Linguistics Today
Vol. 1, No.1, July, 1997, pp. 34- 50

Electronic Dictionaries and Computational Tools

K. Narayana Murthy

Department of CIS University of Hyderabad

Introduction

Linguistics has come to assign an increasingly central role to the lexicon in the recent past. Words of a language, alongwith the phonological,morphological, syntactic, semantic and other pieces of information associated with these words form a very important part of the knowledge of language. Dictionaries are storehouses of such information and thus they have a key role to play in NLP too. The development of large computerized lexical knowledge bases has emerged as probably the most urgent, expensive and time consuming task facing linguistics and NLP. The importance of computational tools for the design, development and maintenance of such lexical knowledge bases cannot be over emphasized.

In this article we sketch briefly a number of relevant computational tools which are being developed at the department of computer and information sciences, University of Hyderabad, under the common title 'tpTools'. These tools are under different stages of development - while some of the simpler tools are fully developed and are in use for quite some time, some are still in the initial stages of development. All these tools have been designed to work on personal computers and special care has been taken to permit handling of large data. Data can be as large as two giga bytes, or whatever is the capacity of the secondary memory. We begin by considering the various possible relationships between dictionaries and computers and then we look at some of the relevant tools briefly.

There are at least three different ways computers interface with dictionaries:

Dictionary on the computer.

Computer for dictionary making.

Dictionary for the computer.

We will now look at these three aspects briefly, with emphasis on the computational tools of relevance to each of these.

Dictionary on the computer

A dictionary on the computer is a dictionary which is made available in computer readable form rather than as printed pages. Instead of turning the pages of a printed dictionary, we can use the computer to search for relevant information in the dictionary. In a minimal sense, therefore, the computer is acting only as a fast page turner. However, in reality, computational tools give us not only speed and convenience but they also enable us to extract relevant pieces of information contained in the dictionary in more flexible ways than is possible in a conventional printed dictionary.

To see clearly the advantages of such dictionaries let us consider a few examples. Normally, the entries in a dictionary are arranged in alphabetical order so that words that we wish to look up can be quickly located. Now, suppose for example that somebody wants to study how nouns acquire verb senses and vice versa. He will be intersested in gathering words such as 'shop', 'pen', 'table', 'cut', 'raise', and 'kick' for his study. As a first approximation, he may want to retrieve all words which are marked as noun as well as verb in the dictionary. How do we retrieve all such words from a conventional printed dictionary ? There is no better way than to search sequentially through the entire dictionary - a humanly almost impossible task. For more examples, consider retrieving all verbs whose past tense and past participle forms are homonymous, all words of a particular etymological origin, all bisyllabic words belonging to a given grammatical category, or all words ending with any of a given set of suffixes. In all these cases, and in many other cases as well, all the required information is certainly available within the dictionary. Yet it is very difficult to extract the relevant information manually from a printed dictionary. A dictionary on the computer and appropriate computational tools make possible many new and imaginative ways of accessing relevant pieces of information than is possible from a conventional printed dictionary.

A dictionary on the computer can also be quite different from a conventional dictionary in terms of content and organization. A conventional dictionary depicts all relevant information associated with a given entry in one place, as plain running text except that different type faces and other punctuation marks are used to help the reader to quickly locate specific fields. In a computational dictionary, the relevant fields may be organized in more complex ways, for example, using database concepts, HyperText or multimedia. A database is a structured collection of information, organized so that a variety of operations can be performed on either individual units or groups of structural units. These operations help us to select entries satisfying given conditions, to pick out only those fields which are required and to display them in an appropriate manner. While text is normally considered to be entirely sequential in presentation, HyperText permits textual data to be organized in complex non-linear ways so that related items of text can be obtained easily on demand. Multimedia tools have become commonplace now-a-days. With a multimedia computer, we can also associate pictures and sound with the dictionary entries. Animated sequences and video clippings can also be incorporated if required.

A dictionary on the computer has other advantages as well. It never goes 'out of print' - updates and new editions can be brought out a lot more easily. Stored in CDs or floppies, it occupies less space, weighs much less and is much more easy to carry along than a printed dictionary. Stored on the hard disk of a computer or computer network, the user is freed from having to carry the dictionary physically. It saves on paper and is therefore environment friendly too.

The ability to access relevant pieces of information in fast and flexible ways comes mainly from the way information is organized in the compute and from the associated algorithms which manipulate this information. We shall briefly look at some of the indexing techniques in forthcoming section.

Computer for dictionary making

Computers also form a very important tool for dictionary making or lexicography. Hardly any dictionaries are prepared these days without the use of the computer at some stage or the other. Here we will briefly sketch some of the important tools that help us in dictionary making.

