On Virtual Partitioning of Large Dictionaries for Contextual Post-Processing to Improve Character Recognition

Rainer Hoch

DFKI-RR-93-45

This report is an improved and longer version of a paper published in the Proceedings of *Second IAPR Conference on Document Analysis and Recognition (ICDAR'93)*. Tsukuba Science City, Japan, October 20 - 22, 1993 (the paper is one of only 20 long papers from 216 accepted papers in total)

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITW-9003 0).

On Virtual Partitioning of Large Dictionaries for Contextual Post-Processing to Improve Character Recognition [1]

Rainer Hoch

German Research Center for Artificial Intelligence (DFKI)
P.O. Box 20 80, D - 67608 Kaiserslautern, Germany
Phone: (++49) 631-205-3584, Fax: (++49) 631-205-3210
hoch@dfki.uni-kl.de

ABSTRACT: This paper presents a new approach to the partitioning of large dictionaries by virtual views. The basic idea is that additional knowledge sources of text recognition and text analysis are employed for fast dictionary look-up in order to prune search space through static or dynamic views. The heart of the system is a redundant hashing technique which involves a set of hash functions dealing with noisy input efficiently. Currently, the system is composed of two main system components: the dictionary generator and the dictionary controller. While the dictionary generator initially builds the system by using profiles and source dictionaries, the controller allows the flexible integration of different search heuristics. Results prove that our system achieves a respectable speed-up of dictionary access time.

KEYWORDS: contextual post-processing, word validation, large vocabularies, dictionary look-up, fast access, indirect hashing, views, filter, exact string matching.

[1] This work has been supported by the Germany Ministry for Research and Technology BMFT under contract ITW 9003 0.

TABLE OF CONTENTS:

1 Introduction

In this paper we describe how lexical knowledge is efficiently used to support document analysis resulting in a hybrid dictionary organization. Our work has been influenced by [15] integrating various knowledge sources in text recognition where the dictionary (or lexicon) is some special kind of contextual knowledge.

The paper shows the consequent continuation and implementation of the dictionary architecture presented in [5] and [6]. Our actual prototype conforms with the dictionary requirements for document analysis also stated in [5]. For character recognition, the requirements comprise in brief: fast access for the verification of correct words, efficient pattern matching for noisy input, tolerance and robustness towards different kinds of errors, compact storage allocation, flexibility for dictionary maintenance, openness as well as the definition of views on the dictionary.

The dictionary is a central component of our document analysis system improving results of character recognition as well as enabling partial document understanding. The goal of the document analysis system is to close the gap between traditional printed products and the electronic medium. Exemplary, the structure and partial semantics of German business letters are analyzed. The system is model-driven and based on the ODA (Office Document Architecture) platform, an international standard for the representation and the exchange of documents. The system includes several interlocked phases of analysis: *layout extraction*, *logical labeling*, *text recognition*, and *partial text analysis*. Details and experimental results of the analysis system are given in [1].

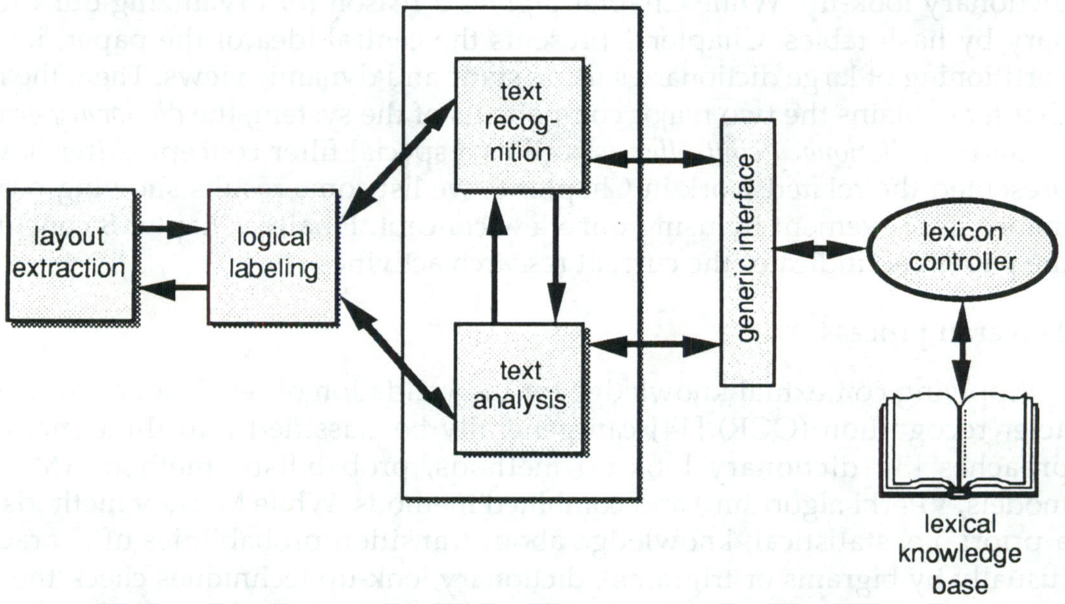


Fig. 1: Phases of document analysis and role of dictionary

In this particular article, we concentrate on the central dictionary module. Fig. 1 illuminates the role of the dictionary in our analysis system. The dictionary is based on a hybrid architecture involving two different data structures (hash tables, tries) to deal with different types of noisy input most efficiently [2].

For accessing the dictionary, additional knowledge sources of text recognition and text analysis can be used to prune search space and to speed up search in case of noisy input. For example, the context of a word according to the logical structure of a document often comprises a relative small set of words such as relevant words of addresses (names, streets, cities, zip codes) or company-specific data (employees, products, tasks, etc.). Thus, the inherent complexity of a complete search over the entire and large dictionary can be avoided.

Consequently, our dictionary allows the definition of what we call *views*. Views can be seen as contextual restrictions on a large dictionary [2], i.e. the dictionary is virtually (not physically!) partitioned into several parts. In other words, a completely new access structure is attached to the dictionary speeding up word verification.

If post-processing of recognized words has additional information about possible views, e.g. structural information of a document, word verification is facilitated drastically. Because our view concept is rather general, arbitrary types of views can be defined: logical views (determined by document structure such as address), letter-based views (significant characters, n-grams, word length, etc.), syntactic views (e.g. noun, verb, adjective, etc.) as well as dynamic views (determined by document contents such as employee names, products, etc.).

The paper is organized as follows: Chapter 2 sketches the overall process of dictionary look-up. While Chapter 3 gives a reason for organizing our dictionary by hash tables, Chapter 4 presents the central idea of the paper, i.e. the partitioning of large dictionaries using static and dynamic views. Then, the next chapter explains the two major components of the system, the *dictionary generator* and the *dictionary controller* as well as a special filter concept. After having presented the related work in Chapter 6, we list some results showing performance improvement by using our view concept. Finally, Chapter 8 concludes the paper and indicates the current research activities.

2 Overall process

Applying contextual knowledge for the validation of results of optical character recognition (OCR) [14] can generally be classified into three main approaches [3]: dictionary look-up methods, probabilistic methods (Markov models, Viterbi algorithm) and combined methods. While Markov methods use a priori, i.e. statistical, knowledge about transition probabilities of characters (usually by bigrams or trigrams), dictionary look-up techniques check the current input string against a legal set of words being collected in a dictionary.

[2] We call a dictionary large if it is a real-life collection of more than 100,000 entries.

We use a dictionary-based approach for two reasons: First, while statistical methods are very fast and efficient, they are extremely sensitive in case of noisy input. Moreover, our tests with bigrams and trigrams of the German language for the rejection of illegal words yielded poor results. Second, storing dictionary entries associated with appropriate lexical information enables a subsequent partial analysis of the document's contents (keyword analysis and syntactic parsing [1]).

In Fig. 2 the overall process of dictionary look-up is depicted.

