

Analysis of Parsing Algorithms in Natural Language Processing

Pankaj V. Nimbalkar *, Dr. P. K. Butey **

*(Department of Computer Science, R.T.M. Nagpur University, Nagpur-10

** (Department of Computer Science, R.T.M. Nagpur University, Nagpur-10

ABSTRACT-

In this paper, we make comparative study of different parsing algorithms which is used in machine translating system. We studied two parsing algorithms namely Early parsing Algorithm and Cocke-Younger-Kasami (CKY) Algorithm. Probabilistic grammar can also be used to disambiguate parse trees. In this paper we used certain parsing techniques which remove certain ambiguities during parsing. We used bottom-up and top-down parsing techniques to reduce the ambiguity. We examine both the CKY algorithm and the Early algorithm in the context of modern multi-processor hardware and modify both algorithms to take advantage of the parallelism available with such machines[1]

Keywords - Ambiguity, Top-down parsing, Bottom-up parsing, backtracking

I. INTRODUCTION

A CFG defines the syntax of language but does not specify the assignment of structures. The rules of grammar which rewrite to either generate a particular sequence of words or reconstruct its derivation is termed as parsing. The syntactic parser is responsible for recognizing a sentence and assigning a syntactic structure to it. Any sentence that can have multiple parsers called syntactic Ambiguity. For parsing any sentence we required backtracking in some cases, for that we required search process either from left side or from right side. Some constraints can be used for the search process[2]. We used Top-down and Bottom-up parsing techniques used in two different Algorithms to reduce the ambiguity. Early parser uses three operations namely predictor, scanner, completer whereas CKY algorithms uses probabilistic parsing methods.

A. A Basic Top-down parser

The approach is depth first, left to right search. The depth first approach expands the search space incrementally by one state at a time. At each step, the left-most unexpanded leaf nodes of the tree are expanded first using the relevant rule of grammar. The leftmost node is selected for as it determines the order in which input words need to be considered. A basic top-down parsing algorithm maintains agenda of search states. Each search state consists of partial trees and a pointer to the next input word in the sentence[3]. In any successful parse, the current input word must match the first derivation of the node that is being expanded.

1) Coordination Ambiguity: Coordination ambiguity occurs when it is not clear which phrases

are being combined with conjunction like and. In disambiguation, correct parse may be identified from number of possible parses. A parse may utilize statistical and semantic knowledge to disambiguate the parse tree. Or it may return all possible parses and leave the disambiguation for subsequent processing. The basic top-down parser returns the first successful parse without exploring other possibilities.

2) Local ambiguity: Local ambiguities occur when certain parts of a sentence are ambiguous. The parser makes a few incorrect expansions. Another problem associated with basic top-down strategy is that of repeated parsing. The parser often builds valid trees for portion of the input that are discarded during backtracking.

Dynamic programming algorithms can solve these problems. In parsing, a dynamic programming algorithm builds a table containing sub-trees for each and every constituent appearing in the input.

3) Early parser: The early parser implements an efficient parallel top-down search using dynamic programming. It builds a table of subtrees for each of the constituents in the input. The states in each entry provide the following information:

- A subtree corresponding to grammar rule
 - Information about the progress made in completing the subtree.
 - Position of the sub-tree with respect to input.
- Earley typically outperforms CKY parsing because it generates fewer intermediate parse trees that do not contribute to the final parse tree, it is also much harder to parallelize because the operations performed by the algorithm are less independent than the operations performed by CKY. Thus, while Earley may outperform CKY running on a single processor,

the limitations the algorithm places on parallelism allow parallel CKY to surpass it given enough processors. The Earley algorithm operates by constructing a chart with $n + 1$ entries, for an n -word sentence. It fills each entry with states representing a partial parse tree that has been generated thus far. Each partial subtree is represented only once. The algorithm iterates over each entry in the chart, filling it completely before moving to the next entry. Within each entry, it iterates over each state in the entry, predicting when it encounters a non-terminal state, scanning when it encounters a terminal state, and completing when it encounters the end of a non-terminal state. While existing states are never modified, these operations can add states not only to the next entry, but also to the entry currently being processed. Furthermore, these operations can generate duplicates of states already present in an entry. In order to prevent not only wasted computation, but infinite recursion in some grammars, entries must keep only unique states[4].

```

Input :Sentence and the grammer
Output:chart
Chart[0]←S'→S,[0,0]
n←length(sentence)
    //Number of words in the sentence
for i=0 to n do
    for each state in chart[i] do
        if (incomplete(state) and next category is not a
part of
            speech )then
            predictor(state)
        else if(incomplete(state) and next category is a
part of
            speech )
            scanner(state)
        else
            completer(state)
        end-if
    end-if
end for
return
Procedure
predictor(A→X1.....B.....Xm[I,j])
    for each rule(B→α) in G do
        Insert the state B→.α,[j,j] to chart [j]
    End
End
Procedure
scanner(A→X1.....B.....Xm[I,j])
    If B is one of the part of speech associated with
word[j]
        then
            Insert the state B→word[j],[j,j+1] to chart [j+1]
        End
End
Procedure completer (A→X1.....[j,k])
    For each B→x1....A.....[I,j] in chart[j] do

```

```

        Insert the state B→x1....A.....[i,k] to chart[k]
    End

```

Fig 1 Early parsing algorithm

II. OPERATIONS IN EARLY PARSING

At each step one of the three operations are applicable depending on the state. Application of these operators results in addition of new states to either the current or next set of states.

