International Journal of Engineering Research and Development
***e*-ISSN: 2278-067X, *p*-ISSN: 2278-800X, www.ijerd.com**
Volume 4, Issue 3 (October 2012), PP. 08-12

Optimizing Parsing with Multiple Pipelining

Madhusudan[1], Aman Kumar Sharma[2]
[1]*Department of Computer Science, Himachal Pradesh University, Summerhill, Shimla.*
[2]*Associate Professor, Department of Computer Science, Himachal Pradesh University, Shimla.*

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 pipelining concept for fast searching and indexing. The running time of a parser depends upon the searching of respective words in the word-bank and their respective tags to match with the parse trees stored in the Parse Tree database.

Keywords:- Parsing, Natural Language Processing, Tagging, Pipelining, Part-of-Speech, Top-down, Bottom-up.

I. INTRODUCTION

Parsing natural language sentence can be viewed as making a sequence of disambiguation decision: determining the part-of-speech of the words, choosing between possible constituent structures and selecting labels for the constituents [1]. A tagger assigns to each word in a sentence the part-of-speech that it assumes in the sentence. Although most English words have only one possible part of speech, many words have multiple possible parts of speech, and it is the responsibility of a tagger to choose the correct one for the sentence at hand. A simple algorithm can achieve 90-percent accuracy [2]. Each sentence-tree pair in a language has an associated top-down derivation consisting of a sequence of rule applications of a grammar [3]. A pipeline system consists of a sequence of processing stages such that output from one stage provides the input to the next [4]. Each stage in such a pipeline identifies a subset of the possible solution and later stages are constrained to find the solutions within the subset. Pipeline systems are ubiquitous in natural language processing, used not only in parsing [5], but also machine transition [6] and speech recognition [7]. Despite the widespread use of pipelines, they have been understudied, with very little work on general techniques for designing and improving pipeline systems [8]. In the existing techniques namely Top-down Parsing and Bottom-up parsing, lot of time and effort is wasted. In Top-down scheme, parser searches for a parse tree by trying to build from the root node *S* down to the leaves. The algorithm starts by assuming the input can be derived by the designated start symbol *S.* The next step is to find the tops of all the tress which can start with *S.* Then using the rules of grammar the *S* is expanded and all the possible trees are generated in parallel. Trees are grown downward until they eventually reach the Part-of-Speech categories at the bottom of the tree [9]. Bottom-up parsing starts with the words of the input and tries to build trees from the words up, again by applying rules from the grammar one at a time. The parse is successful if the parse succeeds in building a tree rooted in the start symbol *S* that covers all of the input [10]. The Top-down approach spends considerable effort on *S* trees that are not consistent with the input. This weakness in Top-down parsers arises from the fact that they can generate trees before examining the input [11] [12]. In the bottom-up strategy, trees have no hope of leading to an *S* or fitting in with any of its neighbors, are generated with the wild abandon [13]. Neither of these approaches adequately exploits the resources. Also, these approaches do not return expected results while dealing with ambiguous words. Words with multiple parts of speech lead to repeated parsing of tree and searching the words time and again in the database. Most of the time while processing an input sentence is wasted in these tasks and hence results in low performance. The error or mistake in the input sentence also leads to its wrong interpretation and thus wastage of time and space. The complexity of an algorithm is defined in the terms of time and memory [14]. If repeated efforts are made to generate the parse tree then it leads to lot of time consumption and memory utilization. In Software engineering the error in the first stage propagates to later stages and gets multiplied by a certain factor at each stage leading to failure of the designed software. Similarly the error in the first stage of Natural Language Processing (NLP) i.e. submitting the input propagates to the final stages and results in misinterpretation [15]. Also, in the primitive methods, various trees are generated for the ambiguous sentence resulting in wastage of memory and time. The input sentence is entered into the parser/tagger as the first step. The machine depending upon the knowledge base, assign tags to the words in the input sentence [16]. Then the text is mapped into the parse tree and its syntactic and semantic analysis is performed. This is called natural language understanding [17]. Then a desired output is generated but need to be converted into the human readable format like English, so that it can be interpreted by the users. This mapping from meaning to text is called natural language generation [18].

The main objective of this paper is discussed in section 2. Section 3 explains the proposed framework with in-depth description of 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 multiple pipelines and Parse-tree database.

III. PROPOSED FRAMEWORK

A. Multiple Pipelining

Pipelining is the technique of processing in which the task is divided into smaller subtasks and each subtask is carried out independently. When all the pipelines are over with their processing, their output is collected for the solution of the whole task. In parsing of given input sentence, the input is fed into the multiple pipes equal to the number of words in that sentence. All the pipelines carry the task of searching and tagging irrespective of others. In the end, the output from all the pipelines is collected for generating the parse tree.