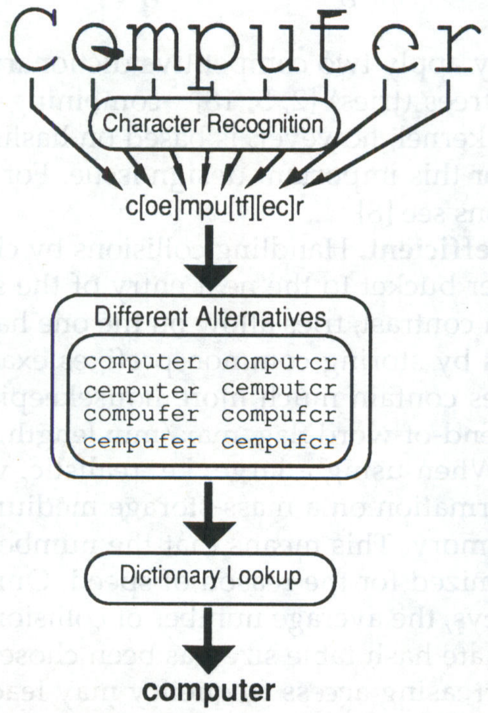


Fig 2: Contextual post-processing of recognition results

While designing our dictionary, we had to consider several system restrictions:

• Because of the large vocabulary, the dictionary should not be kept in main memory—it rather should be primarily organized on secondary storage (files).

• Keeping the vocabulary in a modular way facilitates maintenance of the dictionary allowing the modification of entries as well as the adaptation to other domains.

• Avoid redundant information: Redundancy, i.e. storing entries several times, would blow up our files and renders all maintenance tasks more difficult.

These additional requirements were decisive for organizing the dictionary using hash tables which is motivated in the next section.

3 Dictionary organization and access

A review of literature shows that a multitude of dictionary organizations and corresponding access techniques have been developed. Knuth [8] and Elliman [3] give a survey of adequate data structures for the representation of dictionaries. Harris [4] also compares different dictionary structures (binary search, indexing, tries, hashing) and furthermore indicates a possible syntax of lexical entries. However, sophisticated techniques which are adapted to the special needs of character recognition dealing with incomplete input are hardly found in literature.

Our studies mainly apply two competitive dictionary data structures—hash tables [8] and letter trees (tries) [2, 5, 18]—combining inherent advantages of both. The dictionary kernel, however, is based on hashing. In the next, we will give some reasons for this important design issue. For a definition of hash tables and hash functions see [8].

Hashing is space efficient. Handling collisions by chaining [8] requires one additional pointer per bucket to the next entry of the same hash code. This is the only overhead. In contrast, tries allow on the one hand a very compact representation of words by storing common prefixes exactly once, while on the other hand trie nodes contain much more housekeeping information such as next/previous char, end-of-word flag, max/min length, etc. [2]

Hashing is fast. When using a large, i.e. realistic, vocabulary, we want to keep our lexical information on a mass-storage medium (e.g. hard-disc) in order to save main memory. This means that the number of accesses to the dictionary will be minimized for the reason of speed. Our tests have shown, that on complete word keys, the average number of collisions has only a length less than two if an adequate hash table size has been chosen. Sorting the entries of the dictionary in decreasing access frequency may lead to even better results, where a trie data structure always takes n storage accesses to find a word of length n.

Noisy input. When dealing with incomplete words, we have to rely on partial hash information. This requires additional hash tables and hash functions which compute their hash codes, for instance, by the beginning or the end of the spelling or even by other features such as length or shape of a word (e.g. word envelopes [14]). Here the problem is to develop appropriate hash functions. Once a hash code has been calculated, the dictionary system can easily traverse its collision chains and pass all matching entries to the calling application. Again, a trie needs a lot more storage accesses to build a list of matching words (e.g. recursively traversing a subtree by a depth-first search).

Hash tables are static. When creating hash tables, respective hash function values are computed in advance. That implies we are forced to develop clever hash functions dealing with noisy input, e.g. using a half-word or significant characters of a word as hash information. Thus, the drawback of a hashing-based architecture is the need for an initial generation process. But at the same time such a dictionary initialization reveals some further interesting advantages:

• The source dictionaries are kept in an easy editable (ASCII-) format providing good maintenance and exchange,

• they are separated in different files (modularity) and

• the dictionary can easily be adjusted to another domain or another purpose.

After all, we consider hashing as the best technique to meet our goal handling the above system restrictions.

4 View concept

Dictionary access can be seen as post-processing the results of character recognition where the input is often noisy. Typically, character recognition is capable to provide the dictionary look-up routine with further information about the current word hypothesis, e.g. the word was found in the address part of the letter.

To make use of this kind of knowledge we decided to define *views*. Such views separate our dictionary into several parts each carrying words of different attributes or features. For instance, we can define a view on the length of a word, a view on all logical objects of a document (e.g. sender, recipient, date, body of a letter) as well as the view on the part-of-speech (e.g. noun, verb, adjective, etc.).

There are several ways to solve the problem of organizing views:

1) First, we could keep multiple copies of our dictionaries, each being created by another view—but this method causes a lot of redundant information when words belong to many views.

2) Next, we could physically split our dictionary into several views. This will result in a large number of small subdictionaries each of them holding words which are unique on all of our views, e.g. a subdictionary containing all nouns of word length 5 within an address. However, when looking for a word without any view identification, we have to consult all the different subdictionaries successively to find possible word candidates or to reject a word hypothesis. A second problem arises from the fact that the intersection of views is in general not empty (e.g. views on logical objects of a document).

3) As a solution, we decided to *virtually* divide our dictionary into several parts, called *virtual views*. All corresponding information of a word has to be kept once thus proving minimal redundancy and easy maintenance. Words belonging to one single view are linked together by a separate collision chain. This structure requires n*m additional pointers per entry where n is the number of different hash functions for redundant hash addressing and m is the number of the views defined.

As shown in Fig. 3, we use three distinct hash functions (H1, H2, H3) for indirect hashing. Consequently, three respective hash tables are provided. While hash function *H1* primarily deals with complete input, *H2* applies the left half of the input word for dictionary access, or *H3* applies the right half, respectively [5]. For H2 and H3, however, the hash tables refer to separate view entry tables each giving the first entries of a view.