To begin with we need to collect and select the words that we wish to include as the entries in our dictionary. A variety of tools can be used to help us in this process. Starting with a corpus, a computer program can separate the text into words and sort the words in many different ways. For example, a wordlist sorted by frequency helps us to pick up frequently used words and to eliminate words such as technical terms, abbreviations and acronyms, proper names, loan words from other languages, slang, words that have gone into the oblivion, etc. There are other tools which help us to update a wordlist based on a new corpus or other wordlists, and to do union, intersection and different operation on worldists. There are also a variety of more sophisticated tools that employ heuristics and partial knowledge of language to select, reject and rank words for final scrutiny by the human expert.

Once the words are carefully selected, the next step is to incorporate grammatical information including the categories and other grammatical features. Computational tools can make preliminary guesses about these categories and features to simplify the task of the dictionary maker. There are basically two different sources of information that can be used for such guessing. Firstly there is often some information within the word itself. For example, a complete word form (that is, a fully inflected word) may have morphological affixes which indicate possible grammatical categories as also the root form of the word. The second type of clue lies in the sentential context in which a particular word is used. The context, in particular the adjacent words, often carry substantial amount of information about the grammatical categories and other features. For a simple example, words that follow determiners like 'the' in English are most likely to be either adjectives or nouns. A smart tool can apply many different heuristics and combine evidence from multiple sources to make fairlygood guesses.

Further, tools such as the KeyWord In Context (KWIC) tool go a long way in helping the lexicographer to identify word senses, pick out good examples from the corpus etc. The KWIC tool displays the various textual contexts in which a given word is used in the corpus. Typically, a KWIC tool gives either complete lines containing the specified word or a specified number of words to the left and right of the given word. A somewhat more sophisticated tool can list complete sentences rather than complete lines of text from the corpus.

A major advancement in the KWIC tool will be the ability to provide a filtered context, giving only the important parts of the sentence rather than the whole sentence. For examp,e, the various senses of the word 'handle' depend mainly on what is handled (handle household accounts, handle the brush, handle troops, handle a person etc.). In computer science mere availability of informations not of much interest, extracting the relevant portions and only the relevant portions, from the large storehouse of information is crucial. Filtering out the hit sentences has double effect in reducing the information load - it reduces the amount of irrelevant information displayed in each sentence involving the given keyword, and consequently, it also reduces the number of different sentences displayed since sentences which differ only in the less significant parts can be pruned. The ultimate in KWIC would be a parser based tool that can parse each sentence including the given keyword and then display the relevant parts in a structured manner. This makes the task of the human expert much simpler.

A second kind of improvement that can be made over conventional KWIC tools is related to morphology. Most available KWIC tools can search only based on exact string match and hence cannot deal with inflected and derived words effectively. A shortcut often followed in practice is to specify only a part of the keyword that will be included in all or most of the inflected and derived words. For example, to include sentences with 'handle', 'handled', 'handling' as well as 'handles', we may specify the keyword as 'handl'. But the output would then include sentences with unrelated words such as 'chandler' and 'chandlery'. A good morphological analyzer is therefore an essential part of a good KWIC tool, especially for Indian languages.

There are also other tools for creating, updating, managing, compressing, indexing, accessing, printing and analyzing large dictionaries and corpora. A non-word analyzer program takes a corpus and checks the words in it against a given dictionary and a morphological analyzer. The results are useful for validating and/or enriching the dictionary as well as the morphological analyzer. Tools also exist for entry and validation of dictionary items, internal consistency checking and for performing a variety of analyses on the dictionary itself. For example, we may list out all the words which are used in defining the meanings of the head words in the dictionary. We may check that all these words, which are normally taken to be the simplest of the words in the language, are also included as head words. We may ensure, for another example, that the definitions are not cyclic, that is, two or more words are not defined in terms of each other.

Dictionary for the computer

Electronic dictionaries are also directly usable by computer programs. Electronic dictionaries or lexicons as they are known in NLP and linguistics, form an integral compoent of almost every activity in computational linguistics and NLP - vocabulary studies, spelling error detection and correction, word processing and text critiquing systems, automatic abstracting and indexing, concordance, frequency analysis and other statistical studies, stylistics, lexicographers' workbench, hypertext databases, morphological analysis and synthesis, text tagging, parsing and generation, story understanding, taxonomical studies, knowledge acquisition, machine translation, question answering, natural language interfaces to databases, information retrieval, speech synthesis, speech recognition, computer aided instruction, psycholinguistic studies, office automation, etc.

We are not concerned here about the contents and conceptual organization of the lexicon. The answers to such questions depend heavily on what the specific application demands. The questions of content and organization are also closely tied up with theoretical framework. The Demands on the lexicon in Lexical Functional Grammar framework, for exampie, are very different from the demand made by say, Tree Adjoining Grammar.

