ISSN: 2319-5967
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 1, Issue 1, September 2012

Catalogue-Based Parsing

Madhusudan, Aman Kumar Sharma
m9736177566@gmail.com, sharmaas1@gmail.com

Abstract- ***This paper presents a technique for tagging in natural language processing that can enhance the speed and accuracy of the part-of-speech tagging in the statistical parsing by using the concept derived from the schema that is used in the database design i.e. relation schema along with the multilevel paging and pipelining concept for fast searching and indexing. The running time of a parser depends upon the searching of respective words in the chart and their respective tags to match with the parse trees stored in the Parse Tree database.***

Index Terms **- Parsing, Natural Language Processing, Tagging, Pipelining, Part-of-Speech, Top-down, Bottom-up.**

I. INTRODUCTION

Parsing is the process of breaking the input sentence into phrases and further breaking these phrases into words and finding the possible tags for the words. A tag would be the possible part-of-speech for a given word according to the rules of grammar. A number of tags are combined to form the sentence and derive its interpretation. Generally each word in the sentence has one meaning, but at times same word(s) may have more than one possible meaning. For example, words like can, rust, race have different meaning and tags under different circumstances. The word race is a noun when it is followed by determiners and it is a verb when it is followed by 'to'. It is the responsibility of the tagger to select the most appropriate tag so as to have the correct meaning of the sentence formulated [1]. Different studies have been conducted related to pipelining. One of the approach is to linearly sequence the activities of the pipelining, where in morphological analysis is followed by semantic and syntactic analysis along with word-disambiguation [2]. Another study proposes parallel pipelining approach. Number of pipelines would adhere to the needs of all the activities namely morphological analysis, word-disambiguation and semantic analysis. All analysis are performed in parallel. Madhusudan et.al. in their study have proposed a framework which significantly reduces the time needed for parsing. The study proposed to use multiple pipelines being addressed to a word-bank, containing all the possible words in a given language along with the possible tags, generating word tags which are being fed into the main memory for the purpose of parsing. The concept of having more than one pipeline generates a result parallel and reduces the time used to a great extent. Although the efforts behind the parsing remains the same [3]. Grammar rules are applied in a top-down manner on the sentence in top-down parsing [4]. Input sentence is broken down into the constituent phrases and then these phrases are further decomposed into the words. Then the tags are assigned to these words. On the contrary in bottom-up parsing technique, the individual words are assigned tags and then according to the rules of grammar these tags are combined into the phrases and so on to generate the complete parse tree for the input sentence [5]. There are many ways of combining the best features of top-down and bottom-up parsing into a single algorithm. The straightforward approach is to adopt one technique as the primary control strategy used to generate trees and then use the constraints from the other technique to filter out the inappropriate parses [6]. All the possible parse trees can be generated in parallel but it typically entails the use of an unrealistic amount of memory to store the space of trees as they are being constructed [7]. A more reasonable approach is to use depth first strategy expanding the search space incrementally by systematic exploring one state at a time. The parsing algorithm in this technique uses the top-down, depth-first and left-to-right strategy. Even augmented with the bottom-up filtering, the top-down parser has three problems that make it insufficient solution to general-purpose parsing problem [8]. These three problems are left-recursion, ambiguity and inefficient reparsing of sub trees. Depth-first search has a well-known flaw when exploring an infinite search space: it may dive down an infinitely deeper path and never return to visit the unexpanded states [9]. A left-recursive non-terminal can lead a top-down and depth-first left-to-right parser to recursively expand the same non-terminal over and again in exactly the same way, leading to the infinite expansion of trees [10]. Top-down parser also suffers from the problem of ambiguity where it may fail while choosing the correct part-of-speech for a word [11]. The ambiguity problem is related to another inefficiency of the top-down parser where it often builds valid trees for a portion of the input then discards them during backtracking, only to find that it has to rebuild them again [12]. There is lot of memory consumption and efforts in the basic top-down parser [13]. Natural language parsing is the process of mapping an input string or a sentence to its syntactic representation. Every sentence in a text has a single correct analysis and that speakers of the language will typically agree on what this preferred analysis is, but we do not necessarily assume that there is a formal grammar defining the relation between sentences and their preferred interpretations [14].

ISSN: 2319-5967
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 1, Issue 1, September 2012

The main objective of this paper is discussed in section 2. Section 3 explains the proposed framework with in-depth description of multilevel paging with Chart, parallel processing & tree database. The paper is summarized with future remarks in section 4.