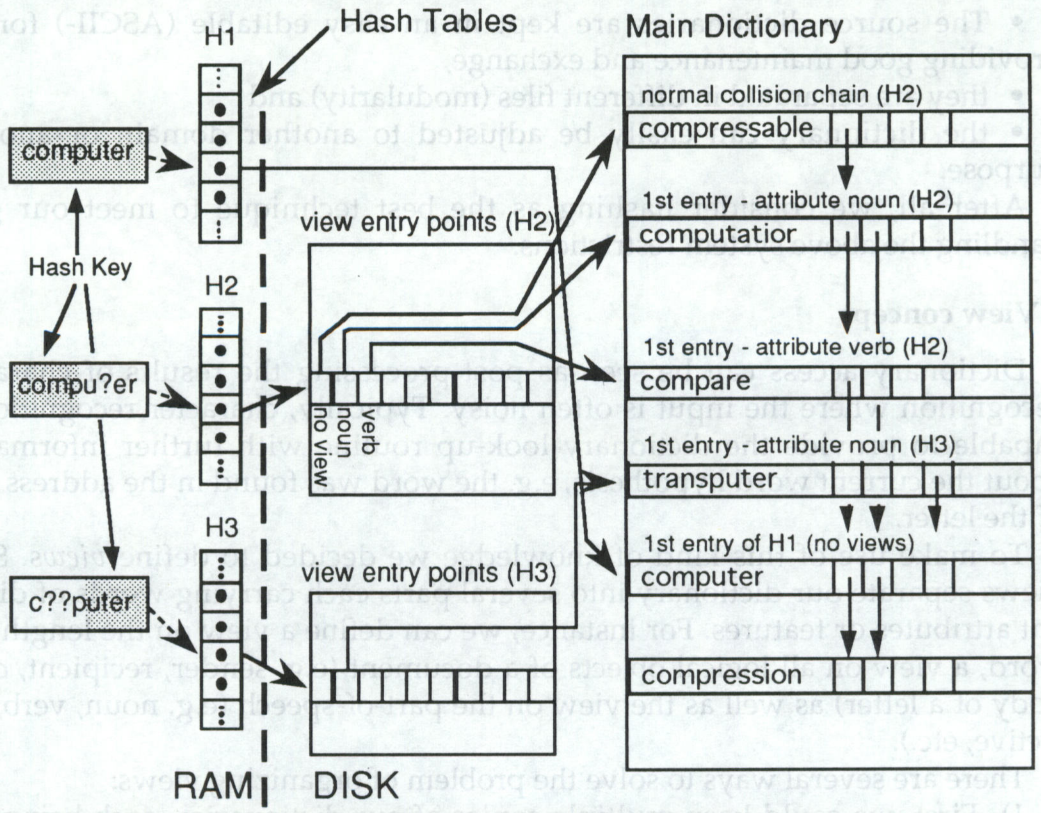


Fig. 3: Architecture of hashing-based dictionary

For example: The regular expression "c[oa]mpu[tf]?r" is passed to the dictionary and some other specialist, e.g. a parser, tells us that we are looking for a noun [3]. Neither the beginning nor the end of the word (hash information of 4 letters) are accurately recognized, but the beginning includes less alternatives ("comp", "camp") than the end following a hypothesize and test strategy. Now, we have to look within the collision chains of "comp" and "camp" (H2) to identify a noun matching "c[oa]mpu[tf]?r". Look-up in the view entry table of H2 identifies the first noun in the collision chain of the view "noun". Now, traversing all the corresponding collision pointers and matching the keys against the input pattern yields the hypothesis "computer".

In principle, we distinguish between two types of virtual views, static views and dynamic ones. Both are now explained in more detail.

4.1 Static views

So far, we have motivated the intention and mechanisms of *static views*. "Static" means that the definition of one of these views causes the *mandatory* allocation of a fixed (or static) amount of memory for every entry of the dictionary, regardless whether a word belongs to a view or not (Fig. 4). This makes static views expensive in terms of disk usage.

[3] The example also holds for other contextual restrictions such as view "address" or length of word.

For a dictionary of about 100,000 words, two hash functions using static views and a pointer size of four byte almost consume one megabyte of secondary storage per static view (100,000*2*4 byte). However, this view can be reused for several chains when the words of the different chains are disjoint, i.e. have no common elements. Typical examples of static views are the different attributes of part-of-speech of a word or a view on different word lengths.

The major advantage of static views is the reduction of disk accesses since only words within a specified view will be found when traversing the chain. This leads to a reduction of search space and accelerates search time dramatically. Some results mirroring these performance improvements are reported in Section 7 of this paper.

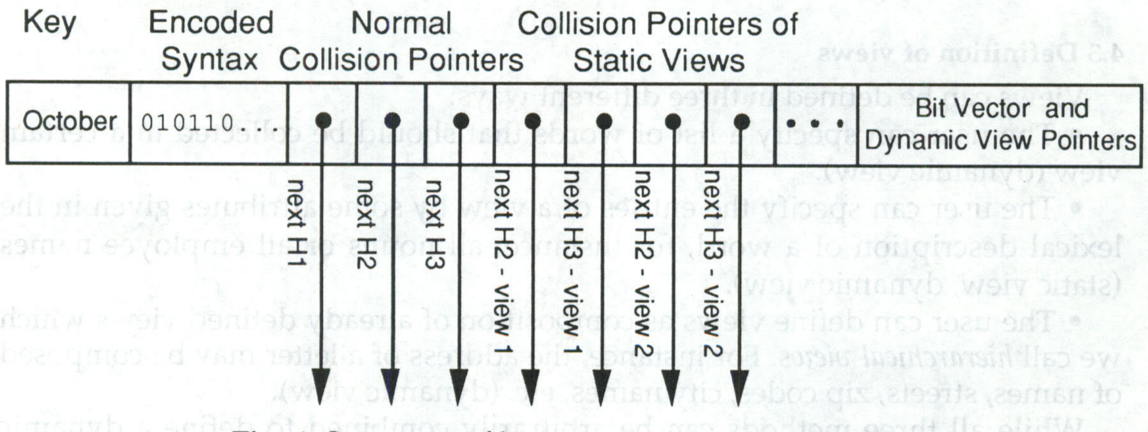


Fig. 4: Structure of lexicon entry w.r.t. static view

4.2 Dynamic views

In addition, we developed another class of views, called *dynamic* or *conceptual views*. These views join arbitrary sets of words together, regardless of their cardinality. Dynamic views need one reference pointer exclusively for those words that belong to a view. A bit vector which has to be kept for every entry signals the membership of a word to any of the defined dynamic views (Fig. 5). In principle, dynamic views are independent of hash functions, i.e. spelling of word keys [4].

The purpose of dynamic views is either to get a list of all words in a view solely by specifying the view's name (e.g. give me all employee names) or to get a list of views in which a certain word is defined. They were primarily designed to meet the special needs of text analysis as well as to realize views on logical objects of a document. A hierarchical definition of dynamic views is also possible.

[4] We use an AVL-tree accessing the first entry of such a view.

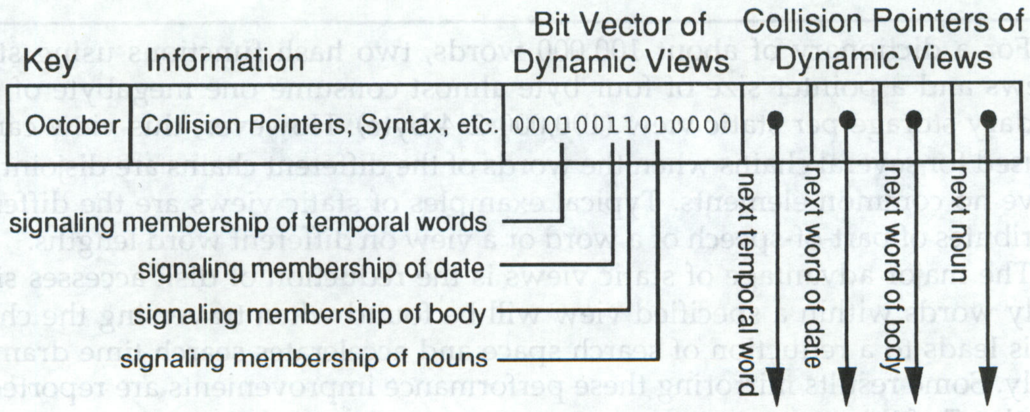

Key
Information
Bit Vector of Dynamic Views
Collision Pointers of Dynamic Views
October
Collision Pointers, Syntax, etc.
0001001101000
signaling membership of temporal words
signaling membership of date
signaling membership of body
signaling membership of nouns
next temporal word
next word of date
next word of body
next noun

Fig. 5: Structure of lexicon entry w.r.t. dynamic view

4.3 Definition of views

Views can be defined in three different ways:

• The user can specify a list of words that should be collected in a certain view (dynamic view).

• The user can specify the entries of a view by some attributes given in the lexical description of a word, for instance, all nouns or all employee names (static view, dynamic view).

• The user can define views as composition of already defined views which we call *hierarchical views*. For instance, the address of a letter may be composed of names, streets, zip codes, city names, etc. (dynamic view).

While all three methods can be arbitrarily combined to define a dynamic view, only the second one can be applied to static views. In other words, a hierarchy of static views is not possible.

5 System implementation

5.1 Dictionary generator

As mentioned above, we can take a set of source dictionaries as input for the generation process. These sources are either word lists or can include simple lexical information about a word such as part-of-speech. Actually, the source dictionaries involve German nouns, verbs and adjectives, typical words of addresses (employees, titles, German cities, countries, etc.), German abbreviations, domain-specific words for business letters and some other word lists for logical objects (e.g. date).

Furthermore, we have some profiles (see Fig. 6), which tell the generator how to build the different static and dynamic views, how to encode/decode the syntactic information and how to define text phrases being an additional feature.

After the successful termination, the generator has created some new dictionary files which keep all relevant information of the hash tables and first entry addresses of views in different hash chains as well as the file containing the main, virtually partitioned dictionary. Another file, called *system info file*, keeps the path names of the profiles, the path names of the generated files and the lengths of the hash tables. When using the dictionary, only the latter file has to be specified.