The idea behind using pipeline is that as the next word is entered (since 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 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. One may use 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.

Another possible better way is to use the concept of dynamic programming. In order to reduce the ideal time of pipes, when the first word is being entered, the next pipe, that is, its adjacent pipe is set to the *Active* state preparing it for tagging and rests all other pipes to *Inactive* state. The pipes will receive the input and will help for fast indexing and searching of the words in the chart. Also, as the tagging is over for the last or for second last word, that pipe is set to *Inactive*. The pipe is attached with cache to copy the searched word along with its tag(s) to be used for the further stages of processing. In order to incorporate the good and efficient pipelining, an average number of pipes can be set to the *Active* state as the input is started being entered. This is for average case and a wild guess can be made. On the basis of experience and sources required for the single word tagging, we can estimate the number of pipes to be set *Active* and thus helping judicious use of the available resources because setting all the pipes to active initially will result into low usability and performance beating the purpose of the technique. For example if on an average a given language contains the maximum sentence length, then setting all the ten pipes to *Active* will demonstrate the worst case scenario.

$$\text{Resource Utilization} = \text{Memory Utilization} + \text{Pipeline usage}.$$

$$\text{Pipeline Usage} = \frac{\text{no .of pipelines used}}{\text{Total no .of pipelines}}.$$

Where resource utilization is the defined as the percentage of time for which the resources are used to the total available time. Memory utilization is defined in terms of the available memory to the used at that instant. Pipeline usage is also defined in the terms of percentage of used pipelines to the total number of pipelines.

B. Tree Database

Tree database is collection of all the possible parse trees that can be generated from the rules of grammar in a particular language. All the possible parse trees are stored in a database in mutually exclusive sets such that the trees with noun phrase are stored in one module, the verbs in other and so the auxiliary are. The idea behind portioning is that regarding the parse tree of the input sentence an efficient technique is employed. As soon as the first word is tagged as a noun or a verb in the input sentence is entered, then only those parse trees in the 'Parse Tree' database are set *Active* which applies to the given sentence and bind according the rules. The requirement of the algorithm is that the parse tree in the database, are stored in groups. All the parse trees with the noun phrase as the beginning are categorized and stored at consecutive memory locations for fast search and attached with the signal wire to be set *Active* or *Inactive* accordingly. Similarly verb phrase are stored and used. As the sentence with the beginning of noun phrase is entered, then only that particular database is set active and searched reducing the time and ambiguity. This reduces the search time and repeated searching which is often encountered in the tagging process like top down and bottom up parsing methods. As the pair of input sentence and parse tree is obtained, that database is set to *Inactive* state for low consumption of resources and thus enhancing the performance index and overall speedup.

Once the tagging is over, the possible parse trees are 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, $\boldsymbol{P(T|S)}$ for each candidate parse tree $\boldsymbol{T}$ for a sentence $\boldsymbol{S}$. the parser itself is an algorithm which searches for the tree $\boldsymbol{T}$ that maximizes $\boldsymbol{P(T|S)}$. Once the parse tree is discovered, the sequence can be assigned for mapping it to meaning. Same technique is used for generating the text-output.

IV. MODEL

The generation of desired output as expected by the user is possible only if the machine is provided with accurate input without any syntactic and semantic error and thus machine is able to interpret the input as well as the expected output. So, both the halves of natural language processing i.e. natural language understanding and natural language generation are the mirror images of each other with the correct parse tree and its semantic analysis as the criteria. If the first half is correct and accurate, then the probability of desired output increases, excluding the case of analysis.

In the existing scenario, if on evaluation of the input sentence by the parser an error is encountered the entire sentence is evaluated again to find the possible solution. In this way lot of time and effort are consumed in again evaluating the entire input.

If it is assumed that the parser to be designed, is for only specific application or purpose, then it's possible to create such a table for exact and hundred percent accuracy. Similarly the table for different applications can be created as

discontinuous sets (mutually exclusive) and finally can be merged to create a general purpose machine or super machine. As the input is entered into the machine as a text, it's tagging is done in parallel by feeding them to multiple pipelines, collecting all the possible tags for a given word and generating the various possible parse trees and then selecting the tree with highest probability. Indexing and fast memory are used to enhance the speed of tagging process. Interleaved piping can be used within the words but there may result in expensive and excessive usage of resources and less usability of pipelines. The concept of cache is used regarding the search of repeated occurrence of a particular word. For example in English language words like *the, a* are encountered often frequently and can be kept in cache for not traversing the table time and again into the chart for tagging. The same cache is used to store the words recently searched and tagged which will be useful in case of ambiguous sentences. If 'the' word is encountered again in the input sentence then all the previous pipelines are checked to find if it has been already tagged or not.

