: Performance of different weight sets on 62,385 words drawn from the text corpus

With the Altmeyer text corpus we had the possibility to verify, if the ″clean room optimized weights″ would also improve the productive efficiency of WPM for real queries of real users. Over the past ten months, users of the online version of Altmeyer′s encyclopedia entered about 20,000 patterns to our fulltext search interface. As some of these patterns occurred more than once, we collected about 8,505 distinct queries from our web server′s log files. Of these queries only 5,424 words W satisfied the length constraint 9≤|W|≤30.

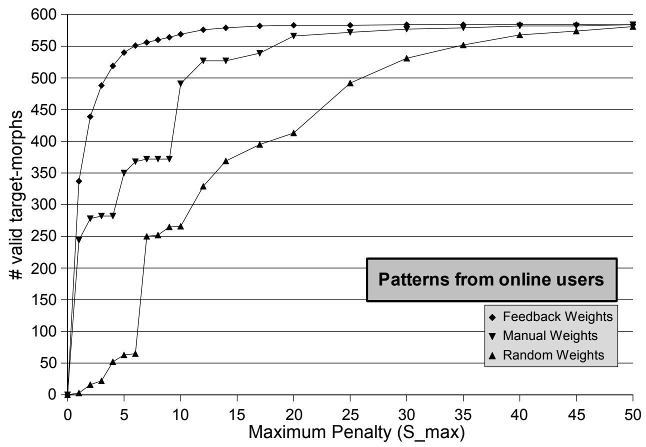

Figure 4. User queries: Performance of different weight sets on 5,424 different search terms collected from the online logs of Altmeyer

Figure 4 shows that the same weights that had been automatically adjusted in the preprocessing step - without knowledge of real user queries - (see table 2 and figure 3) also improve the performance of the WPM retrieval algorithm on the patterns of the online users in a similar way to the corpus-to-corpus experiments.

So, the experiments of figure 4 encourage us to future application of the described automatic pre-processing weight adjustment technique on search environments where real user queries are unknown or not present in a

Table 3. Examples of WPM results on the English text corpus of "Parasitology Research"

Search Pattern	*Usertime*	*#Morphs*	*#Filtered out*	*#Morphs with Hits*	*Morphs with Hits*	*#Morphs w/o Filter*
3-dimensional	93 ms	1504	1502	2	3-dimensional (4) three-dimensional (29)	8154
acknowledgements	156 ms	10147	10143	3	acknowledgements (359) acknowledgments (4) acknowlegements (1)	25688
anthelminthic	109 ms	2830	2823	3	anthelminthic (12) anthelminthik (3) anthelmintic (394)	4706
antibacterial	125 ms	5347	5342	2	antibacterial (34) anti-bacterial (2)	10600
bielorussia	109 ms	3965	3961	4	bielorussia (2) byelorussia (1) belorussia (48) belarussia (9)	9437
bromosulfophthalein	140 ms	5772	5764	3	bromosulfophthalein (3) bromosulphophtalein (1) bromosulfophthlein (2)	20562
cacodilate	109 ms	4497	4495	3	cacodilate (1) cacodylate (105) cocodylate (1)	6952
cesbron-delauw	125 ms	5442	5439	2	cesbron-delauw (12) cesbron-delaw (3)	12828
cholodkawsky	156 ms	6725	6722	3	cholodkawsky (1) cholodkowsky (2) cholodkovsky (3)	12521
diamphenetide	125 ms	3491	3488	4	diamphenetide (1) diamphenethide (17) diamfenethide (4) diamfenetide (4)	8448
neighbour	125 ms	4599	4598	2	neighbour (38) neighbor (61)	7845
tübingen	73 ms	613	611	3	tübingen (55) tubingen(56) tuebingen (15)	2418
zerkarien	93 ms	3298	3296	2	zerkarien (7) cercarien (10)	6722

significant amount (e.g., HagerROM). Or, in other words: the presence of real user search patterns is not necessary to fine-tune the morph penalty weights when the described technique is applied. This is a very interesting result of this paper, as it frees the developer from the usual vicious circle: If the programmer has no existing user feedback it is difficult to fine-tune the algorithm for real-world scenarios. But if an algorithm performs poorly due to missing fine-tuning, users might not use it frequently.