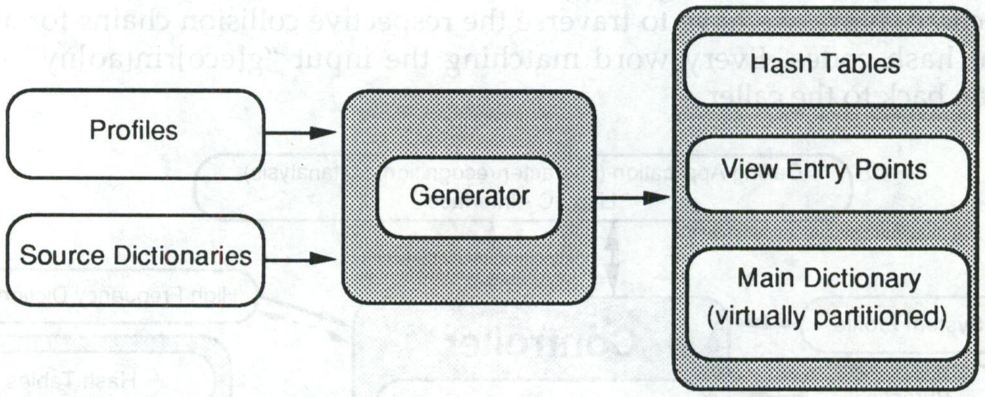


Fig. 6: Generation phase of dictionary

The profiles are needed during run time and must not be edited unless a new generation process is restarted. While Fig. 6 summarizes dictionary generation, Fig. 7 reveals the run time system, i.e. the dictionary controller.

5.2 Dictionary controller

After termination of the generation process, the dictionary system can be used as follows:

1) Finding a word by its spelling and static view identification:

• Search argument is a string similar to a regular expression of UNIX (csh, ed) representing the actual recognized word hypothesis. The dictionary controller now examines this string for wildcards ('?', '*') as well as alternatives at some word positions indicated by '[' and ']'

(a) If the word was entirely recognized, a search with complete hash information is initiated (H1 in Fig. 2). In this case the corresponding hash code is calculated by interpreting letters as digits of base 30 (cardinality of German alphabet including German umlauts and s-zet). Since we do not expect long collision chains, the view information is not considered in this case.

(b) Searching for an incomplete word is explained by an example: Assume the input argument is "g[eco]rm[ao]ny" ADDRESS. The input word contains alternatives at position 2 and 5. The total length of the word is well defined since no '*'-wildcard was found. The characteristic string for the search has length 3 (= minimum of (word length 7 div 2) and maximum half-word constant 4). Evaluation of the word beginning "g[eco]r" results in three distinct hash codes: H("ger"), H("gcr") and H("gor"). Evaluation of the word end "[ao]ny" results in only two hash codes: H("any") and H("ony"). Because of the smaller number of alternatives, the dictionary controller decides to use the end of the input word for subsequent search.

• The additional parameter ADDRESS tells the system that it should take the collision chain of the view named ADDRESS containing all words that might occur within the address part of the letter that is provided by logical labeling [1].

• Now, the dictionary system is initialized for the search: the search mode—either complete word, beginning or end of word—and a static view have been specified. Next, we have to traverse the respective collision chains for all relevant hash codes. Every word matching the input "g[eco]rm[ao]ny" is then given back to the caller.

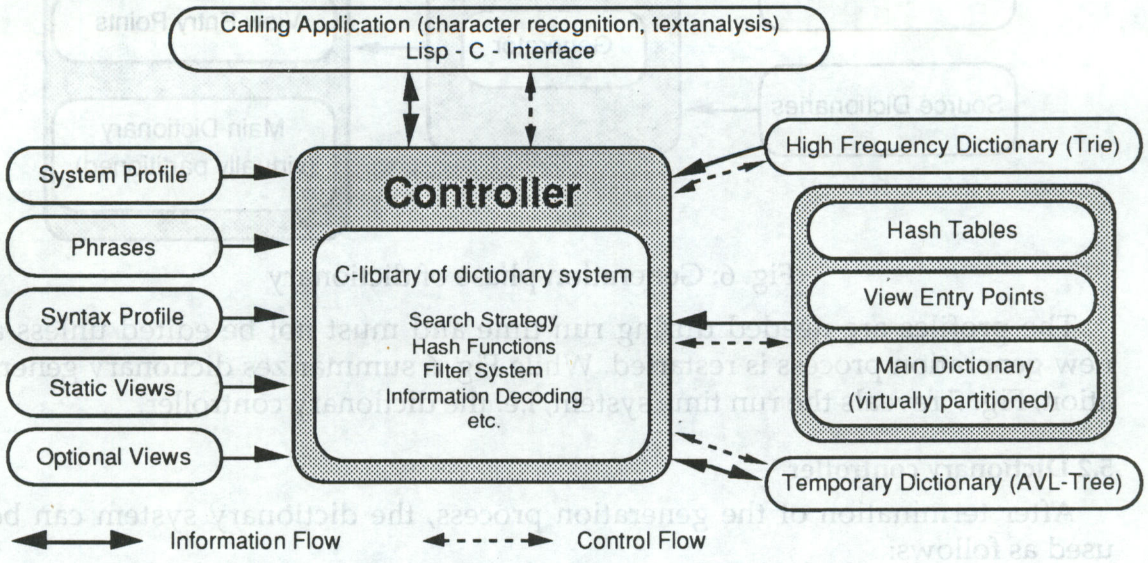


Fig. 7: Run-time system of dictionary

2) Finding words by specifying a dynamic view:

• The calling application invokes the dictionary controller for search by naming a dynamic view.

• Consultation of the internal (AVL-)search tree gives the entry points as well as the ordinal of this dynamic view in the corresponding bit vector.

3) Traversing the dictionary and handing back information:

Main parts of the text recognition as well as text analysis components were implemented in Common Lisp. For reasons of speed and limited resources the entire dictionary has been implemented using C. This requires an interface which is able to call C functions from Lisp. The respective functions of the Common Lisp interface allocate memory for the arguments passed and even strings are transformed which makes the interface to some kind of bottleneck with respect to information flow.

Due to this fact only matching words are passed through the interface. The user traverses the dictionary in a step by step manner, always getting the next matching word or the next view member, respectively.

If the calling application addresses words that are members of a specified dynamic view, this can be done by using filters before passing the interface.

5.3 Filter

A special filter concept has been designed to get disjunctions or conjunctions of views without evaluating large word lists. The system provides several filters that can be set for one or more dynamic views. Setting one filter several times causes a conjunction of the corresponding views, whereas setting different filters results in a disjunction. Filters might be negated: in this case, only words that are not within the specified view will pass through it.

If any filter is set, at least one of them has to be passed before a word is given back (to the Lisp process via a C-Lisp foreign function interface). According to dynamic views, the essential mask information for the interpretation of filters are bit vectors that are provided for every lexicon entry.

6 Related work

While general hash table methods have extensively been developed over the last two decades and were well explored, there is a pressing need for sophisticated hashing which is tailored and specialized to improve character recognition. Here, only a few papers can be found ([9], [12], [16]).

While [12] also proposes a twofold hash code access via the left and right half of a word, our approach is much more flexible allowing the integration of different hash functions as well as the definition of views. [9] presents a scheme of redundant hashing based on trigrams as significant word features. This procedure was primarily developed for dealing with distorted phonemic strings in speech recognition.

In [6], [10] and [11] a concept is shown partitioning the dictionary physically either by frequency or by word contents (technical words, multi-lingual, etc.). Other dictionary concepts are based on conventional, i.e. relational, database technologies [7]. A hierarchical structuring of the dictionary for address recognition can be found in [13]. As far as we are concerned, there does not exist a comparable dictionary approach based on virtual views yet.

7 Results

All system components of the dictionary are fully implemented in C on Sun SPARCstation with exception of the external Common Lisp interface. The source dictionaries are ASCII-files either containing word lists or simple lexical information.

