An Efficient Lexicon Management System for Machine Translation

Gi-Chul Yang and Jonathan C. Oh

Minimal usage of storage space, maximal speed in retrieval of desired textual items, and ability to update with minimal side-effects on the lexicon are three major concerns in lexicon management systems. We propose an efficient mechanism for organization of a lexicon along with its retrieval, update mechanisms. The update operation is for all practical purposes side-effect-free; the retrieval operation is not closely dependent on the size of the lexicon nor adversely on the complexity of the internal representation of the lexical items, rather it has an interesting property of often allowing faster retrieval for more complex items; and the storage overhead is negligible.

Key Words and Phrases: Lexicon, Retrieval Mechanism, Search, Conceptual Structure, Lexicon Update.

1. Introduction

Fast retrieval from a very large lexicon is one of major consideration for natural language processing systems including machine translation and natural language interface systems. Even a severely limited system in its text domain requires a large lexicon if it is to serve a practical purpose at all. Much work is done on the organization of lexicons, their update and retrieval mechanisms (Goldman (1975), Miller & Johnson-Laird (1976), Deering et al. (1981), Cullingford & Joseph (1983)). We propose in this paper some ways to improve on these organization, update, and retrieval mechanisms of lexicons for natural language processing. We will call the proposed system Efficient Lexicon Management System (ELMS).

ELMS is a part of machine translation system which is being implemented at the University of Missouri. The machine translation system is mainly based on lexical knowledge (syntactic knowledge is also used for efficiency) which is represented as a conceptual structure ('Schank (1973, 1975)) and implemented as slot-filler structure (cf. ERKS structure (Cullingford (1986)) using Common Lisp. We will review some literature on lexical item retrieval methods in section 2. In section 3, we will discuss our system ELMS in some detail. And we will conclude the paper in section 4.

2. Review of Some Previous Works

In this section we will review some earlier works on lexical item retrieval based on conceptual representations. Goldman introduces a retrieval method of lexical items for which he uses a 'discrimination net' (Goldman (1975)). It has a binary tree structure. Terminal nodes of the tree are lexical items. A nonterminal node consists of predicates which are used in determining the path to take. Nonterminal nodes in effect serve as a guide in picking out a terminal node. Goldman's approach is the first lexical item retrieval method based on conceptual structures.

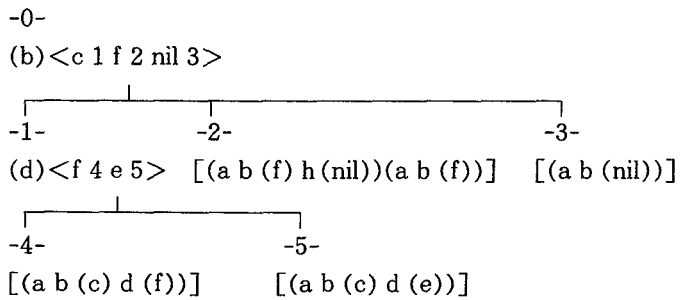
Levelt, et al. discusses some difficulties with Goldman's approach (Levelt et al. (1986)). The problem with the Goldman's approach is the employment of the serial arrangement of predicate tests at the nonterminal nodes. This makes a search in a large lexicon very expensive and creates a dependency on the complexity of the conceptual structure of a lexical item. The search for a more complex item takes a longer time than a simpler one. Also this approach is purely hierarchically organized, an obvious problem for a natural language lexicon, which often involves more than just a hierarchical structure.

Miller and Johnson-Laird proposes an alternative method (Miller & Johnson-Laird (1976)) based on a table - known as 'decision table' -rather than a tree. Pollock shows that a decision table can be translated to a flow chart, truth table, or key (Pollock (1971)). The decision table provides a more abstract representation than other otherwise equivalent representation schemes. This is due to the fact that the ordering of the testing is immaterial in a decision table unlike in the other representation styles. This has a rather important consequence, that is, that the parallel operation is

readily available for the decision table. A problem with this approach is the exponential growth of the table size as more conditions are considered. This problem may be alleviated somewhat by employing table linkage.

Cullingford and Joseph have developed a knowledge base organizing program called SOT (Self Organizing Tree) (Cullingford (1986), Cullingford & Joseph (1983)). It is a variation of Goldman's work. SOT seperates items into groups, on the basis of the path in the tree. Terminal nodes of SOT are lexical items as in a discrimination net. Each nonterminal node of a tree consists of a 'path' and an 'index'. The search always starts with the root node as is generally the case with slot-filler-based searches. Our approach departs from such traditional approaches in the search strategy, it employes a backward searching mechanism.

