Combining Index Structures for application-specific String Similarity Predicates

André Reckhemke
Poznan University of Technology
andre.reckhemke@gmx.de

Eike Schallehn
University of Magdeburg
eike@iti.cs.uni-magdeburg.de

Abstract

This paper presents new approaches for supporting string similarity matching based on a combination of techniques from the fields of information technology and computational linguistics to achieve better results regarding accuracy and efficiency. The homogenization of plain text reduces the volume of index structures and concurrently increases the quality of hit-lists. Furthermore it shows the careful and context dependent dealing with abbreviation, acronyms, and synonyms. The core of this work is a general approach to support stepwise application-specific and index supported string similarity predicates. It uses the Hybrid-Ternary Search Trie as one of the fastest index structure for strings. Tries guarantee best results for exact matching as well as inexact matching in preprocessed data and can be used for external data storage. Especially Hybrid-Ternary Search Tries are easy adaptable for common strings and provide best average results for inexact matching without any limitations.

1 Introduction

Matching strings strings by similarity is an important issue in information retrieval and database management systems, but it is gaining even more importance in integration scenarios and heterogeneous environments like the World-wide Web. The approach presented here is based on homogenization of strings, the usage of word lists, and the support by index structures. The pre-processing for matching may include the homogenization of each single token, expanding of acronyms and abbreviations, replacement of each single word with its lemma, substitutions of synonyms, and deleting of stop words. The achievement of the defined goal heavily depends on the effort of implementation and on the maintenance of each single word list. In best cases the preprocessing leads to a quality of 95%-99% [Zie00]. The remaining percent is mostly based on mistakes in the disambiguating of dots and on a false substitution of synonyms. Methods (algorithms) like phonetic transformations or stemming are easy to handle (no maintenance), but do not deliver a high quality. For this reason, a mix of the latter mentioned methods improves the search for application-specific information.

In contrast to traditional index structures, tries and trees are developed for the storage of strings. They are efficient in time and space (the length of the string has no influence) and can be used for external storage systems. Besides that, these index structures support different kinds of string searching operations. For example, some of them efficiently support approximate string matching and other ones are better for pattern matching. One Trie variant that combines all features for string matching is the Hybrid-Ternary Search Trie (HTST) [Sed03]. Its root can be individually customized and all nodes have the same size.

Approximate string matching is supported by several approaches [Nav01]. Some of them are not applicable to preprocessed texts and are optimized for special problems. Ukkonen's CutOff mechanism [Ukk85] supports approximate string matching in tries based on the Levenshtein distance [Lev65].

Based on these previous approaches, this paper provides the following own contributions: extension and evaluation of the HTST for different string matching operations, the combination of similarity predicates, index based approach for application-specific string matching, and an analysis and evaluation of word lists.