As shown in the figure 1 the algorithm devised above receives the input which is processed through the multiple pipes which are attached to the independent cache memory. The pipelines are attached to the Word-bank, which contains the English words. After search, the word along with its possible tags is saved in the memory. The memory is added with the Parse-tree database containing the various available Parse-trees.

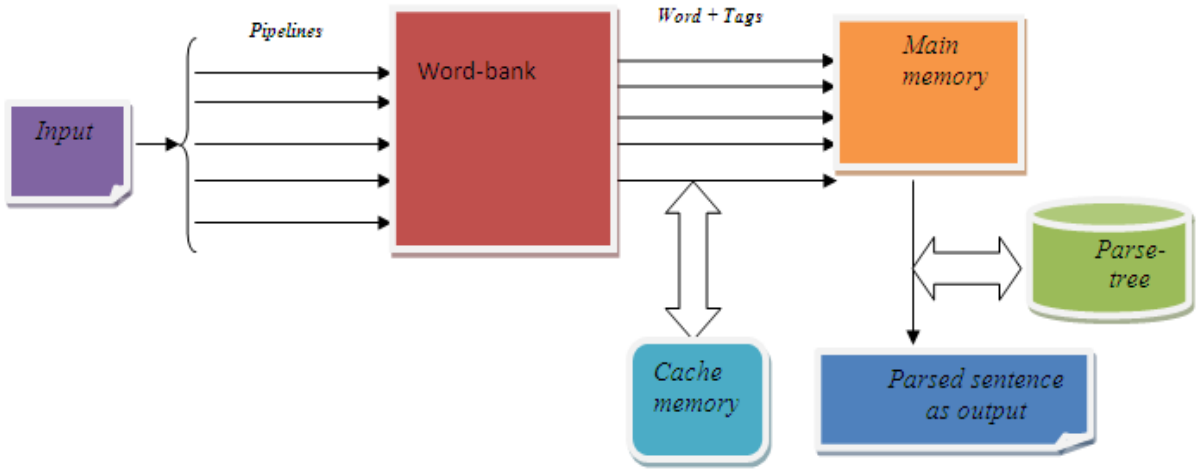

Fig. 1 Example of parsing with multiple pipelines

Total Time Required = Search Time + Propagation delay.

Search Time ≈ (number of Words) / (number of pipelines).

Propagation delay = communication time in pipeline.
Where search time is the time required for finding the given word in the database and propagation delay is the time taken by the pipeline to carry the word and returning it to and from the database in the parsing process.

According to above equations, in case of parsing without pipelines, the whole database is searched thus resulting in unnecessary and time consuming search. But with the use of pipelines, the searching time is reduced by the factor of number of pipelines used. There is considerable reduction in the search time. Let the given input sentence contains 10 words and each word require 1 time cycle, therefore 10 cycles will be required to process the sentence. In case of pipelining, only one cycle is required to process the entire sentence since 10 pipelines are activated using parallel processing model. Resource utilization is greater in case of pipelines because not all the pipelines are set to active state initially. With the use of dynamic programming when the words are entered step by step the next adjacent pipe is set to active state and not all the memory blocks are set to active but only those with the first alphabet same as the input word. Therefore overall utilization is high as compared to the primitive techniques and speed is also increased by the factor of number of pipes. If some ambiguous word is there in the input sentence then to deal with these words, all the possible tags for it are stored in the cache attached to the respective pipe at the first stage and thus repeated search and construction of Parse-tree is not required as in the primitive methods. The order of the part of speech in the processed sentence is used to map it on the Tree database and eliminating all other trees whose first part of speech doesn't match with the part of speech of the input sentence hence further saving the time.

Parsing with the help of multiple pipelines is shown Fig. 2. The input sentence is fed into the pipelines in parallel and the search and their tagging is done saving the tags in the respective caches.

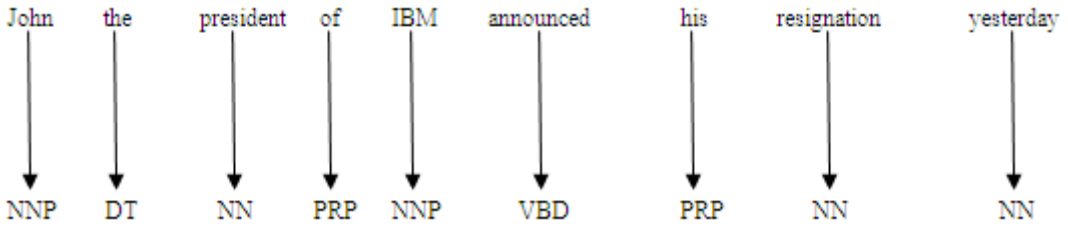

Fig. 2 An example of parsing with multiple pipelines

In case of sentence with words having multiple part-of-speeches, all the tags are saved in the cache attached to the pipelines and if there is ambiguity with one tag then the parse tree is tested with the next tag and so on. No repeated search is performed and only pipelines with ambiguous part-of-speech are disturbed. As shown in the figure.3, all the possible tags are saved for the ambiguous words. The correct one are highlighted.