An example of SOT tree is:



The above tree contains the following five items:

(a b (nil))
(a b (c) d (e))
(a b (c) d (f))
(a b (f) h (nil))
(a b (f))

In a SOT tree, terminal nodes are sets of lexical items enclosed in []. Nonterminal nodes consist of a path enclosed in () and an index in <>.

Such SOT trees are built recursively. Initially one starts with the total set of all internal representations of lexical items - ERKS structures explained below -to be discriminated. The nature of a tree depends crucially on the

paths that are employed. A path is a sequence of slot-names that lead to a given value. The path for '(masc)' in the example below is '(to val partof gender)'. The only path all items have in the beginning is the null path '()'. SOT creates a candidate path by adding a slot name to preexisting paths which distinguishes the items in the set under examination. The selection of an appropriate slot name in extending old paths will contribute to the efficiency of overall retrieval strategy[1]. Once the original set of items to be discriminated is divided on the basis of the new path, the same process recursively applies to each of the resultant subgroups.

Cullingford uses ERKS (Eclectic Representations for Knowledge Structures) structures for internal representation of lexical items. An ERKS structure consists of a header followed by zero or more pairs of a role and its filler. The filler itself is an ERKS structure. An example of a well-formed ERKS structure is:

```
(propel actor
  (person persname (Olivia) gender (fem))
    object (bpart bptype (hand) partof
           (person persname (Olivia) gender (fem)))
        to (physcont val (bpart partof (person
           persname (Muhammed) gender (masc))
             bptype (nose) ref (def)))
      time (times time l (: past))
           mode (modes mode l (:t)))
```

Pointers may be used for solt-fillers. This example is a representation of the sentence 'Olivia punched Muhammed in the nose.' The syntax for ERKS is provided below, for the discussion in section 3 presupposes some knowledge of ERKS structure.

```
struct  -->  (header body)
header  -->  "atom" | nil
body    -->  slot filler | (slot filler)*
filler  -->  struct | <struct> ; <> indicates
```

[1]Cullingford works under the assumption that simpler lexical items tend to be used more frequently and therefore must come higher in the SOT tree.

the pointer

slot —> "atom" ; symbolic atom, not T & nil

3. An Efficient Lexicon Management System

Three major concerns for large lexicon management systems are ability to update with minimal side-effects on the lexicon as a whole, maximal speed in storage space. With the scheme outlined in the previous section, the update operation can be very expensive. The whole lexicon structure must be reconstructed everytime when an item is added or deleted. Otherwise the tree might go out of balance badly. Furthermore the retrieval operation is dependent on both the size of the lexicon and the complexity of the item to be retrieved.

ELMS meets all three criteria outlined in the previous paragraph. Lexical word entries are represented as slot-filler structures similar to Cullingford's ERKS structures. This slot-filler structure has compositional characteristics, so it is possible to have one conceptual structure for a whole sentence or a paragraph. From such an inclusive conceptual structure which contains more than one word, the language generator will select a word by recursive decomposition and search. We will not discuss decomposition process in this paper. Our main concern is how efficiently we can select an appropriate target word from the conceptual structure which represents the word.

3.1. Organization and Update Mechanism of Very Large Lexicon

ELMS consists of a Lexical Item Table (LIT) for the lexical data itself and a Header Index Set (HIS) for the control of LIT. LIT is maintained for analysis of the source language and HIS is maintained for generation of the target language. LIT is a two dimensional array organized along the axes of item numbers, one for each item, and header numbers. A header index set provides for each header a list of indices each of which contains the slot - filler for the header in question and also all the item numbers of those items which have that value for that header.

We will illustrate the LIT organization with a simplified example. The fol-

lowing illustrates simplified conceptual structures for lexical items.

(p x (d))
(p x (b) y (c))
(p x (d) y (e))
(p x (d) y (c))
(p x (f) y (e) z (c))
(p u (d))

As the first item is added to LIT, item number[2] 1 will be assigned to it together with header number for all the headers appearing in that item. There are two headers (p and d) in the item. The first header of the item is 'p' and the second header of the item is 'd'. Hence, ELMS assigns header number 1 (H1) to 'p' and header number 2 (H2) to 'd'. LIT at that point will have one item and two headers. The number attached to the item number is the number of headers in the lexical item called Total Header Number (THN). For instance, i1-2 means that the first lexical item (i1) has 2 headers (in this case 'p' and 'd'). LIT and HIS are given below :

LIT :

item#	H1	H2
i 1	(p x	(d))

HIS : HIS1={(p, i1-2)}
HIS2={(d, i1-2)}

When the second item, (p x (b) y (c)), is added, that item will have assigned item number 2 and one additional header (H3) since its header length is three. The Lexical Item Table (LIT) and Header Index Set (HIS) will be updated as below :

LIT :

item#	H1	H2	H3
i1	(p x	(d))	
i2	(p x	(b) y	(c))

[2] This item number can be an actual word or pointer to the words in a real system. At least 2 words should be there, one for source language and the other one for target language.