2 Supporting String Similarity Predicates

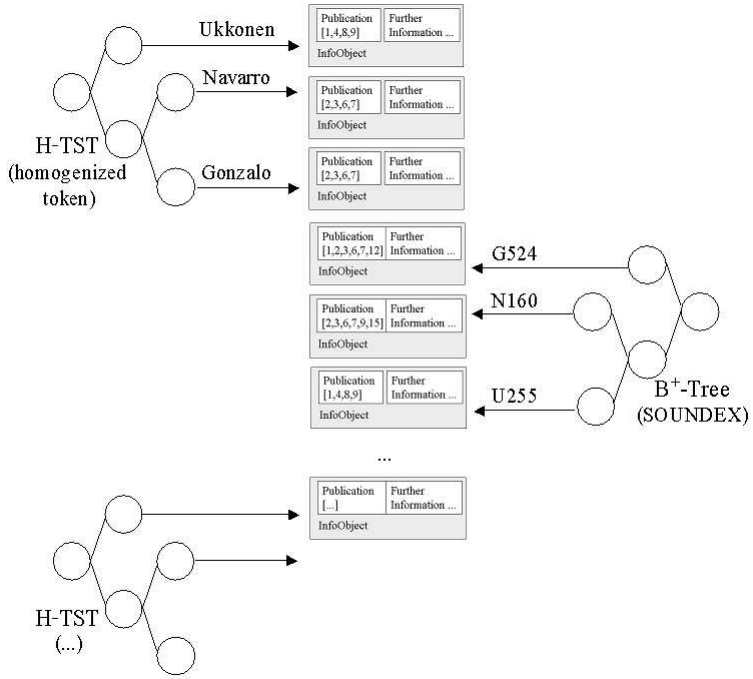

Figure 1: Combining index structures for similarity matching

The presented approach is supported by three kinds of word lists and applies several transforming rules to one string. Stop word lists and lists of abbreviations and acronyms can be adapted from the standard frequency lists like the British National Corpus (BNC), but must be enriched with words of the specific application area. This can be done either by using special corpora, or by preprocessing the specific text and creation local consistency abbreviation and acronym lists.

For improving application specific string matching, several transforming rules will be applied to one string and connected by a simple OR operation to improve the recall. Therefore, it is necessary to create different index structures for each transformed string (e.g. SOUNDEX, stemming). For saving strings of variable length the Hybrid-TST is the best choice; numbers or strings with identical length can be stored in the B^+-tree (or hash index). One of the key ingredients of our approach is an implementation of the Hybrid-Ternary Search Trie with an extension to support approximate string matching based on the Levenshtein distance and Ukkonen's CutOff mechanism. The Ternary Search Trie developed by Bentley and Sedgewick is the latest development level in the Trie history and based on n-ary tries and digital search tries. It works with the digital transformation of each character and reduces the drawbacks of n-ary tries (many children have no successor nodes) [Sed03]. An extensive theoretical analysis of the TST is presented in [Cle97]. Derived from the TST, the Hybrid-TST enhanced this Trie with a fix array as the header, as shown in Figure 2). In its basic version, each cell presents one character and leads to an array considering all characters of the given alphabet [Sed03].

We extend the Hybrid-TST for approximate string matching applying techniques from Dynamic Programming as introduced by Ukkonen for common Tries. To illustrate Ukkonen's

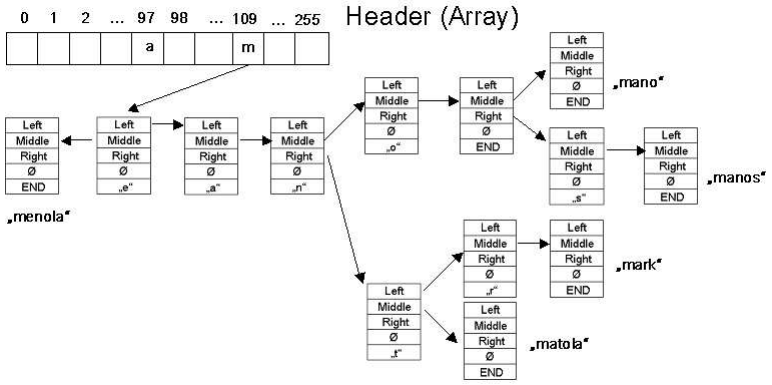

Figure 2: Hybrid-TST

CutOff mechanism let us assume we have a standard Trie, search string 'sand' and edist is limited to one. This algorithm calculates for each prefix the minimum number of edit operations as shown in Figure 3 and Table 1.

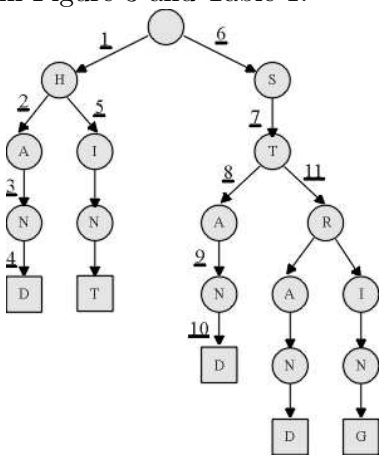

Figure 3: Graph: Ukkonen's CutOff mechanism

Searchpath	Prefix	edist	Action
1	H	1	accepted
2	HA	1	accepted
3	HAN	1	accepted
4	HAND	1	hit
5	HI	2	cutoff
6	S	0	accepted
7	ST	1	accepted
8	STA	1	accepted
9	STAN	1	accepted
10	STAND	1	hit
11	STR	2	cutoff

