MVA '92 IAPR Workshop on Machine Vision Applications Dec. 7-9,1992, Tokyo

HYBRID STRUCTURED DICTIONARY FOR IMPROVING TEXT RECOGNITION

Rainer Hoch
German Research Center for Artificial Intelligence (DFKI)

P. O. Box 20 80, D-6750 Kaiserslautern, Germany
Phone: (++49) 631-205-3584, Fax: (++49) 631-205-3210, e-mail: hoch@dfki.uni-kl.de

ABSTRACT

This paper presents an approach for structuring large dictionaries and proposes respective access techniques. The dictionary is used as a post-processing tool to verify recognized word candidates dealing with different types of incomplete input.

Structuring a dictionary is attractive because search space is pruned enormously, especially when recognized words are rather noisy. For instance, in the context of addresses only a relative small set of words is relevant. Furthermore, the so-defined partitions can easily be exchanged when proceeding analysis from one part of a document to another (e.g., from address to body). Moreover, there are dictionaries including high-frequency words, common abbreviations, domain-specific words and a base dictionary for German.

Two complementary dictionary data structures, namely hash tables and tries, are used combining their inherent advantages to organize respective subdictionaries. Both approaches can deal with different types of word incompleteness. While hashing is most appropriate for fast look-up of well-recognized input or simple distortions, tries in combination with a special access matrix are able to handle very fragmentary input words, in particular when large prefixes or suffixes are missing. *

1 INTRODUCTION

Today, many researchers argue that traditional OCR has reached its frontiers. In contrast to classification of isolated characters, a new generation of OCR systems will apply contextual knowledge (e.g., lexicon, syntax, and semantics) for improving recognition accuracy [9].

In this paper we describe how lexical knowledge can be efficiently used to support document analysis resulting in a hybrid architecture of a structured dictionary. The intention of our document analysis system is to bridge the gap between traditional print products and the computer. Exemplarily, the structure and (partial) semantics of German business letters are analyzed. The system is model-driven based on the ODA platform (Office Document Architecture [6]), i.e., an international standard for the representation and exchange of documents is supported. The entire system includes several interlocked phases of analysis, including *layout extraction, logical labeling, text recognition,* and *(partial) text analysis*. Further details and experimental results of the system are given in [1].

The dictionary is a central component for our analysis system supporting and improving results of character recognition as well as enabling a partial document understanding. In the remainder of this paper, we will use terms *dictionary* and *lexicon* synonymously. Both designate a collection of words of a language or a special domain, possibly provided with simple statistic, morphologic and syntactic information.

In literature, there is a lack of smart approaches for designing large dictionaries that will improve character recognition. In Seino et al. [14] a knowledge processing (KP) method is shown. This KP method is applied by Toshiba OCR systems and comprises—among other knowledge sources—many word dictionaries being hierarchically structured, e.g., an address dictionary for hand-printed Japanese characters. Word occurrence probabilities stored in the dictionaries are also used to eliminate irrelevant word alternatives.

Richy et al. [12] describe the spelling checking component of the Grif system. Beside a general base dictionary (English, French), some special dictionaries for technical words, acronyms, proper nouns, etc. are provided. By exchanging respective subdictionaries, Grif can support the editing of multi-lingual documents. However a global architecture of the dictionaries exploiting the logical structure of documents does not exist.

Other former models are either restricted to specific application domains (e.g., spelling checking [11]) or apply conventional (i.e., relational) database technologies to organize dictionaries [5].

In contrast, we propose a partitioning of the dictionary based on word frequencies, the logical structure of documents as well as application domains involving distinct data structures and access mechanisms. The latter is hidden for character recognition.

The paper is organized as follows: Chapter 2 describes general dictionary requirements for document recognition and understanding. These requirements are considered in the next chapter which exhibits the design of an appropriate dictionary architecture. Chapter 4 is more technical, pointing to our dictionary data structures and corresponding access techniques which are primarily based on hash tables and tries. In the last chapter, we summarize the current state of implementation and ongoing research activities.

2 DICTIONARY REQUIREMENTS

Applying contextual knowledge for the verification of optical character recognition (OCR) results [15] can generally be classified into three main approaches [3] [15]: dictionary look-up methods, probabilistic methods (Markov models, Viterbi Algorithm, Bayes) and combined methods. While Markov methods use a-priori (i.e., statistical) knowledge about transition probabilities of characters (often by bigrams or trigrams), dictionary look-up techniques verify the actual input string against a legal set of words being collected in a dictionary.