HIS : HIS1={(p, i1-2, i2-3)}
HIS2={(d, i1-2) (b, i2-3)}
HIS3={(c, i2-3)}

The final LIT and HIS are given below after all six items are added:

Item#	H1	H2	H3	H4
i1	(p x (d))			
i2	(p x (b) y (c))			
i3	(p x (d) y (e))			
i4	(p x (d) y (c))			
i5	(p x (f) y (e) z (c))			
i6	(p u (d))			

Figure 1. Lexical Item Table.

HIS1={(p, i1-2, i2-3, i3-3, i4-3, i5-4, i6-2)}
HIS2={(d, i1-2, i3-3, i4-3, i6-2) (b, i2-3) (f, i5-4)}
HIS3={(c, i2-3, i4-3) (e, i3-3, i5-4)}
HIS4={(c, i5-4)}

The deletion is hardly expected but if it should happen then just delete the item from the LIT and its item number from the HIS. The cost for the latter would not be worse than $O(n)$. Thus, the update operation does not cause any restructuring of the database, just addition or deletion of the item in LIT and of its item number in the header index lists.

Some simple software tool can be used here to order the role-filler pairs in a consistant way and also in such a way that the least frequently used role in the dictionary items with its filler come last. This order is fixed at first time when the structure is defined. This is a useful supplementary technique for reduced searching time, but it is not essential for ELMS.

Two or more items may have the ERKS structures in such a way that they have slot-fillers with the same header but their slots are different in the same position. Compare i1 and i6 in the above example. These two have the same headers but their second headers are for two different roles (i.e., x and u). In such a case, we may not be able to uniquely identify a lexical item. But we conjecture that whenever this happens the set will not be very

large, and we can use some pattern matching algorithm (for each roles) to find the particular item under search.

3.2. A Fast Lexical Item Retrieval Mechanism

The most important consideration in a very large lexicon management system is the speed of the retrieval operation. Furthermore, the complexity of the internal structure of an item should not adversely affect the efficiency of the retrieval operation. More complicated item (i.e., the lexical item which has more slot-fillers) should be accessed faster than the simpler lexical items from the lexicon, since more complicated lexical items are more specialized, i.e. more salient features which are not found in majority of other lexical items (principle of greater specificity). In other words a large number of items may be eliminated from the set of items to be searched in by using those extra salient features in comparison. Similar opinions have been expressed elsewhere. For example, Levelt et al states that more complex verbs take systematically less time to access than less complex ones do (Levelt et al. (1978)).

Assuming that the patterns for comparison and the dictionary entries are both maximally specified, or to put it differently that the retrieval operation is based on complete identity, retrieval can be accomplished by 1) getting the header index set HISj for the last header where j is the number of headers in the ERKS structure of the lexical item under search; 2) finding the header index list[3] (HIL) which contains the j-th header in the pattern. If it is a singleton list[4](a list with exactly one item), then that item must be the entry we are looking for. If not, compare THN of each lexical item in the HIL with THN of the entry we are looking for and get the lexical item numbers which has the same THN with the lexical item under search. If it is a singleton list, then that item must be the entry we are looking for. If not, get the correct HIL from HIS(j-1) for the (j-1)th header and get the intersection list between these two lists. If the resultant list is a singleton

[3] Header Index List (HIL) is a subset of HIS. For example, (c, i2-3) is a HIL of HIS3.

[4] For example, (c, i2-3) is a singleton list, since there is only one item (i2-3) in the list. We are not consider the header in the list as an element of the list.

list, the item therein is the entry we are looking for. If the intersection list is still not a singleton, we repeat the above process until eventually we come to a singleton list, in which case we have succeeded in locating the lexical item we want, or hopefully to a small but not a singleton list, in this latter case we use some pattern matching algorithm to find the particular item under search.

For example, suppose that a generator tries to access the fifth item in Figure 1. First we get HIS4, which contains only one HIL, (c, i5-4), which is a singleton list. The item we are looking for is i5. For another example, let us consider a pattern identical to i3. We get HIS3 since the number of the headers in the structure is three. Since the last header (i.e., third header for this example) for this item is 'e', we get the HIL '(e, i3-3, i5-4)'. This is not a singleton set, hence, the system will check THN of each lexical item in the HIL. THN of i3 is 3 and THN of i5 is 4 in this case and THN of the lexical item we are looking for is 3. Therefore the system will retrieve i3. In order to retrieve the lexical item same with i4, get HIS3 as previous example with the same reason and get HIL, (c, i2-3, i4-3). This time the HIL is not a singleton list and THN of each lexical item in the HIL '(d, i1-2, i3-3, i4-3, i6-2)' because the second header in the structure is 'd'. The intersection of these lists ((c, i2-3, i4-3) $\wedge$ (d, i1-2, i3-3, i4-3, i6-2)) is the singleton list '(i4-3)', so the fourth item (i4-3) is selected before we consider the first header of the ERKS structure. Notice that the search in most cases need not exhaust the ERKS structures as these examples demonstrate. The complex items in internal representation are not necessarily penalized, rather often rewarded with a faster match. As the size of the dictionary increases, the individual HIL will grow longer, demanding a longer search time within a given HIL. However, since search might end sometime before we get to HIS1, the dependency of the efficiency of retrieval operation on the size of lexicon is not a close one.