II. OBJECTIVE

The objective of this study is to study the existing technique of processing an input sentence while parsing in NLP and devise a new technique for increasing the speed and throughput of overall process by reducing the time and memory utilization with the help of multilevel paging, multiple pipelines and Parse-tree database.

III. PROPOSED FRAMEWORK

A. Multilevel paging

The process of parsing is described as a search where the appropriate tags for the words in the given input sentence are searched through the database containing the words along with theirs tags. These words along with their respective tags are combined to generate the parse tree and its possible interpretation. All the possible words in a language are stored in a database called Word-bank. If the words are searched in the database while tagging in a linear manner, then in the worst case scenario the whole word-bank need to be traversed. To reduce the search time, multilevel paging is devised in this model. All the words are categorized in pages having their first two letters same within a catalogue presented in a box. The words with same first letter are stored in the same page. For e.g. all the words with A are stored in one page similarly with B and so on. But this will reduce the search time by a factor of 26 (if it is assumed that there is uniform distribution throughout for all the words) only which is not so efficient. So again the page with A are further portioned into the 26 pages with indexing. This will further reduce the time factor by 26. The alphabet and the part-of-speech table are indexed or like in database the only schema provided to the user is the external schema. Similarly, for fast indexing, the table is supposed to contain only few words or just the starting letters of the particular class falling under the category. For example, in case of 'Natural', it contains only the 'Na' or 'Nat' in the front table and the actual word in the attached database, so as to reduce the wastage of space and search time. Thus a multilevel page database is used here. Since the fast memory is to be characterized by the efficient usage of time and space along with high speed-up, multilevel page can prove fruitful with hundred percent accuracy of the inputted text.

$$Total\ Time\ required = Search\ Time + Propagation\ delay$$

$$Search\ Time = \frac{(no.of\ words\ in\ a\ block * time\ to\ search\ single\ word)}{(no.of\ blocks\ in\ the\ chart * no.of\ pages\ in\ the\ that\ block)}$$

$$Propagation\ delay = communication\ time\ in\ pipeline$$

$$Memory\ utilization = \frac{Total\ Memory}{(no.of\ blocks * no.of\ pages)}$$

As the input is entered into the machine as a text, it's tagging is done word by word, collecting all the possible tags for a given word and generating the various possible parse trees and then selecting the tree with highest probability.

In this technique as the first letter or word is entered, it is carried to the table for highlighting the particular column and turning its state to *active* and all other columns to the *inactive* state. For example, if the word entered is "Auckland" then the A column is ignited to active state. Since each entry of the matrix is further partitioned alphabetically so 'AU' will only be set active and selecting only one column reducing the search time nearly by a factor of 26.

The partitioning helps in reducing the time of searching by the factor depending upon the weight of that column and words in that column. This is so because the words in English are not equally distributed under each alphabet. Also since the linear search is being used so the search time will be proportional to the number of words in that particular column.

The idea behind using pipeline is that as the next word is entered (to increase the speed, we are not going to wait for the tagging of last word and then entering the next word) the tagging of the previous word is over, making it ready for parse tree assignment. Pipeline here is also useful in case if the tagging of the previous word is not over yet then also the tagging of next words should proceed. Multiple pipelines are used so that when the user enters the last word of the sentence, the tagging of all the previous words should be over making them ready for parse tree.

ISSN: 2319-5967
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 1, Issue 1, September 2012

Once the tagging is over, the possible parse trees can be generated or mapped from the "Parse tree" database and choosing the one with the highest probability value. Statistical parsing is used here to discover the best parse tree using the conditional probability. In general, statistical parsing model defines the conditional probability, ***P (T|S)*** for each candidate parse tree ***T*** for a sentence ***S***. the parser itself is an algorithm which searches for the tree ***T*** that maximizes ***P (T|S)***. Once the parse tree is discovered, the sequence can be assigned for mapping it to meaning. Same technique can be used for generating the text-output.