We can only say here that the requirements of a dictionary for use by the computer are often quite different from the requirements for dictionary meant for use by human beings. For example, while monolingual dictionaries meant for human use define head words in terms of other words of the same language, such a thing would not make much sense in an NLP application. At present our bundle of computational tools includes only a few application specific tools, mainly including those related to spell checking, morphological analysis and synctatic parsing under the UCSG framework.

Indexing Techniques

Indexing helps to locate a specific information quickly. Indexes also help us to view a single physical lexicon as if it were organized in several different ways. For example, if we need to look at the dictionary entries sorted in the right-to-left alphabetical order of the head words, we do not need to store or print another version of the entire dictionary sorted in that manner. The information content in the 'normal' and 'reverse' dictionaries are exactly the same, only the organization of the information is different. Hence, we do not need to duplicate all the information and create another version of the dictionary, just an index will do. In fact dictionaries, thesauri, gloossaries and encyclopedias have substantial amounts of overlap in terms of the information they contain and hence combinations of these can be made available on the computer with a minimum amount of redundancy. Thirdly, indexing also helps us to extract relevant pieces of information in a very flexible manner. Let us now take a quick look at some of the most important indexing techniques.

The simplest thing we can do is to store the entire dictionary as a single text file, using certain delimiting characters to demarcate the different fields. Let us call this file the dictionary file. It must be noted that text file is simply a sequence of characters. There is no further structure in it which can be exploited by computer programs. Also, the fields of an entry will all be of variable lengths in a dictionary. Even the head words will be of variable lengths. Hence to search for a given word, we have to search, or at least skip over, all the fields of the dictionary, not just over the head words. Also, we cannot get any milage out of the fact that the entries can be sorted on the head words. Thus this organization is the least efficient.

A major improvement over this organization is to add another file called the index file. The index file is a structured file containing two fields. The first field contains the head word and the second field contains a number indicating the location of the corresponding entry in the dictionary file. With this organization, to search for the details of an entry, we start by searching for the given word in the first field of the index file. If it is found, the second field tells us where exactly in the dictionary file the required information is stored. Of course, to reach that location in the dictionary file, we may have to start from the beginning and skip the required number of characters in the most straight forward implementation. With some more sophistication added to exploit the way the text file is physically stored in the memory, this process can also be made a lot more efficient.

In any case, use of an index file is a substantial improvement since to search for the given head word we need not look at all other fields. Further, since the index file is structured, we can have the head words sorted on, say, alphabetical order and then make use of more efficient search techniques such as binary search. Binary search has somethings similar to the way we normally search for a given word in the conventional printed dictionary. To search for a given word we do not always start from the first word in the first page of the dictionary and proceed linearly. We usually start somewhere in between and by comparing the given word with the words on the page we have opened, we decide to go further into either the part of the dictionary before that page or the portion after that page. To use binary search, let us assume that the fields are of fixed lengths (we can make the field width big enough to accommodate all the head words). Let us also assume that the fields are sorted. Then we can directly calculate the location of the middle item in the list. We compare the given word with the word at the so computed middle location and depending upon which is alphabetically greater, we decide to throw off either the first or the second half and continue the search by the same method in the remaining portion. We continue the search until the given word is found or until portion of the dictionary to be searched further becomes empty in which case we can say for sure that the given word is not found in the dictionary at all. To clearly see the advantages, it can be verified that to search for a given item in a sorted list of 1000 items, sequential search rquires 1000 comparisons in the worst case whereas binary search requires only 10 comparisons even in the worst possible situation. To search for a word in a wordlist of one million words, it takes a maximum of only 20 comparisons.

Binary search is effective only when the index file is small enough to fit in the main memory of the computer. Often dictionaries are larger than this and we need to go for the other indexing techniques wholly or in conjunction with the basic indexing scheme just described. One possible extension of binary search is B-Tree indexing. Binary search is effectively a search in a sorted and balanced binary tree - at each step we compare the given word with the word at a particular node in the binary search tree and depending upon which is greater, we continue the search down the right subtree or the left A

B-Tree is also a balanced and sorted tree but each node contains a larger number of key words. A B-Tree of order 15, for example, is equivalent to dividing the dictionary into fifteen equal parts, each one of these again into fifteen equal parts, and so on, as long as we need to go on splitting like this. Therefore, at each step in the process of search, we compare the given word with upto 14 words and depending upon the comparisons decide which portion to use for continuing the search.