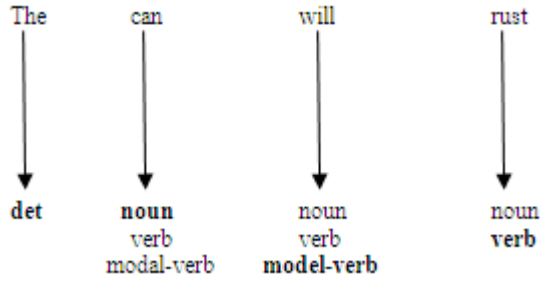

Fig. 3 An example of tagging with multiple part-of-speeches

The technique mentioned above increases the speed of the parsing process and thus overall throughput of Part-Of-Speech tagging which is proportional to the number of words in the given sentence. More the number of words in the given input sentence more will be the pipelines required and thus increasing the cost but will process the greater sentences with more speed and accuracy. The technique is basically devised to increase the speed keeping the accuracy of processing intact so that no mistakes are being committed at the first place, so the given technique will maintain the accuracy along with optimum utilization of the resources. The dynamic programming of the pipelines used results in maximum utilization of the resources and also the cache memory attached with the pipes saves the respective words that it was assigned along with the tags till the correct parse tree for the current input sentence is obtained and thus reducing the repeated search and tagging of the words.

V. CONCLUSION & FUTURE SCOPE

The basic techniques of the parsing namely Top-down and Bottom-up, neither of them adequately exploits the resources and provides expected results while dealing with ambiguous words. Top-down approach waste high amount of memory and time generating the possible trees in parallel and Bottom-up technique is not sure of the expected results. Words with multiple parts of speech lead to repeated parsing of tree and searching the words time and again in the database. Most of the time while processing an input sentence is wasted in these tasks and hence results in low performance. The error or mistake in the input sentence also leads to its wrong interpretation and thus wastage of time and space. The proposed technique eliminates the repeated search and parsing and thus saves the time and efforts. If there is some error in tagging of a word in the sentence then only respective pipeline need to be re-examined and not the whole sentence. The model also results in higher resource utilization.

There are possibilities for improvement by induction of dependency parsing and interleaved pipelining which can further increase the value of factors like precision and recall.

REFERENCES

[1]. D. Magerman, Statistical Decision-Tree Models for Parsing, Proceedings of the 33rd annual meeting of association for computational linguistics, pages 276-283, 1995.

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

[3]. M. Collins, "Three Generative, Lexicalized Models for Statistical Parsing". Available: http://www.cs.columbia.edu/~mcollins/.

[4]. K. Hollingshead. and B. Roark, "Pipeline Iteration".

[9]. A. Ratnaparkhi, "*Learning to Parse Natural Language with Maximum Entropy Models*", 1999. Available: https://sites.google.com/site/adwaitratnaparkhi/publications.

[10]. F.J. Och and H. Ney, "*A Systematic Comparison of Various Statistical Alignment Models*", Computational Linguistics, 2003.

[11]. V. Goel, "*Segmental Minimum Bayes-risk ASR Voting Strategies*", In proceedings of ICSLP, 2000.

[12]. J. R. Finkel, "*Solving the Problem of Cascading Errors: Approximate Bayesian Interference for Linguistic Annotation Pipelines*", In proceedings of EMNLP, 2006.
[13]. A. Glennie, "*On the Syntax Machine and the Construction of a Universal Comipler*", 1960.
[14]. V. H. Yngve, "*Syntax and the Problem of Multiple Meaning"*, 1955.
[15]. S. Kuno, "*Functional Sentence Perspective: A Case study from Japanese and English*", 1972.
[16]. E. T. Irons, *"A Syntax Directed Compiler For ALGOL 60*", 1961.
[17]. A. V. Aho, R. Sethi and J.D. Ullman, "*Compilers: Principles, Techniques, and Tools"*, 1966.
[18]. S. Lipschutz, Data Structures and Algorithm, 3rd edition, Tata-Mcgraw hill.
[19]. D. M. Magerman, "Natural *language parsing as Statistical Pattern Recognition*", Doctoral dissertation, Stanford University, Stanford, California, 1994
[20]. B. J. Grosz, "*The Representation and Use of Focus in Dialogue Understanding*", University of California, 1977b.
[21]. E. Reiter and R. Dale, *"Building Natural Language Generation Systems*", Cambridge University Press, Cambridge, 2000.
[22]. D. Jurafsky and J. H. Martin, Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition second edition.