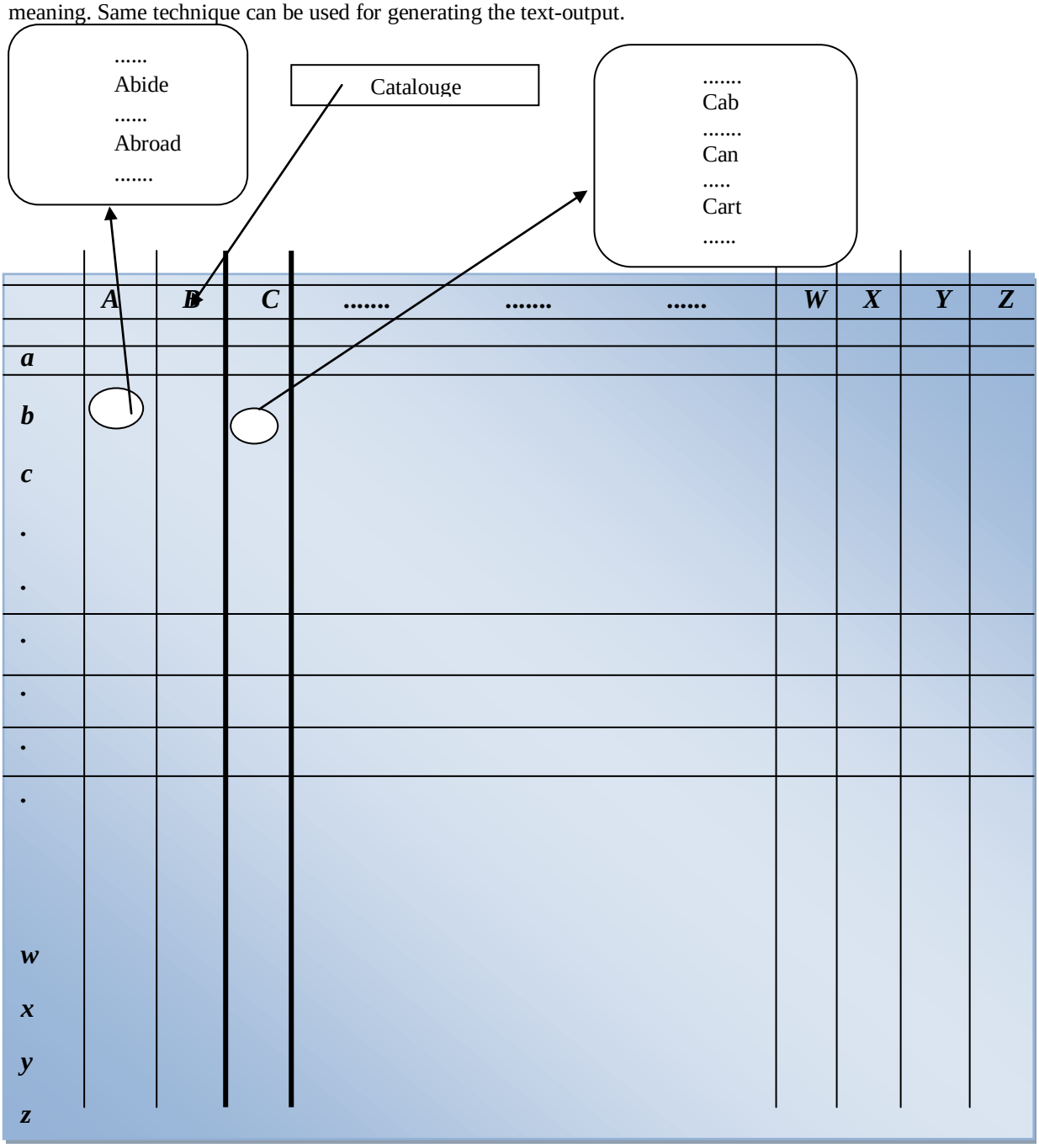

Fig.1 .Multilevel Paging

Fig. 1 is explained as under. The framework comprises of a catalogue formulated on the basis of rows and columns. The rows as well as columns contain English alphabets in an alphabetical sequence. The first letter of the word is presented in the first row and the second alphabet is in the first column. The junction of the row and column would contain the list of all those words which have first letter of the word same as first row and similarly the second letter of the word will be as per the first column. To explain the concept the magnified

ISSN: 2319-5967
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 1, Issue 1, September 2012

junction point of Ca and Ab are presented in the figure 1. Similar notion shall be followed for all the remaining junctions of rows and columns. The framework is explained using two cases as depicted below.