Table 1: Ukkonen's CutOff mechanism

3 Implementation and Evaluation

For evaluation purposes, a prototype implements the Hybrid-TST and the SOUNDEX code. The former supports exact and approximate matching. The latter exemplary shows the possibility to combine several matching rules and is stored in a B^+-Tree. Figure 4 shows the general implementation structure.

As a test data set the *Digital Bibliography & Library Project* (DBLP) was selected. The stop word list is taken from the general frequency list of the BNC and mainly includes the first 100 entries. The abbreviation and acronym lists are based on common entries of the BNC and of much more special entries in the DBLP. Author and titles are saved in different and independent index structures, each single publication is saved with an unique index in a hash table. Each leaf (Hybrid-TST, B^+-Tree) refers to an info-object which includes the relevant publications. The hit-lists combine all sub-hit-list by a simple set operation(OR).

Using this implementation and test scenario thorough evaluations were carried out regarding insertion operations, exact matching, approximate matching, and the accuracy. Detailed results are described in [Reck05]. As an example, Figure 5 presents a comparison of approximate matching for six different pattern lengths in a Hybrid-TST of 1,000,000 strings. The time is assigned to the Y-axis and the X-axis represents different k distances. It shows that the request

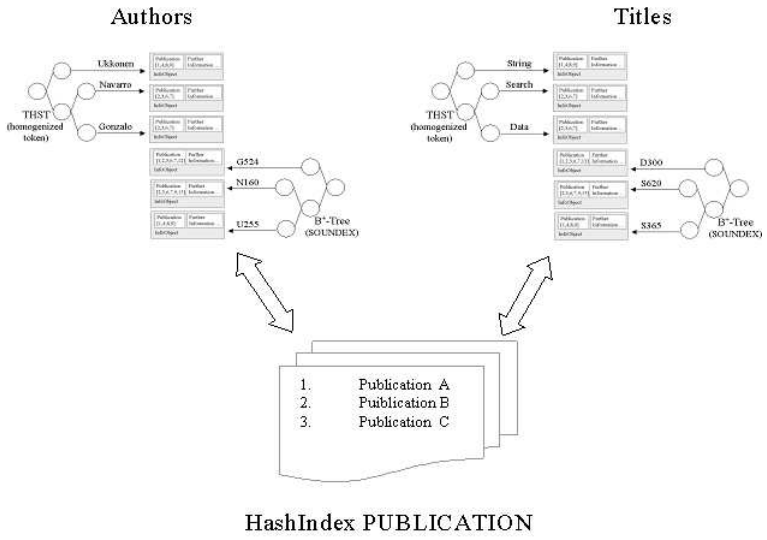

Figure 4: Structure of the implementation

time of each graph (logarithmic scale) increases with the pattern length m and the k distance. Furthermore, it shows the exponential dependency of k and m.