We use a dictionary-based approach for two reasons: First, while Markov methods are very fast and efficient, they are extremely sensitive in case of misspelled or incomplete input words. Second, storing dictionary entries and corresponding lexical information explicitly will enable a subsequent partial understanding of documents which is a main goal of our project. A complete text understanding and full

* This work has been supported by the German Ministry for Research and Technology BMFT under contract ITW 9003 0 (Project ALV).

semantic analysis, however, is not intended (see also [1]).

Text recognition (character recognition) is primarily concerned with the spelling of a word form. Storing lexical information apart from orthography is not necessary. Consequently, requirements for the dictionary include:

1) if input is complete: fast access of entries for word verification, or rejection, respectively;

2) if input is incomplete: efficient pattern matching for the selection of appropriate word candidates;

3) tolerance and robustness towards different kinds of recognition errors;

4) compact storage allocation for loading large parts of dictionary into main memory and therefore minimizing access time;

5) dynamics: the set of words represented in the dictionary are periodically updated and enlarged in course of time;

6) flexibility: exchange of special dictionaries at run time, e.g., dictionaries reserved for technical words, domain-specific words, or multi-lingual documents [12];

7) views: definition of (virtual) subdictionaries, e.g., according to document model; these views reflect a structural restriction of context to improve recognition (e.g. possible addressees, cities and zip codes of a recipient [1]);

8) openness: an open and extendible interface allowing the integration of other knowledge sources or intermediate results of recognition, e.g., simple image features (word envelopes), length of word, cryptographical attributes, etc.

For *text analysis*, the dictionary could then provide a lexical knowledge base involving:

1) rich morphologic and syntactic information such as parts-of-speech, stems, inflections, synonyms, etc.;

2) internal (semantic) links for the association of dictionary entries, e.g. synonyms and typical phrases;

3) external links to specialized dictionaries, thesaurus or encyclopedia;

4) a coherent and homogeneous representation (lexical structure) which facilitates the acquisition and modification of lexical entries.

In fact, we take into account requirements of character recognition for the design of an appropriate dictionary architecture. This design will be described in the next chapter.

3 DICTIONARY ARCHITECTURE

After having briefly described our document analysis system and having motivated respective requirements, we will now propose an architecture of a structured dictionary which is adequate for document analysis. Peterson [11] has also suggested a similar dictionary concept which is exclusively designed to assist spelling checking and does not intend a representation of lexical knowledge.

Partitioning (i.e., structuring) the dictionary into several distinct parts or subdictionaries (word clusters) is beneficial for several reasons. First, by applying contextual knowledge, search space will be pruned enormously, in particular when input is rather incomplete. For example, in the context of letter addresses only a relative small set of words is significant (names, countries, cities, zip codes, etc.). This agrees with the second of above requirements improving access time. Second, partitions can easily be exchanged when proceeding analysis from one logical object of a document to another with respect to the document structure (cf. ODA). For instance, an address reader only needs a restricted vocabulary (names, cities, zip codes, etc.) in contrast to a general purpose recognition system.

A first strategy to partition the dictionary is based on the frequency of words. Language-specific statistics reveal that usually a small number of words is used most frequently in documents. In German, the most frequent 500 words cover 63% of common text [10]. Because text recognition is more robust in finding short words (articles, conjunctions, adjectives, etc.) and corresponding errors only refer to one or two characters, we compiled and stored them in a separate *high-frequency dictionary* as full forms. The high-frequency dictionary is organized by hash tables allowing fast access of complete words and simple error elimination which will be explained later (see Chapter 4.2). This matches requirements 1 to 3 of text recognition presented above.

Additionally, we have also compiled other 8.500 German full forms which most often appear in our text corpora. These words cover another 25% of text and are stored in a so-called *base dictionary* of German. In contrast to former dictionary, the base dictionary has been implemented employing a trie data structure. Here, we use a trie representation since our search heuristics can efficiently handle fragmentary words of character recognition (see Chapter 4.1).

In parallel, there are several *domain-specific dictionaries*. They include specific words or phrases which are significant for a particular application domain. For the analysis of business letters, these dictionaries comprise typical words, salutations, closing forms, references, and courtesies (e.g., "Dear Dr.", "sincerely", "enclosed you find", "look forward to", ...). Note that these dictionaries should be pluggable: on demand they can be added or taken away for efficiency.