To get an idea of the performance of our dictionary, we performed some comparative tests showing the run time for specific regular input patterns. The tests give an impression of the dictionary access time in comparison with standard UNIX utilities. The competitive processes were "agrep" (a very fast UNIX tool for approximate string matching [19]), "look" (a UNIX tool for finding words or lines in an alphabetically sorted list) and our dictionary system. The word list we used has either a size of 160,086 or 103,023 German words. The results are illustrated in the following tables.

access time [msec]			
word pattern	*agrep*	dictionary	matches
?nlo[ec]k*d	720,0	87,5	1
b?[gqj][eco]ist*rt	770,0	245,8	1
be*orschun[gq]	830,0	115,8	1
drei??n[hb]al??ache	670,0	12,5	1
e[ij][rnm]zel[hkb]?bine	670,0	3,3	1
???chleicht	660,0	238,3	1
geb[ax]eu[da][ec]sicheru??	650,0	6,6	1
gemi*t	690,0	21,6	5
[bhk]erru*rten	740,0	625,8	1
*otten[hk]rieg?	1.170,0	258,3	1
??editi*	750,0	1.431,6	8
turfil	710,0	32.920,0	4
ober*ung[xs]ger[ji][ec]ht	1.550,0	15,8	1

Table 1

access time [msec]		
word pattern	*look*	dictionary
anlockend	4,0	0,167
begeistert	4,0	0,250
berufsforschung	3,0	0,333
dreieinhalbfache	4,0	1,667
einzelkabine	6,0	1,333
erschleicht	4,0	1,583
gebaeudesicherung	4,0	1,667
gemildert	5,0	1,333
herruehrten	2,0	0,167
hugenottenkriege	1,0	0,333
kreditiert	2,0	0,333
kulturfilm	3,0	0,083
oberverwaltungsgericht	5,0	1,583

Table 2

In Table 1, dictionary access versus the "agrep" command reveals a maximal improvement by a factor of 200 (fifth word pattern). However, in some rare cases when the input word begins and ends with a wildcard of arbitrary length, our dictionary fails (second last word pattern). In comparison to the "look" 5 command (Table 2), the maximal improvement of access time is about a factor of 16 (second last word pattern).

5 Note that "look" does not allow regular expressions as input.

access time [msec]				
word pattern	without view	matches	with view	matches
[rn][eco][ce][hk][eo]nanl*	60,0	1	37,5	1
[bh][eco]si[ft]*[oc]sigk[eco]i[ft]	10,8	1	5,8	1
c[oa]mpu[tf]?r	7,5	1	5,0	1
d[eco]s[oc]?[oc]ris?[ft]ion	226,7	1	207,5	1
dipl[oc]m?[ft][eco]nsc[kh]r[eco]i[bh][ft]*[kh]	5,8	1	4,2	1
f[eco]ri[eco]n?us[ft]?u*[kh]	10,0	1	9,2	1
g[eco]sc[kh]ickli*k[eco]i[ft]	36,7	2	16,7	1
gul?sc[kh]k?n[oc]n[eco]	9,2	1	9,2	1
h?ndf[eco]g[eco]r	85,8	1	90,8	1
hin[ft][eco]rl?ss[eco]nsc[kh]?f[ft]	15,8	1	8,3	1
i[oc]n[eco]nw?nd[eco]rung	399,2	1	260,0	1
kaiser*	12,5	28	6,7	18
kr?nk[eco]ng[eco]sc[kh]ic[kh][ft][eco]	70,0	1	65,0	1
l[eco]uc[kh][ft]r?k[eco][ft][eco]	16,7	1	6,7	1
me[eo]r[ec]sl[eco]uc[kh][ft][eco]n	8,3	1	7,5	1
m?nito*	83,3	2	47,5	2
n?[bh][eco]ls[ft]r?ng	492,5	1	463,3	1
s[eco]l[bh]s[ft]kos[ft][eco]ns[eco]nkung	75,8	1	64,2	1
s[ft]?d[ft]s[ft]r[eco]ic[kh][eco]r	595,0	1	410,8	1
ur[ft]i[eco]rc[kh][eco]n	8,3	1	5,8	1
v[eco]r?rsc[kh]ung	181,7	1	155,8	1
z[eco]n[ft]im[eco][ft][eco]rm?ss	12,5	1	7,5	1

Table 3

dictionary generation [sec]		
dictionary size	without any views	one static view
1.000	4,2	7,6
2.000	10,0	15,6
3.000	27,9	45,5
4.000	37,8	64,4
5.000	51,4	79,6
7.500	92,8	139,1
10.000	145,0	211,1
15.000	285,4	401,5
20.000	498,2	696,9
107.427	—	17.222,3

Table 4

Another test reports the speed-up when searching for a pattern applying the concept of virtual views (Table 3). Here, we compare a normal search without using views and a search where the system knew about the word category. We took a dictionary of 103,023 German words including a static view over the part-of-speech. The results are given in Table 3 indicating a slight improvement of search time as well as the reduction of word alternatives (word pattern "kaiser*" is a *noun*).

Generation time for different dictionaries is given in Table 4 where the complexity is $O(n^2)$ (n = number of words in dictionary). The diagrams in the Appendix present the mean dictionary access time per word over a sample of about 100 business letters using our own OCR (dictionary size is 10,000). In addition, the last diagram compares *dictionary access/word in average* for two dictionary sizes (10,000 and 100,000).

8 Conclusion and future work

In this paper, we presented a new approach of partitioning large dictionaries by virtual views. The main idea is that additional knowledge sources of text recognition and text analysis can be applied for fast dictionary access by pruning the search space. Our results have shown that this approach speeds up dictionary look-up for complete as well as for noisy input drastically. Moreover, we have implemented two main system components: the dictionary generator and the dictionary controller. The controller allows a flexible integration of different search heuristics including approximate string matching techniques.

While testing our dictionary, we decided to integrate some additional features to the current implementation:

- Currently, the time complexity of dictionary generation is $O(n^2)$ because all collision chains of the main dictionary are traversed for finding the right end to put in a new dictionary entry (n = number of words in dictionary). This can be improved to $O(n)$ by the recording of last entries of the collision chains. Incidentally, the generation process of a virtually partitioned dictionary including one static view "part-of-speech" with four attributes, sixteen virtual views and a size of 103,023 word entries takes about five hours on a Sun SPARCstation.
- Dynamic extension of the existing dictionary by new entries without starting the time consuming generation process once again.
- Definition of links between dictionary entries. We would like to provide some additional storage to create links from one entry to another, for example, combining synonyms, phrases or variants.
- Our matching component is only capable of exact string matching, i.e. words can only be found if each set of character alternatives includes the right letter. Thus, misspelled words are the problem. Consequently, the integration of an approximate string matching component, e.g. dealing with edit operations of exactly one character (insertion, deletion, substitution of one character; cf. [17]), is one of our most important future improvements.

Acknowledgements

We would like to thank Dr. George Nagy and Hans-Günther Hein for valuable discussions and their careful reading of the first draft of this paper.

Bibliography

[1] A. Dengel, R. Bleisinger, R. Hoch, F. Fein, F. Hönes. From Paper to Office Document Standard Representation. *IEEE Computer Magazine,* vol. 25, no. 7, July 1992, 63-67.

[2] A. Dengel, A. Pleyer, R. Hoch. Fragmentary String Matching by Selective Access to Hybrid Tries. Proc. of *11th International Conference on Pattern Recognition,* The Hague, Sept. 1992, vol. II, 149-153.

[3] D. G. Elliman, I. T. Lancaster. A Review of Segmentation and Contextual Analysis Techniques for Text Recognition. *Pattern Recognition,* vol. 23, no. 3/4, 1990, 337-346.

[4] M. D. Harris. *Introduction to Natural Language Processing.* Reston Publishing Company Inc., Reston, Virginia, 1985.

[5] R. Hoch. Hybrid Structured Dictionary for Improving Text Recognition. Proc. of *IAPR Workshop on Machine Vision Applications (MVA'92),* Tokyo, Japan, December 7-9, 1992, 295-298.

[6] R. Hoch, M. Malburg. Designing a Structured Lexicon for Document Image Analysis. Proc. of *Seventh Intl. Summer School on Information Technologies&Programming,* Sofia, July 1992, 71-76.