The ELMS has been implemented on VAX11/785 using Common Lisp. It consists of Lexical Item Table Organizer (LITO) and Fast Lexical Item Retriever (FLIR) with some additional routines. LITO builds the LIT and HIS with the user input. FLIR retrieves the correct lexical item quickly by using backward searching. Followings are the summary of the procedures discussed so far.

LITO

Extract all the headers from the input item & maintain Total Header Number (THN).

Build a Header Index Set (HIS) according to the position of each header.

For all headers in the input lexical item:
If HIS is empty then add (new_header, item_number) pair in the HIS
else compare HIS with new header
If any header in HIS is the same with the new header
then append the (lexical) item number to correct HIL
else add (new-header, item-number) pair on the HIS.
End for

FLIR

Extract all the headers from the input and maintain number of headers in THN for the input.
For THNth HIS :
Consider the last header (i.e., n-th header) of input & find the HIL (HIL is a subset of HIS) which contains the last header of the input.
Let X=the number of items in the HIL
Loop: If X=1, We are done. Exit with the item number in the HIL

If X<1, the item is not in the lexicon. Exit with an appropriate message

If X>1, get the lexical item which has same THN as input's THN. If there is only one lexical item then exit with the item number. If not, consider (n-1)th HIS & find the correct HIL which contains the same header as the (n-1)th header in the input.
Let X= size of the intersection of the HIL here and the HIL of the (n-1)th HIS above.
Current HIL=Intersection of two HILs. n=n-1.
Go to Loop.

4. Conclusion

We have outlined an efficient lexicon management system for very large

lexicon with a fast retrieval operation, a side-effect-free update operation. The overhead for organizing the lexicon for the system is favorably compared with other organizations. Unlike the other lexicon ELMS employs backward searching mechanism and it allows to search the lexical item faster, since we often do not have to compare all the headers in the ERKS structure (it does not travel from the root node to a leaf node all the time). Whenever there is only one ERKS structure left for consideration the searching process will be terminated. Each time when we compare headers, numerous ERKS structures might be deleted from the next header comparision.

Selection of an appropriate data structure in implementing header index set (these need not be lists) is an important consideration for the resultant efficiency of the system. Also selection of the order of role-filler pairs used in lexical representations can be used to minimize the searching time. Empirical studies need to be conducted with a view to determining the relative frequency of the use of these roles in the lexicon.

References

Cullingford, R. E. (1986) *Natural Language Processing : A Knowledge Engineering Approach*, Rownam & Littlefield.

Cullingford, R. E. & Joseph, L. J. (1983) 'A Heuristically 'Optimal' Knowledge Base Organization Technique,' *Automatica*, Vol. 19, No. 6.

Deering, M, Faletti, J. & Willensky, R. (1981) 'PEARL-A Package for Efficient Access to Representation in LISP,' *IJCAI* 81, Vancouver, British Columbia.

Goldman, N. (1975) 'Conceptual Generation,' In: R. Schank, *Conceptual Information Processing*, Amsterdam: North-Holland.

Levelt, W. J. M, R. Schreuder & E. Hoenkamp (1978) 'Structure and Use of Verbs of Motion,' In: R. N. Campbell & P. T. Smith (eds.) *Research Advances in the Psychology of Language*, New York: Plenum.

Levelt, W. J. M. & Schrifers, H. (1987) 'Stages of Lexical Access,' In: Gerard Kempen (ed.) *Natural Languages Generation*, NATO ASI Series, Martinus Nijhoff Publisher.

Miller, G. A. & Johnson-Laird, P. N. (1976) *Language and Perception*,

Cambridge, Massachusetts: Belknap/Harvard Press.

Pollock, S. L. (1971) *Decision Table: Theory and Practice*, New York: Wiley-Interscience, A Division of John Wiley & Sons, Inc.

Schank, R. C. (1973) 'Identification of Conceptualizations Underlying Natural Language,' In: Schank, R. C & Colby, K. M. (eds.) *Computer Models of Thought and Language*, San Francisco: W. H Freeman.

Computer Science
University of Missouri-Kansas City
Kansas City, MO 64110
U. S. A.