It is also possible to use compound keys with multiple sorting in B-Trees. For example, the keys may specify the length of the keyword, the first character and the last character. Then to search for a given word, say 'apple', the search will be focused on five letter words starting with 'a' and ending with 'e'. By judiciously using compound keys, we can reduce the effective area of search itself and thus achieve very high efficiency.

If we go on further and provide enough information in the key to locate the required word straight away, we have come to a scheme called hashing. A hash function takes a given word as input and after some simple calculations tell us straightaway where exactly that word is located in the dictionary. To see the point, let us assume that we have no more than one word in our dictionary of a given length, given starting letter and given ending letter. Then, by combining these three clues, the hash function can straightaway tell us where exactly the given word is stored in the dictionary if at all. In practice it is not easy to design hashing functions which are simple enough and at the same time effective enough for large dictionaries. However, combined with other indexing techniques, hashing can substantially reduce the effective search space.

In all the schemes seen so far, we are basically making comparisons between entire words. There is an indexing scheme called TRIE, wherein the comparisons will be for individual characters and thus much faster. In a TRIE we store the locations of entries whose head words include specified prefixes. Thus to search for the word '*apple*', we first look for words starting with '*a*', then continue the search and look for words starting with '*ap*', the '*app*', and so on. The complexity of searching with a TRIE is not dependent on the number of entries in the dictionaries but only on the length of the head words. If the longest head word in the dictionary is 'n' characters long, it takes no more than 'n' character comparisons to locate any word in the dictionary. Thus searching for a word in a TRIE can be extremely fast. However, a straight forward implementation of the TRIE index would require an enormous amount of memory. This is because we will have to say explicitly that there are no words in our dictionary starting with, say, '*aa*', '*aaa*', '*aaaa*', etc. A full TRIE for a dictionary with the longest head word of size 'n' requires a memory of the order of the size of the alphabet raised to the power of n+1, an extremely large number of even moderate values of n.

Thus we see that an indexing technique which is extremely efficient in time may be hopelessly bad in space requirenments. In general there is often a trade off between space and time in computer science. Thus no single technique may be good enough in a given situation and we must be able to find out the right combination that gives us the overall best performance. The decision is of course highly data dependent.

Our 'tpTools' includes tools to experiment with given data and figure out the best indexing scheme. Some specific combinations are also recommended for general use. In particular, a two level indexing scheme combining a TRIE index and a sorted index file has been shown to be very promising for most applications. Also, we have provided just two simple user controlled parameters by adjusting which one can get the best performance on any given computer with specified amounts of main memory.

While we have restricted ourselves to the problem of looking a given head word in our dictionary in the above discussions, a good indexing scheme is supposed to help you in retrieving many other useful pieces of information from the given dictionary. We have already seen in section 2 several such examples. 'tpTools' also includes tools to help you in this. Further, there are additional and special requirements when it comes to handling Indian languages. For example, many of our languages exhibit extremely rich inflectional morphology and obviously no dictionary lists all the inflected forms of all the words. Hence morphological analysis is an integral and inseparable part of dictionary look up in these languages. 'tpTools' includes some tools specifically designed for Indian languages. Indian scripts in the GIST notation are directly supported in most of the tools. Inter-conversion programs are also included to enable users to employ other script notations.

Conclusion

In our country there are several organizations seriously engaged in research and development in computational linguistics and NLP. But our lexical knowledge bases in computerized forms are minimal. We do not have, even in the printed form, all the required monolingual, bilingual or multilingual dictionaries let alone dictionaries for use by computer applications. We do now have small corpora for some of the major languages. There is need to make these corpora really available for use and there is need to enhance their size too. While many word processors and DTP packages for Indian languages are now available, the amount of textual data already available in computer readable form is rather small. Different commercial packages follow different coding and storage techniques which are often guarded carefully as trade secrets and so there is need to develop suitable interfaces for extracting pure text from these word processors and DTP systems for linguistic and NLP use. Since Optical Character Recognition Systems for Indian language scripts are also far from reality, corpus building in India has to still depend on manual typing in of texts.

Given this scene, development of computational tools for creation, management and application of textual databases, wordlists and lexicons for a variety of linguistic, lexicographic and NLP applications seems to be one of the most urgent needs. In this article we have briefly described some such tools being developed at the department of computer and information sciences, University of Hyderbad. These tools are all in different stages of development and testing. While we cannot promise readymade solutions to specific problems, we hope that the tools will be of general use to many. We also hope that the initial and definitely sketchy descriptions of these tools given here will spark off enough interest among potential users and force us to strive hard to meet their demands in full.

REFERENCES

Briscoe, E. J. & B. K. Boguraev (1988) *Computational Lexicography for Natural Language Processing.* London: Longman.

Knowles, Gerry (1994) Annotating Large Speech Corpora: Building on the Experience of Marsec, *Hermes 13*