[7] N. M. Ide, J. Veronis, J. Le Maitre. Outline of a Database Model for Electronic Dictionaries. Proc. of *RIAO 91 Intelligent Text and Image Handling, Barcelona,* April 2-5, 1991, vol. 1, 375-393.

[8] D. E. Knuth. *The Art of Computer Programming, vol. III: Sorting and Searching.* Addison-Wesley, Reading, Mass., 1973.

[9] T. Kohonen, E. Reuhkala. A very fast associative method for the recognition and correction of misspelt words, based on redundant hash addressing. Proc. of *Fourth Intl. Joint Conference on Pattern Recognition,* Kyoto, Japan, Nov. 7-10, 1978, 807-809.

[10] J. L. Peterson. Computer Programs for Detecting and Correcting Spelling Errors. *Communications of the ACM,* vol. 23, no. 12, December 1980, 676-687.

[11] H. Richy, P. Frison, E. Picheral. Multilingual String-to-String Correction in Grif, a Structured Editor. Proc. of *Electronic Publishing '92,* Lausanne, Cambridge University Press, 1992, 183-198.

[12] J. Schürmann. Multifont Word Recognition System. *IEEE Transactions on Computers,* vol. c-27, no. 8, August 1978.

[13] K. Seino, Y. Tanabe, K. Sakai. A linguistic post processing based on word occurrence probability. In: *From Pixels to Features III: Frontiers in Handwriting Recognition,* S. Impedovo, J. C. Simon (eds.), Elsevier Science Publications B. V., 1992.

[14] R. M. K. Sinha. On Partitioning a Dictionary for Visual Text Recognition. *Pattern Recognition,* vol. 23, no. 5, 1990, 497-500.

[15] S. N. Srihari, J. J. Hull, R. Choudhari. Integrating Diverse Knowledge Sources in Text Recognition. *ACM Transactions on Office Information Systems*, vol. 1, no. 1. January 1983, 68-87.

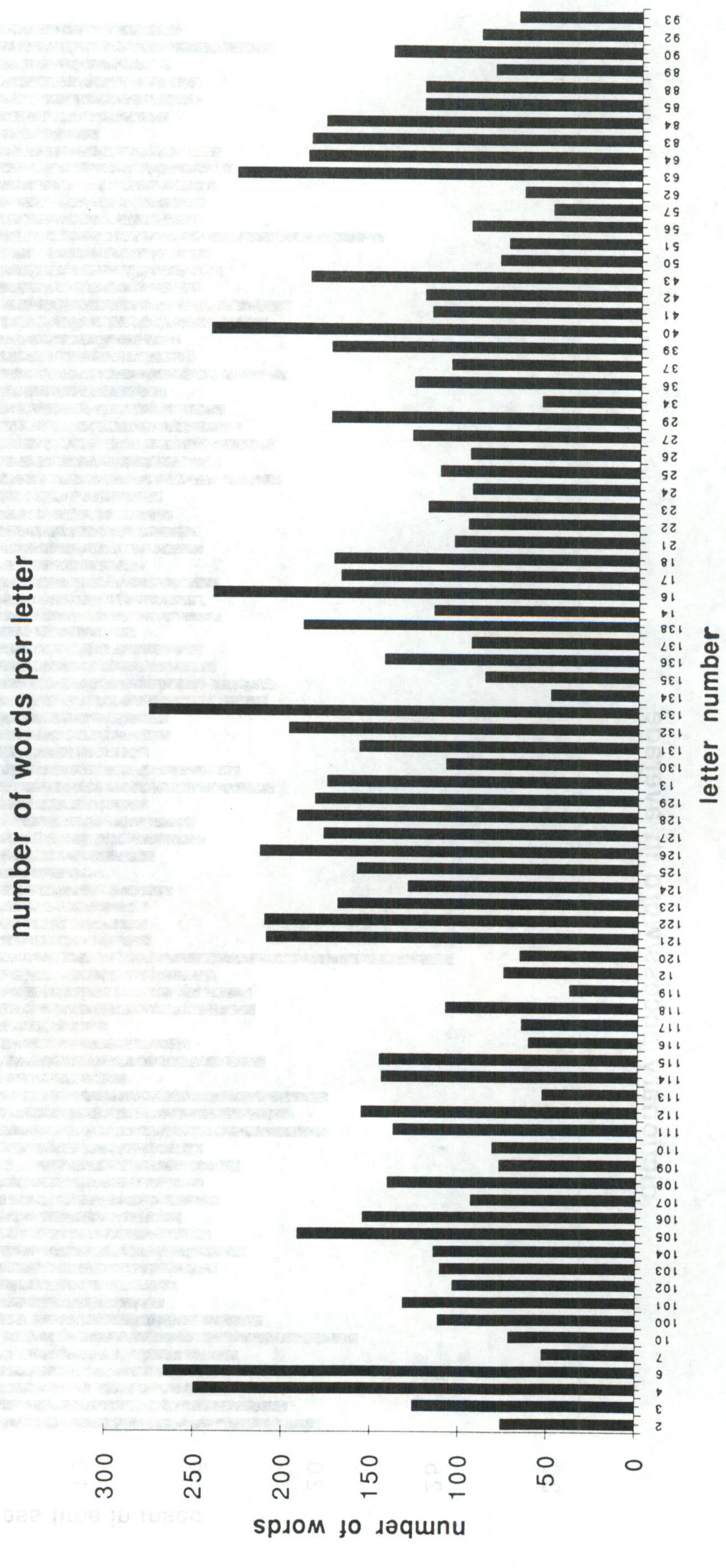
[16] H. Takahashi, N. Itoh, T. Amano, A. Yamashita. A Spelling Correction Method and its Application to an OCR System. *Pattern Recognition*, vol. 23, no. 3/4, 1990, 363-377.

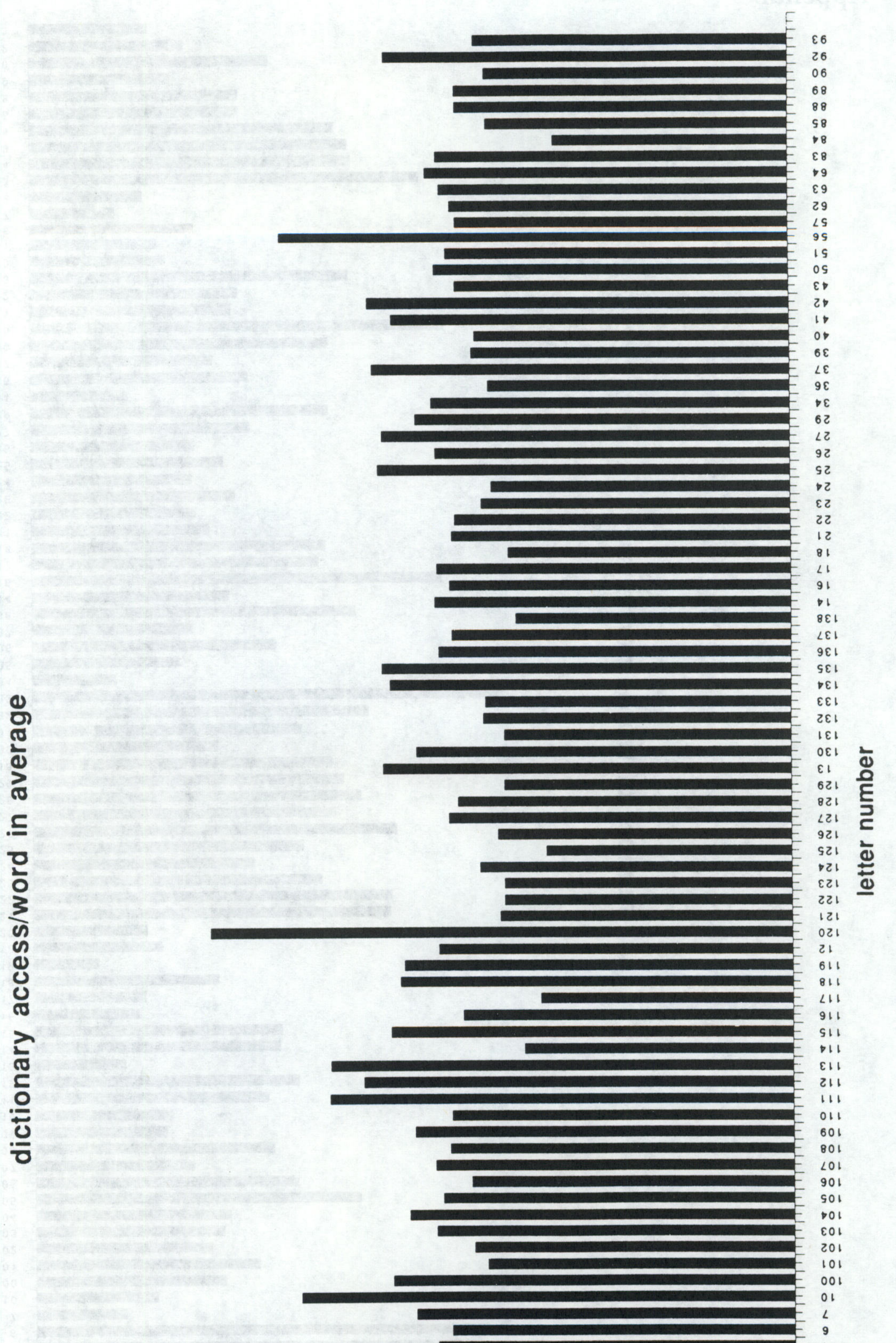
[17] R. A. Wagner, M. J. Fischer. The String-to-String Correction Problem. *Journal of the Association for Computing Machinery*, vol. 21, no. 1, January 1974, pp. 168-173.