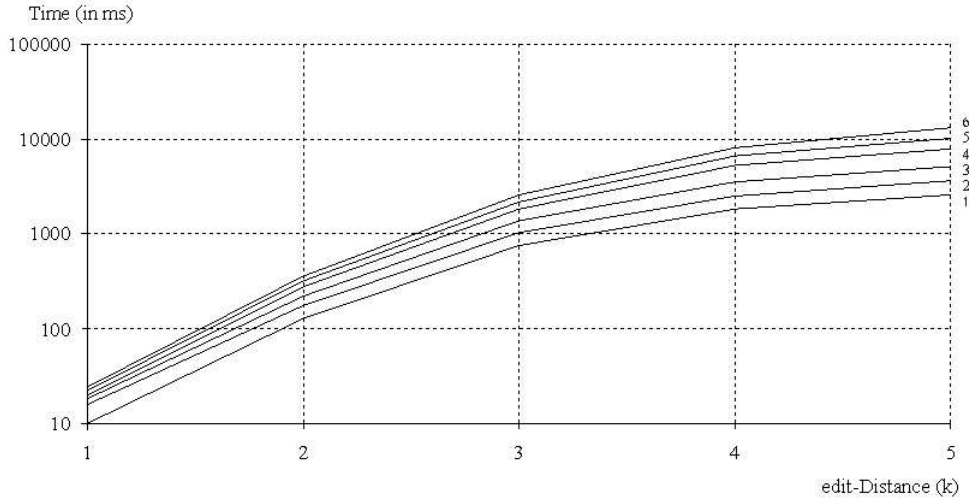

Figure 5: Hybrid-TST: graph for approx. string matching (n=1,000,000) - 1. m=2, 2. m=4, 3. m=6, 4. m=10, 5. m=14, 6. m=20 - logarithmic scale

Furthermore, for the evaluation the DBLP data was analyzed regarding the usefulness of abbreviations, acronyms and stop words for combined similarity based matching. Table 2 splits the DBLP data into several word groups. In the case of the titles, the most important step is the expansion of acronyms (e.g., DBMS → Database Management System). The expansion of initials (acronyms) of author names is almost impossible (without the acceptance of mistakes) and the number of stop words as well as abbreviations can be neglected.

The combination of approximate string matching and SOUNDEX has increased the total number of hits. The quality of this hit-list depends on the operation mode: an OR operation shows the join of both hit-lists (better recall), an AND operation presents the intersection and improves the quality (better precision). An evaluation, based solely on quantitative statements is a task for further work.

Word Type	Set	Author	Title	Σ
abbreviations	all	0	1,386	1,386
	grouped	0	535	-
acronyms	all	501,092	120,319	621,411
	grouped	100	21,810	-
stop words	all	1,432	1,178,459	1,179,891
	grouped	68	112	-
other	all	2,627,880	3,546,642	6,174,522
	grouped	59,313	32,428	-
Σ	all	3,130,404	4,846,806	7,977,210
	grouped	159,481	54,885	-

Table 2: Word groups of the DBLP

4 Conclusion and Outlook

In this paper we proposed to combine methods from information technology with methods from computational linguistics to support efficient and accurate similarity-based string matching. For this purpose we extended the Hybrid-TST and achieved good results for all evaluated string operations fulfilling the requirements for the usage in real-time applications. All mentioned activities focused on English written text, but in most cases it can be easily adapted to French, German, or Spanish texts.

The key aspect of future work must be further research on other relevant preprocessing and matching techniques to work toward an easily usable framework for constructing and using application specific similarity-measures.

5 Bibliography

References

[Cle97] Clement J.: The analysis of Hybrid Trie Structures, *Algorithms project, INRIA Rocquencourt, 1997*

[Lev65] Levenshtein V.: Binary codes capable of correcting spurios insertations and deletions of ones, *Probl. Inf. Transmission 1, pp. 8-17, 1965*

[Nav01] Navarro G.: A guided tour to approximate string matching, *ACM, Vol. 33, No. 1, pp. 31-88, 2001*

[Reck05] Reckhemke A.: The construction of application-specific and index supported string similarity predicates, Master Thesis, *Poznan University of Technology, 2005*

[Sed03] Sedgewick R.: Algorithms in Java - Parts 1-4 (3rd Edition), *Addison-Wesley, 2003*

[Ukk85] Ukkonen E.: Finding approximate patterns in strings, *J. Algor., Vol. 4, No. 1-3, pp. 132-137, 1985*

[Zie00] Zierl M.: (German) Entwicklung und Implentierung eines Datenbanksystems zur Speicherung und Verarbeitung von Textkorpora (2. Auflage), *Friedrich-Alexander Universität Erlangen-Nürnberg Inst. für Computerlinguistik, 2000*