A. Predictor

The predictor generates new states representing potential expansion of the non-terminal in the leftmost derivation. A predictor is applied to every state that has a nonterminal to the right of the dot. The application of this operator results in this creation of as many new states as there are grammar rule for the non terminal.

B. scanner

When a state has a part of speech category to the right of the dot then scanner is used. The scanner examines the input, if it is part-of –speech appearing to the right of the dot matches one of the part –of-speech associated with the current input. The parser finds a part-of-speech category next to the dot, it checks if the category of the current word matches with the expectation in the current state.

C. Completer

When dot reaches the right end of the rule then completer is used. This state signifies successful completion of the parse of some grammatical category.

III. THE CYK PARSER

The CKY algorithm parses sentences by constructing an $n * n$ chart, where each cell (hi, ji) corresponds to the sentence fragment spanning from word i to word nj . It fills each cell with the parse trees for its corresponding fragment, a process that culminates in a set of parse trees that span the entire sentence. The insight behind the algorithm is that the parse trees in each cell are binary combinations of the parse trees for each way of bisecting the cell's sentence fragment on word boundaries. At the core of CKY, the algorithm for computing the contents of a single cell consists of three nested loops. The outermost loop bisects the sentence fragment spanned by the new cell on each word boundary, examining each pair (l, r) of sentence fragments that can be concatenated to form the new cell's sentence fragment. Each of these sentence fragment pairs corresponds to a pair of neighboring cells: the parses of l are found in a cell to the left of the new cell and the parses of r are found in a cell below it[5]. We

refer to this outer loop as a dot product because it combines these neighboring cells in a pair-wise fashion, collecting parse trees that can be constructed as binary combinations of parse trees from some pair of cells. The inner two loops of CKY perform a join operation between each pair of neighboring cells, pairing up each possible parse of l with each possible parse of r and constructing new parse trees out of any pair of parse trees that matches the right hand side of any grammar rule. For example, if l can be parsed as an NP or a JJ and r can be parsed as a VP or a PP, the algorithm will try each combination.

```

Let  $w = w_1 w_2 w_3 w_4 \dots w_j \dots w_n$ 
and  $w_{ij} = w_i \dots w_{i+j-1}$ 
//Initialization step
  For  $i = 1$  to  $n$  do
    For all rules  $A \rightarrow w_i$  do
      Chart[ $i, i$ ] =  $\{A\}$ 
//Recursive step
  For  $j = 2$  to  $n$  do
    For  $i = 1$  to  $n - j + 1$  do
      Begin
      Chart[ $i, i + j$ ] =  $\Phi$ 
      For  $k = 1$  to  $j - 1$  do
        Chart[ $i, j$ ] := chart[ $i, j$ ] U  $\{A/A \rightarrow BC$  is a
        production
                                and  $B \in \text{chart}[i, k]$ 
                                and  $C \in \text{chart}[i + k, j - k]\}$ 
      End
  If  $S \in \text{chart}[1, n]$  then accept
  Else
    reject
    
```

Fig 2 The CYK algorithm

A. Measuring Parallelism

When parallelizing any algorithm, it is important to consider the algorithm's work and span. Work is the total amount of computation performed by the algorithm, across all processors (or, equivalently, the total time taken when run on one processor). Span is a theoretical measure of the fastest time an algorithm could execute given a infinite number of processors. An algorithm's span is limited by its critical path, the longest sequence of computations that depend on each other's results[8]. The ratio of an algorithm's work to its span limits its speedup, or the performance it can achieve relative to its single processor performance. An ideal parallel algorithm achieves linear speed-up; that is, given P processors, it will execute P times faster than it would on one processor[6]

B. Parallelizing Parsing

We mention methods of parallelizing CKY and Earley algorithms earlier. Both CKY and Earley are instances of dynamic programming algorithms

and both divide the parsing problem into smaller subproblems that can be solved separately[7]. This structure makes them amenable to parallelization because the separate subproblems can often execute in parallel. CKY parallelizes very well because the dependencies between operations are very sparse and well-defined. Earley proved more difficult to parallelize; in literature some parallelized Earley by making the entries in the Early chart independent, and predicting all possible rules per entry instead of working only with a subset of states from the entry that came before [9]. We chose to instead maintain the top-down nature of Earley for contrast with CKY.