Furthermore, we provide *logical dictionaries* according to logical objects of our document model (cf. [1], [6]). Logical objects divide the contents of a letter into entities associated with a sender's intellectual meaning such as subject, address or date. Each logical dictionary represents a restriction of context accompanied by a special vocabulary and phrases complying with the seventh requirement of text recognition. In Fig. 1 several logical dictionaries such as employee names, countries, cities, months, etc. are illustrated.

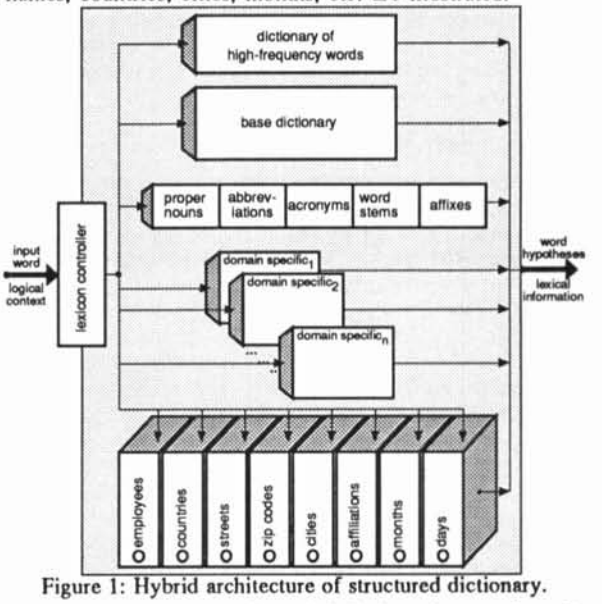

Figure 1: Hybrid architecture of structured dictionary.

Moreover, a large set of *special dictionaries* contain typical German abbreviations, acronyms, proper nouns, etc. as well as word stems and affixes. Latter two are relevant for our morphologic tool for German (Morphix [1]) which evaluates simple lexical information (e.g., parts-of-speech).

So far, we represent domain-specific dictionaries, logical dictionaries and special dictionaries as word lists. These lists are stored in separate files which neither include any lexical information nor frequency counts.

Fig. 1 schemes the architecture of our prototypic structured dictionary. The *dictionary controller* supervises and schedules access of all subdictionaries dealing with different

types of fragmentary word input and context information. For instance, it determines in which order input words are searched for. Right now, the simple scheduler implemented consults all subdictionaries sequentially. More complex and intelligent strategies have to be developed as the system evolves.

The following sections explain which dictionary organizations and access methods we support for the development of a hybrid structured dictionary for document analysis.

4 DICTIONARY ORGANIZATION AND ACCESS TECHNIQUES

A review of literature shows that a multitude of dictionary organizations and corresponding access techniques have been developed. Knuth [7] and Elliman [3] give a good survey of adequate data structures for the representation of dictionaries. Also Harris [4] compares different dictionary structures (binary search, indexing, trie search, hashing).

In our system, two competitive dictionary data structures—hash tables and tries—are used combining the advantages of both. The next two sections describe these data structures and access methods carefully weighing their advantages and disadvantages and indicating when they are best used to implement respective subdictionaries.

4.1 HYBRID TRIE

One well-known technique for dictionary organization is that of a *trie* data structure [7] [15]. Tries are attractive because of their simple and compact storage allocation. In a trie, word entries are stored character by character beginning from left to right. A trie is represented in principle by an n-ary tree where each node holds one character of corresponding words and points to maximal *n* successors (n, number of characters of the underlying alphabet). By this way, common prefixes of words are stored exactly once.

Search in a trie-based dictionary is straightforward starting at the root and comparing the keys of all successor nodes successively until the right character is found which is then chosen for the next comparison. On the other hand, if search fails, the largest prefix match corresponding to the input pattern, however, is found.

Because key information (word form), lexical data and access pointers are closely interwoven, partitioning a large trie dictionary is a serious problem. This matter is of great importance since practice reveals that trie dictionaries allocate huge parts of memory, in particular for additional link information. A second problem of search arises when input words are incomplete, in particular if the beginning of the word is misspelled or unknown. An exhaustive depth-first search, for instance, will then degrade recognition performance dramatically.

To encounter and to alleviate some of these drawbacks we have implemented several heuristics which minimize storage demand and speed up search. For a compact representation, i.e., complying with requirement 4) of character recognition, three different types of trie nodes are employed: bit-array nodes, char-pointer nodes, and string compaction. During initialization of the dictionary, the most efficient node representation is chosen automatically. For more details on the different node types see [2] and [18].