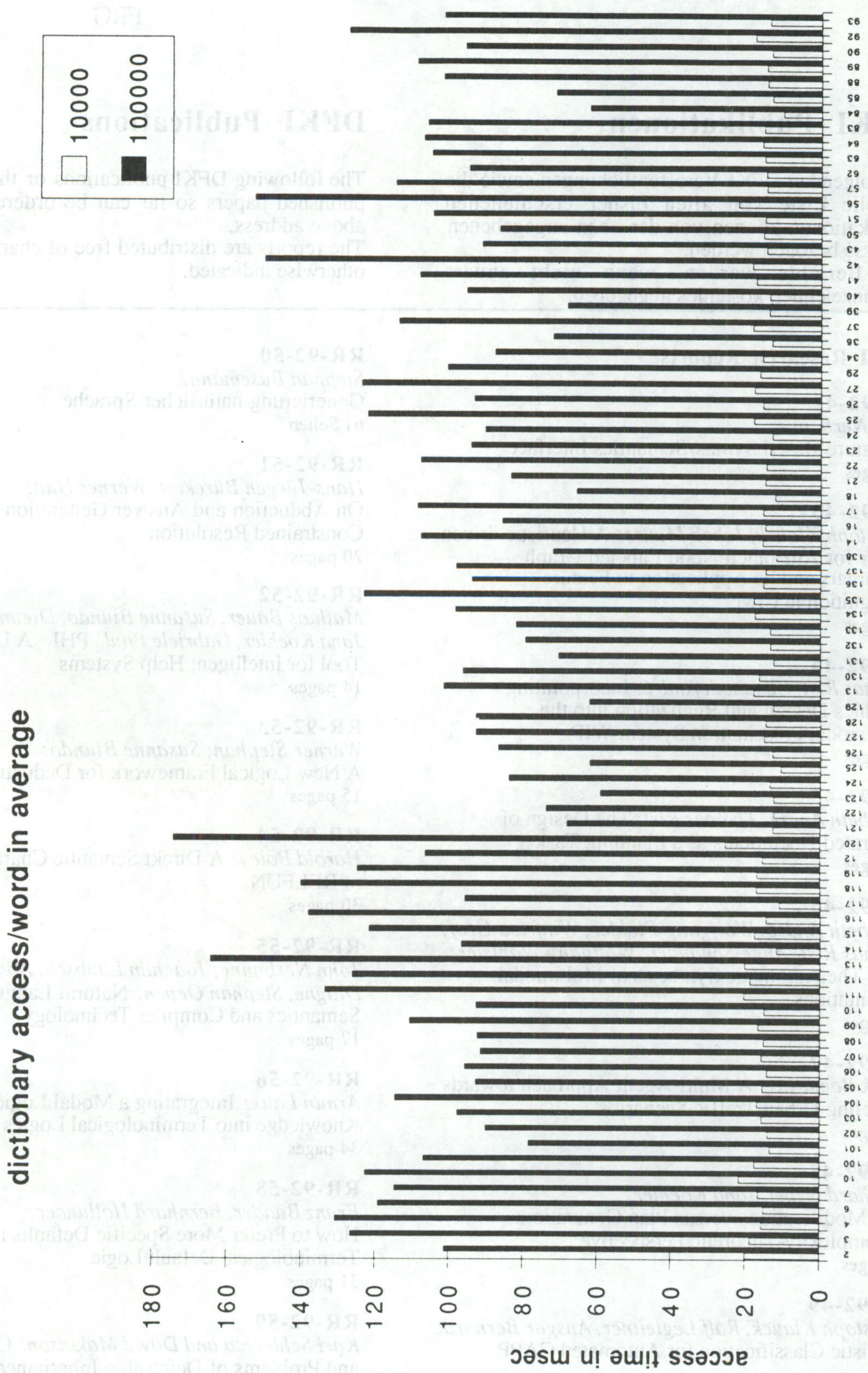
[18] C. J. Wells et al., Fast Dictionary Look-Up For Contextual Word Recognition. *Pattern Recognition*, vol. 23, no. 5, 1990, 501-508.

[19] S. Wu, U. Manber. Fast text searching allowing errors. *Communications of the ACM*, vol. 35, no. 10, October 1992, 83-91.

Appendix


number of words per letter
number of words
letter number
0
50
100
150
200
250
300
2
3
4
6
7
10
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
12
120
121
122
123
124
125
126
127
128
129
13
130
131
132
133
134
135
136
137
138
14
16
17
18
21
22
23
24
25
26
27
29
34
36
37
39
40
41
42
43
50
51
56
57
62
63
64
83
84
85
88
89
90
92
93


dictionary access/word in average
access time in msec
0
5
10
15
20
25
30
letter number
2 3 4 6 7 10 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 12 120 121 122 123 124 125 126 127 128 129 13 130 131 132 133 134 135 136 137 138 14 16 17 18 21 22 23 24 25 26 27 29 34 36 37 39 40 41 42 43 50 51 56 57 62 63 64 83 84 85 88 89 90 92 93


dictionary access/word in average
access time in msec
letter number
10000
100000
0
20
40
60
80
100
120
140
160
180

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse bezogen werden.
Die Berichte werden, wenn nicht anders gekennzeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far can be ordered from the above address.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-92-42
John Nerbonne:
A Feature-Based Syntax/Semantics Interface
19 pages

RR-92-43
Christoph Klauck, Jakob Mauss: A Heuristic driven Parser for Attributed Node Labeled Graph Grammars and its Application to Feature Recognition in CIM
17 pages

RR-92-44
Thomas Rist, Elisabeth André: Incorporating Graphics Design and Realization into the Multimodal Presentation System WIP
15 pages

RR-92-45
Elisabeth André, Thomas Rist: The Design of Illustrated Documents as a Planning Task
21 pages

RR-92-46
Elisabeth André, Wolfgang Finkler, Winfried Graf, Thomas Rist, Anne Schauder, Wolfgang Wahlster:
WIP: The Automatic Synthesis of Multimodal Presentations
19 pages

RR-92-47
Frank Bomarius: A Multi-Agent Approach towards Modeling Urban Traffic Scenarios
24 pages

RR-92-48
Bernhard Nebel, Jana Koehler:
Plan Modifications versus Plan Generation: A Complexity-Theoretic Perspective
15 pages

RR-92-49
Christoph Klauck, Ralf Legleitner, Ansgar Bernardi:
Heuristic Classification for Automated CAPP
15 pages