IV. CKY PERFORMANCE

The final CKY algorithm achieves near-perfect scaling up to 16 cores, the number of cores available in our test system. However, parallelism allows the parser to scale better not only with the number of available processors, but also with the sentence length and grammar size. Increasing either of these factors increases the ambiguity of the final parse, which, in turn, increases the opportunities for parallelism.

A. Earley Performance

Earley is a top-down parser. Single processor performance compare against 8 processor and 16 processor performance. The performance for short sentences is different for a different number of processors, but increase the sentence length and thus the ambiguity, the gap between single processor performance and multi-processor performance does not widen as dramatically as with the CKY parser, This is because as the number of words increases, the number of entries in the chart increases, and the Earley parser must process these sequentially.

V. RESULTS

The CKY and Earley algorithms are implemented on Cilk++ 4.2.4 (a commercialization of MIT Cilk-5 [11]), utilizing Cilk's fine-grained work scheduling algorithm. All experiments ran on an AMD 16-core system running Linux with 64 Gbytes of memory and used the Penn Treebank Wall Street Journal grammar, as well as the WSJ grammar samplings. Compare both the absolute times required by the parser implementations, as well as their speed-up relative to single processor performance. Considering both is important, as the added complexity of parallelism can negatively impact the absolute performance of an algorithm, even if it improves its relative performance.

VI. CONCLUSION

Natural language parsing is a natural application for parallelization, though achieving linear speedup requires carefully understanding the algorithm and modifying it to create fine-grained units of work that can be solved independently, avoid points of synchronization, and size data to fit within a processor's cache. Furthermore, while some algorithms may achieve higher performance in traditional, single processor settings, in the parallel realm, the scalability of an algorithm has the greatest impact on its performance. As processing power grows with the number of cores on a chip, utilizing gains in computational power to better understand and process natural language will require rethinking existing single-threaded algorithms to take advantage of highly scalable multicore hardware architectures.

Comparative difference between Earley parsing algorithm and CYK parsing algorithm

Early parsing algorithm	CYK parsing algorithm
The early parser implements an efficient parallel top-down search using dynamic programming	The CYK parser implements an efficient bottom-up approach using dynamic programming
It builds a table of sub-trees for each of the constituents in the input	It builds a parse tree incrementally by building a parse tree of length 1.
The algorithm eliminates the repetitive parse of a constituent which arises from backtracking and successfully reduces the exponential time problem to polynomial time	The process is iterated until the entire sentence has been parsed
The early parser can handle recursive rules such as $A \rightarrow AC$ without getting into an infinite loop	This algorithm considering all rules which could produce words in the sentence being parsed.
This algorithms mostly used in Context Free Grammer	This algorithm mostly used in Statistical parsing
The early parser algorithm fills the chart entry in polynomial time, extracting all parse trees still requires an exponential amount of time.	The search space of possible tree structures is usually very large and the search is quite time consuming

REFERENCES

[1]. Bharti, Akshar and Rajeev sangal, 1990, 'A karaka based approach to parsing of Indian languages' proceedings of 13th conference on computational linguistics, Association for computational linguistic-3

[2]. Chomsky, N., 1957, syntactic structures, mouton, the Hague.

[3]. Marcus, Mitchell p., Beatrice santorini and mary Ann, 1993, 'Building a large annotated corpus of English: the penn Treebank computational linguistic, 19, pp.313-30

[4]. Charniak, Eugene, 1993, statistical language learning, MIT press, Cambridge.

[5]. J. Earley. An efficient context-free parsing algorithm. Commun. ACM, 13(2):94-102, 1970.

[6]. J. C. Earley. An efficient context-free parsing algorithm. PhD thesis, Pittsburgh, PA, USA, 1968.

[7]. M. Frigo, C. E. Leiserson, and K. H. Randall. The implementation of the Cilk-5 multithreaded language. In Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation, pages 212-223, Montreal, Quebec, Canada, June 1998. Proceedings published ACM SIGPLAN Notices, Vol. 33, No. 5, May, 1998.

[8]. T. Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Laboratory, 1965.

[9]. A. Koulouris, N. Koziris, T. Andronikos, G. Papakonstantinou, and P. Tsanakas. A parallel parsing VLSI architecture for arbitrary context free grammars. In Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS), pages 783-790, 1998.

[10]. M. Sipser. Introduction to the Theory of Computation, chapter 2.1, pages 98-101. PWS, 1997.

[11]. The Cilk Project. <http://supertech.csail.mit.edu/cilk/>. D. H. Younger. Recognition and parsing of context free languages in time n^3 . In Information and Control, volume 10, pages 189-208, 1967.