To deal with incomplete input words, especially if the beginning of a word was not recognized, prefix oriented search of tries fails. For that purpose, we have developed a so-called selective-access-matrix (SAM) allowing fast access of fragmentary input words. The SAM is a (c*n)-matrix, where c is the cardinality of the alphabet $\sum$ and n a positive integer indicating the position within a word. Each element of the matrix corresponds to a pointer array whose elements refer to trie nodes at level n containing character c [2].

Look-up in the trie dictionary is organized as follows. Complete input words are checked character by character, from left to right, as usual. Simple misrecognized words including alternatives of characters (e.g. "B[e,o]ispiel", where brackets surround alternative characters) or single rejected characters (e.g. "Beispi?l") are handled by a depth-first search.

For words including unknown substrings we employ the SAM. When multiple rejections in a word are directly at the beginning of the word, search starts with a look-up in the access matrix, which gives a number of nodes, where the search can proceed. If a match is found, the missing prefix can easily be reconstructed by backward chaining. For example, let "???inning" be the search word. The look-up in SAM[i,4] gives all nodes at level 4 containing an "i" which are the starting points for further search. When multiple rejections in a word occur in the middle or the end, the SAM is used analogously filling all gaps.

Our trie dictionary employing different node types and a selective access matrix has been completely implemented and actually contains 8500 most frequent German words (= base dictionary). Results show that additional memory needed for the SAM can be compensated by skillfully using different node types. Furthermore, search can be reduced by 10-75%, especially in case of large misrecognized prefixes. The interested reader is referred to [2] for more details.

4.2 THREE-FOLD HASHING

A major shortcoming of most dictionary access techniques is their relative slowness. For this reason, we use in parallel a hashing-based approach for document analysis. 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, e.g., [8] [13] [16].

Since input words often are incomplete, it is neither possible to consider all kinds of a potential word destruction in advance nor to use the entire word form for computation of the respective hash address when storing dictionary entries. Rather, only a few word characteristics are used. These characteristics are called *word features* and typically comprise significant groups of consecutive letters (e.g., digrams, trigrams), first/last letters, less frequent letters (e.g., j, x, y), etc. One problem, however, appears. Because features of dictionary entries generally are not unique, i.e., words usually maintain several distinct features, the same hash address of a word is assigned to feature groups [8] [16], i.e., a word can be reached via multiple features. This is designated as *redundant hash addressing*.

Our approach is similar to the one published in [13]. We also use several hash functions in parallel according to word incompleteness. These hash functions do not compute hash addresses directly, but rather yield an index of corresponding hash tables each referring to word entries in our dictionary. This concept is called *indirect hashing*.

Indirect hashing has many advantages in comparison with direct storing techniques:

1) Dictionary entries may be stored in arbitrary order, e.g., alphabetically, retrograde alphabetically, or by frequency. Hence, different access techniques might run in parallel such as binary search and hashing.

2) Other features can be used for accessing the same word (redundant hashing). These features may change, i.e., new hash functions can easily be included enhancing the dictionary's flexibility. For instance, we are now developing a hash function coping with cryptographic hash information.

3) Indirect hashing allows the compression of word entries. This becomes important if the length of words greatly differs. In contrast, bucket size for direct hashing is ascertained by the largest word length often resulting in a waste

of storage (empty buckets!).

Even the last characteristic compensates the major drawback of indirect hashing being stated in the additional storage which is needed for the hash index tables.

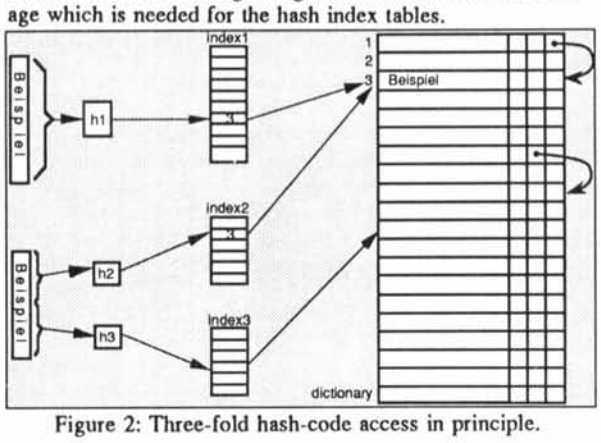

Figure 2: Three-fold hash-code access in principle.