5. Experiments on Filter Efficiency and Speed

In this section we discuss experiments regarding the filter efficiency and the speed of the presented fault-tolerant approach. To increase understanding of an international readership, we manually chose examples from the *Parasitology Research* text corpus, an English language publication. *Parasitology* consists of 50MB of XHTML text in about 3,500 articles which results in about 28MB of raw text (after deleting layout tags).

Table 3 shows the results of some fault-tolerant WPM search experiments. The number of actual morph hits is shown in parentheses. Column 3 to 5 reflect the efficiency of the trie filter (compare column 7). To show the efficiency of the hexagram filter trie, filtering was switched off temporarily (every string passes "O.K.") and, therefore, many more "useless" morphs had to be examined (this is clearly seen in the last column). Column 2 shows that the average time a user has to wait for fault tolerant search results is approximately 120 milliseconds, thus falling within a tolerable range.

Column 6 of Table 3 (labeled *Morphs with Hits*) shows the broad range of word variants which important information carrying terms may have. Despite terms with different legal spellings (for example, `zerkarien` is the German variant of the Latin and English spelling of `cercarien`), we have word differences resulting from special language char entities such as the German umlauts which are absent in non-German keyboards. So the German town ″Tübingen″ has two additional morphs: `tubingen` (diaresis ommited) and `tuebingen` (′ü′ correctly expanded to ′ue′).

In addition the problem of spelling mistakes becomes apparent – in this case probably due to the case that an international journal like ″Parasitology Research″ has many contributing authors that are not native English speakers. So words like ″acknowledgements″ and ″bielorussia″ have various incorrect variants.

All experiments were performed on a standard PC with AMD Athlon® 2.66GHz CPU and 512 MB RAM on local ATA-66 hard disk under Windows XP®. The compressed q-gram index q = {1,2,3,4} needs about 200MB storage-space (this is 7 times |T|) and can be generated on a state-of-the-art Linux computeserver in about half an hour. The filter trie index needs 1.2 MB and thus can be kept in RAM during morph generation for speedup reasons.

6. Conclusion

We demonstrated that nowadays average end-user PCs are capable of performing multiple, iterated, exact text retrievals over a set of morphed patterns and thus simulate a fault-tolerant search. Morph matrices with penalty weights seem much more suitable and flexible to model phonetic similarities and spelling variants in multilingual, multi-author texts than the edit distance metric or phonetic codes like Soundex and its successors.

Automatic definition of allowed submorphs seems impossible without deep understanding of the specific linguistic characteristics of the text corpus′ language, and thus needs manual work. But, defining useful weights for the identified submorphs is a tedious task, which can be accomplished automatically with our technique of morph feedback weight adjustment. We have shown that the generated submorph weights accordingly improve the performance of the WPM algorithm significantly when operating on real user queries.

A further interesting aspect of WPM is that the fault-tolerant part of the algorithm is completely independent of the actual search algorithm and its index. This makes WPM an ideal add-on to existing search engines like databases or web retrieval systems.

Future research will define submorphs and weights for a medieval German text as WPM will drive the fault-tolerant retrieval of the electronic version of Magister Lorenz Fries′ (1489 – 1550) handwritten 1,000 pages "*Chronicle of the Bishops of Würzburg from 742 to 1495*" which will be released in 2004, the 1300-year foundation anniversary of Würzburg.