RR-92-50
Stephan Busemann:
Generierung natürlicher Sprache
61 Seiten

RR-92-51
Hans-Jürgen Bürckert, Werner Nutt:
On Abduction and Answer Generation through Constrained Resolution
20 pages

RR-92-52
Mathias Bauer, Susanne Biundo, Dietmar Dengler, Jana Koehler, Gabriele Paul: PHI - A Logic-Based Tool for Intelligent Help Systems
14 pages

RR-92-53
Werner Stephan, Susanne Biundo:
A New Logical Framework for Deductive Planning
15 pages

RR-92-54
Harold Boley: A Direkt Semantic Characterization of RELFUN
30 pages

RR-92-55
John Nerbonne, Joachim Laubsch, Abdel Kader Diagne, Stephan Oepen: Natural Language Semantics and Compiler Technology
17 pages

RR-92-56
Armin Laux: Integrating a Modal Logic of Knowledge into Terminological Logics
34 pages

RR-92-58
Franz Baader, Bernhard Hollunder:
How to Prefer More Specific Defaults in Terminological Default Logic
31 pages

RR-92-59
Karl Schlechta and David Makinson: On Principles and Problems of Defeasible Inheritance
13 pages

RR-92-60
Karl Schlechta: Defaults, Preorder Semantics and Circumscription
19 pages

RR-93-02
Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, Thomas Rist: Plan-based Integration of Natural Language and Graphics Generation
50 pages

RR-93-03
Franz Baader, Berhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, Enrico Franconi: An Empirical Analysis of Optimization Techniques for Terminological Representation Systems
28 pages

RR-93-04
Christoph Klauck, Johannes Schwagereit: GGD: Graph Grammar Developer for features in CAD/CAM
13 pages

RR-93-05
Franz Baader, Klaus Schulz: Combination Techniques and Decision Problems for Disunification
29 pages

RR-93-06
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: On Skolemization in Constrained Logics
40 pages

RR-93-07
Hans-Jürgen Bürckert, Bernhard Hollunder, Armin Laux: Concept Logics with Function Symbols
36 pages

RR-93-08
Harold Boley, Philipp Hanschke, Knut Hinkelmann, Manfred Meyer: COLAB: A Hybrid Knowledge Representation and Compilation Laboratory
64 pages

RR-93-09
Philipp Hanschke, Jörg Würtz: Satisfiability of the Smallest Binary Program
8 Seiten

RR-93-10
Martin Buchheit, Francesco M. Donini, Andrea Schaerf: Decidable Reasoning in Terminological Knowledge Representation Systems
35 pages

RR-93-11
Bernhard Nebel, Hans-Juergen Buerckert: Reasoning about Temporal Relations: A Maximal Tractable Subclass of Allen's Interval Algebra
28 pages

RR-93-12
Pierre Sablayrolles: A Two-Level Semantics for French Expressions of Motion
51 pages

RR-93-13
Franz Baader, Karl Schlechta: A Semantics for Open Normal Defaults via a Modified Preferential Approach
25 pages

RR-93-14
Joachim Niehren, Andreas Podelski, Ralf Treinen: Equational and Membership Constraints for Infinite Trees
33 pages

RR-93-15
Frank Berger, Thomas Fehrle, Kristof Klöckner, Volker Schölles, Markus A. Thies, Wolfgang Wahlster: PLUS - Plan-based User Support Final Project Report
33 pages

RR-93-16
Gert Smolka, Martin Henz, Jörg Würtz: Object-Oriented Concurrent Constraint Programming in Oz
17 pages

RR-93-17
Rolf Backofen: Regular Path Expressions in Feature Logic
37 pages

RR-93-18
Klaus Schild: Terminological Cycles and the Propositional μ-Calculus
32 pages

RR-93-20
Franz Baader, Bernhard Hollunder: Embedding Defaults into Terminological Knowledge Representation Formalisms
34 pages

RR-93-22
Manfred Meyer, Jörg Müller: Weak Looking-Ahead and its Application in Computer-Aided Process Planning
17 pages

RR-93-23
Andreas Dengel, Ottmar Lutzy: Comparative Study of Connectionist Simulators
20 pages

RR-93-24
Rainer Hoch, Andreas Dengel: Document Highlighting — Message Classification in Printed Business Letters
17 pages

RR-93-25
Klaus Fischer, Norbert Kuhn: A DAI Approach to Modeling the Transportation Domain
93 pages

RR-93-26
Jörg P. Müller, Markus Pischel: The Agent Architecture InteRRaP: Concept and Application
99 pages

RR-93-27
Hans-Ulrich Krieger:
Derivation Without Lexical Rules
33 pages

RR-93-28
Hans-Ulrich Krieger, John Nerbonne, Hannes Pirker: Feature-Based Allomorphy
8 pages

RR-93-29
Armin Laux: Representing Belief in Multi-Agent Worlds viaTerminological Logics
35 pages

RR-93-33
Bernhard Nebel, Jana Koehler:
Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis
33 pages

RR-93-34
Wolfgang Wahlster:
Verbmobil Translation of Face-To-Face Dialogs
10 pages

RR-93-35
Harold Boley, François Bry, Ulrich Geske (Eds.):
Neuere Entwicklungen der deklarativen KI-Programmierung — *Proceedings*
150 Seiten
Note: This document is available only for a nominal charge of 25 DM (or 15 US-$).

RR-93-36
Michael M. Richter, Bernd Bachmann, Ansgar Bernardi, Christoph Klauck, Ralf Legleitner, Gabriele Schmidt: Von IDA bis IMCOD: Expertensysteme im CIM-Umfeld
13 Seiten

RR-93-38
Stephan Baumann: Document Recognition of Printed Scores and Transformation into MIDI
24 pages

RR-93-40
Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, Werner Nutt, Andrea Schaerf:
Queries, Rules and Definitions as Epistemic Statements in Concept Languages
23 pages

RR-93-41
Winfried H. Graf: LAYLAB: A Constraint-Based Layout Manager for Multimedia Presentations
9 pages

RR-93-42
Hubert Comon, Ralf Treinen:
The First-Order Theory of Lexicographic Path Orderings is Undecidable
9 pages

RR-93-45
Rainer Hoch: On Virtual Partitioning of Large Dictionaries for Contextual Post-Processing to Improve Character Recognition
21 pages

DFKI Technical Memos

TM-91-15
Stefan Busemann: Prototypical Concept Formation
An Alternative Approach to Knowledge Representation
28 pages

TM-92-01
Lijuan Zhang: Entwurf und Implementierung eines Compilers zur Transformation von Werkstückrepräsentationen
34 Seiten

TM-92-02
Achim Schupeta: Organizing Communication and Introspection in a Multi-Agent Blocksworld
32 pages

TM-92-03
Mona Singh:
A Cognitiv Analysis of Event Structure
21 pages

TM-92-04
Jürgen Müller, Jörg Müller, Markus Pischel, Ralf Scheidhauer:
On the Representation of Temporal Knowledge
61 pages

TM-92-05
Franz Schmalhofer, Christoph Globig, Jörg Thoben:
The refitting of plans by a human expert
10 pages

TM-92-06
Otto Kühn, Franz Schmalhofer: Hierarchical skeletal plan refinement: Task- and inference structures
14 pages

TM-92-08
Anne Kilger: Realization of Tree Adjoining Grammars with Unification
27 pages

TM-93-01
Otto Kühn, Andreas Birk: Reconstructive Integrated Explanation of Lathe Production Plans
20 pages

TM-93-02
Pierre Sablayrolles, Achim Schupeta:
Conlfict Resolving Negotiation for COoperative Schedule Management
21 pages

TM-93-03
Harold Boley, Ulrich Buhrmann, Christof Kremer:
Konzeption einer deklarativen Wissensbasis über recyclingrelevante Materialien
11 pages

TM-93-04
Hans-Günther Hein: Propagation Techniques in WAM-based Architectures — The FIDO-III Approach
105 pages

On Virtual Partitioning of Large Dictionaries for Contextual Post-Processing to Improve Character Recognition

Rainer Hoch

RR-93-45
Research Report