As shown in Fig. 2, 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. All functions compute their hashing index by means of the division method [7] in combination with length of input word. The main dictionary including lexical information is stored in a separate storage area. Because collisions of keys may occur, collision is solved by linear chaining of word entries. Also note that each entry of the dictionary holds three collision pointers belonging to the different hash functions.

If the hash information of an input word contains wildcards or character alternatives, appropriate character sequences are generated (hypothesize & test). In advance, the number of possible word hypotheses are computed for each hash function to select the appropriate one which has minimal cost [15]. Additionally, we provide a parameter to specify the maximal length of a half of a word. The value of this parameter strongly depends on the number of word entries (hash table size) and distribution of word keys. While a small length typically results in many a-priori collisions, i.e., equal prefixes or suffixes are used for hashing, longer halves may lead to a variety of word hypotheses when input words are fragmentary. Thus, the parameter has to be selected carefully for each dictionary.

We have compared and statistically tested several hash functions, index table sizes, collision handling, and strategies for the selection of hash functions with respect to our high-frequency dictionary, the base dictionary containing the most frequent 8.500 German words as well as a third one which consists of about 50.000 German words. Results show that the average number of base dictionary accesses is 1,39 (resp. 1,57 accesses to the largest dictionary) for complete input words which agrees with the requirement of fast access time. For the left half of the input word 2,57 (2,96) and for the right half 2,79 (3,26) accesses are needed. (Maximal length of half of word is: 4 (base dictionary), 5 (largest dictionary)). For more information and statistics see [17].

5 STATE OF IMPLEMENTATION

A first prototype of our structured dictionary has been implemented on Sun SPARCstations in Common Lisp/CLOS. At the moment, the dictionary comprises a basis of 500 high-frequency German word forms that are stored separately in a specialized hash table allowing fast access of words and rudimentary error elimination by three-fold hashing. Right now, our base dictionary contains further 8.500 words which are collected in a trie-based dictionary extended by a selective access matrix for incomplete input providing efficient search heuristics. The storage needed is 330 KB (trie 200 KB, SAM 130 KB) [2]. All other subdictionaries such as logical dictionaries and domain-specific dictionaries are collected in simple word lists. However, the lexical data base will be increased successively.

All dictionary entries were enhanced by frequency counts of different text corpora and simple morphosyntactic information computed from a morphologic tool for German. Frequency counts and word statistics are used for a pre-classification of business letters applying traditional techniques of information retrieval.

Our current research activities concentrate on the implementation of a more intelligent dictionary controller which takes advantage of different types of subdictionaries and logical views. Also a generator (lexicon manager) is being developed for building special dictionaries according to the requirements of distinct character recognition modules.

ACKNOWLEDGEMENTS

I would like to thank my colleagues Michael Malburg, Rainer Bleisinger, and Andreas Dengel for detailed reading and helpful comments. Special thanks to our students Andreas Wagner and Adolf Pleyer who have implemented main parts of the dictionary prototype.

REFERENCES

[1] A. Dengel, R. Bleisinger, R. Hoch, F. Fein, F. Hönes. From Paper to an Office Document Standard Representation. IEEE Computer, special issue on document image analysis, vol. 25, no. 7, July 1992, pp. 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, Aug./Sept., 1992, vol. II, pp. 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, pp. 337-346.

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

[5] 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, pp. 375-393.

[6] ISO 8613 Information Processing, Text and Office Systems. *Office Document Architecture and Interchange Format (ODA/ODIF)*, parts 1-8, 1988.

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

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

[9] G. Nagy. Teaching a Computer to Read. Proc. of *11th International Conference on Pattern Recognition*, The Hague, The Netherlands, August 30 – September 3, 1992, vol. II, pp. 149-153.

[10] H. Meier. *Deutsche Sprachstatistik*. Georg Olms Verlag, Hildesheim, 2. erweiterte und verbesserte Auflage, Band 31, 1978 (in German).

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

[12] 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, pp. 183-198.

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

[14] 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.

[15] R. M. K. Sinha, B. Prasada. Visual Text Recognition Through Contextual Processing. *Pattern Recognition*, vol. 21, no. 5, 1988, pp. 463-479.

[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, pp. 363-377.

[17] A. Wagner. Organisation und Zugriffsstrukturen eines Lexikons zur Dokumentanalyse. *Diploma Thesis*, CS Department, University of Kaiserslautern, 1992 (in German).

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