Case 1:

In this case an example of searching the word 'can' is used. When a word with *'c'* is encountered in the given input sentence, the domain of the table is searched. The column with C contains all the possible words starting with 'c' in the particular language. Thus, there is no need to search throughout the word bank. Only the column under C is set to active state and all other columns are set to inactive mode thereby reducing the search time and memory consumption. Then the second letter of the word is read, for e.g. if the word is can, the next letter is 'a'. As the second letter is read, the intersection of the column with C is done with the row containing the letter 'a'. Now only one block is set to active state containing their intersection and all other block set to inactive state. Hence, the search is reduced to only one block of the table i.e. the block containing all the words starting with the first letter as 'c' and followed by 'a'. As depicted in the figure the selected block contains all words whose first two letters are c and a. For e.g. car, care, cat, cart etc. Since the word 'can' was being searched but the list contains other words also such as 'car', 'care', 'cart' and many more. The linear search is performed in the block to find 'can'. The search though linear but still will be very fast as the desired word is being searched within a small set of words. Similarly the words like; cat', 'cab' are searched.

Case 2:

In another case the word 'abide' is searched. If the word in the input sentence is starting with 'a', then the column under A is set to active state and all other 25 columns are set to inactive. Again the next letter of the word is read and e.g. In this case if the word is 'abide', then next letter is 'b'. Hence the intersection of the column A is done with the row containing B. The intersection provides only those words which start with 'a' and are followed by 'b'. Hence the search time is reduced by the factor of 26 while setting the 25 columns to inactive state and again by 26 while setting the 25 rows to inactive state. Now the resultant block gives all the words with 'ab' as starting. The selected block contains all English language words whose first two letters are Ab. List of few such words such as 'abide', 'abate', 'abroad', 'abandon', 'abduct', and many more.

This approach reduces the effort of search by a great magnitude. Out of the full planetary set of English words the search is limited to only a few words whose first two characters are known. Moreover the complexity of the search is also not increased during the search. The size of the word ranges between 2 to 10 English alphabets with a rare exception. By restricting the first two alphabets the linear search shall not be for more then a few hundred words. This reduces the complexity, effort and time.

IV. CONCLUSION & FUTURE SCOPE

Parsing is defined as a search process where the possible tags for the words in the input sentence are searched through the database containing these words along with their tags. Most of the time is consumed in searching these words. In the devised technique the search time for the words is reduced by the factor of 25 when the first alphabet of the word is read. Similarly when the second alphabet is fed the search time is reduced by the factor of 25 again. Thus there is considerable amount of time saving using the above method. The complexity is characterized by the factors namely time and space. The proposed framework reduces both the time required and the amount of memory required for parsing. The overall complexity is reduced by a large factor increasing the overall speed and throughput of the parsing process. There are possibilities for improvement by using further interleaved pipelining in the technique where the blocks can further be interleaved by categorizing them into small modules and thus reducing the time for search and the memory required. That means the blocks containing Ca can be further broken down into Caa Cab Cac and so on to induce more pipelining and reducing the search space.

REFERENCES

[1] E. Charniak, "Statistical Techniques for Natural Language Parsing". Available: http://www.cs.brown.edu/~ec/, 1997.

[2] K. Hollings head. and B. Roark, "Pipeline Iteration".

[3] Madhusudan et. al., "Optimising Parsing with Multiple Pipelining", IJERD, 2012.

[4] A. Glennie, "On the Syntax Machine and the Construction of a Universal Compiler", 1960.

[5] V.H. Yngve, "Syntax and the Problem of Multiple Meaning", 1955.

[6] M. Kay, "Experiments with the Powerful Parsers", in proc. 2eme Conference International sur le Treatment Automatique des Langue, Grenoble, 1967.

[7] J. Earley, an Efficient Context-free Parsing Algorithm, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, 1968.

ISSN: 2319-5967
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 1, Issue 1, September 2012

[8] S.L. Graham, M.A. Harrison, and W.L. Ruzzo, An Improved Context-free Recognizer, 1980.

[9] N.Chomsky, the Logical Structure of Linguistic Theory, Plenum, 1975.

[10] T. Kasami, an Inefficient Recognition and Syntax Analysis Algorithm for Context-free Languages, 1965.

[11] D.H. Younger, Recognition and Parsing of Context-free Languages in Time N^3, Information and Control, 1967.

[12] M.Munoz, V. Punyakanok, D. Roth, and D. Zimak, A learning Approach to Shallow Parsing, 1999.

[13] J. Eisner, Efficient generation in Primitive Optimality Theory, In ACL/EACL-97, Madrid, Spain, 1997.

[14] McDonald and Nivre, "Learning dependency translation models as collections of finite state head transducers. Computational Linguistics", 26(1), March 2007.