7. References

Altmeyer P., Bacharach-Buhles M. (2002) *Springer Enzyklopädie Dermatologie, Allergologie, Umweltmedizin*, Springer-Verlag Berlin Heidelberg, ISBN:3-540-41361-8 `http://www.galderma.de/anmeldung.enz.html`

Blaschek W., Ebel S., Hackenthal E., Holzgrabe U., Keller K.,Reichling J. (Hrg.) (2003) *HagerROM 2003 - Hagers Handbuch der Drogen und Arzneistoffe. CD-ROM*, Springer Verlag, Heidelberg, ISBN:3-540-14951-1 `http://www.hagerrom.de`

Esser W. (2004) *Fault-tolerant Fulltext Information Retrieval in Digital Multilingual Encyclopedias with Weighted Pattern Morphing*, McDonald S, Tait J.: Advances in Information Retrieval, Procedings of the 26th ECIR, LNCS 2997, Springer, pp.338-351. `http://www.derwok.de/papers/esser_wpm_ecir04.pdf`

Fredkin E. (1960) *Trie Memory*, Communications of the ACM Vol. 3 (9), pp.490-499. `http://portal.acm.org/citation.cfm?doid=367390.367400`

Gadd T. (1990) *PHONIX: The algorithm*, Program: automated library and information system 24(3), pp.363-366.

Grimm M. (2001) *Random Access und Caching für q-Gramm-Suchverfahren*, Diplomarbeit Lehrstuhl für Informatik II, Universität Würzburg

Hodge V., Austin J. (2001) *An Evaluation of Phonetic Spell Checkers*, Tech.Report YCS338 Dept. of CS, University of York, U.K. `http://citeseer.ist.psu.edu/463597.html`

Jokinen P., Ukkonen E. (1991) *Two algorithms for approximate string matching in static texts*, LNCS 520, Springer Verlag, pp.240-248.

Knuth D. (1998) *The Art of Computer Programming, Second Ed., Volume 3: Sorting and Searching*, Addison Wesley, ISBN:0-201-89685-0

Levenshtein V. (1965) *Binary codes capable of correcting deletions, insertions, and reversals*, Problems in Information Transmission 1, pp.8-17.

Myers E. (1994) *A sublinear algorithm for approximate keyword searching*, Algorithmica, 12(4/5), pp.345–374.

Navarro G., Baeza-Yates R. (1998) *A Practical q-Gram Index for Text Retrieval Allowing Errors*, CLEI (Centro Latinoamericano de Estudios en Informatica) Electronic Journal, Vol. 1, No. 2, pp.1. `http://www.clei.cl/cleiej/paper.php?id=32`

Navarro G., Baeza-Yates R., Sutinen E., Tarhio J. (2001) *Indexing Methods for Approximate String Matching*, IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 24, No. 4, pp.19-27. http://doi.acm.org/10.1145/375360.375365

Philips L. (2000) *The Double Metaphone Search Algorithm*, C/C++ Users Journal, pp.1. http://www.cuj.com/documents/s=8038/cuj0006 philips/

Rosenfelder M. (2003) *Hou tu pranownse Inglish.* http://www.zompist.com/spell.html

Russell R. (1918) *INDEX (Soundex Patent)*, U.S. Patent No. 1,261,167, pp.1-4. http://patimg2.uspto.gov/.piw?Docid=01261167 &idkey=E7455D7DADEF

Stephen G. (1994) *String Searching Algorithms, Lecture Notes Series on Computing, Vol. 3*, World Scientific Publishing, ISBN:981-02-1829-X

Swanson D. (1988) *Historical note: Information retrieval and the future of an illusion*, Journal of the American Society for Information Science, 39(2), pp.92-98.

Zobel J., Dart Ph. (1996) *Phonetic String Matching: Lessons from Information Retrieval*, ACM Press: SIGIR96, pp.166-172. http://doi.acm.org/10.1145/243199.243258

Zobel J., Dart Ph. (1995) *Finding Approximate Matches in Large Lexicons*, Software - Practice and Experience, pp.331-345.