An Adaptive Earley Algorithm for LTAG Based Parsing

Sharafudheen K.A. and Rahul C.

MES College of Engineering, Kuttippuram, Kerala, India

Abstract. In traditional parsing methods Earley parsing is one of the best par ser implemented for both NLP and programming language requirements. Tree Adjoining Grammar is powerful than traditional CFG and suitable to represent complex structure of natural languages. An improved version LTAG has appropriate generative capacity and a strong linguistic foundation. Here we introduce a new algorithm that simply adopts Earley method in LTAG which results combined advantages of TAG and Earley Parsing.

Keywords: Adaptive NLP Parser, Tree Adjunction Grammar, LTAG, Earley Algorithm for TAG.

1 Introduction

Tree Adjoining Grammars are somewhat similar to context-free grammars, but the elementary unit of rewriting is the tree rather than the symbol. Whereas context-free grammars have rules for rewriting symbols as strings of other symbols, tree-adjoining grammars have rules for rewriting the nodes of trees as other trees.

TAG has more generative capacity than CFG. For example it can be shown that $L_3=\{a^n b^n c^n\}$ is a context free language, but $L_4=\{a^n b^n c^n d^n\}$ is not context free. TAG can generate L_4, so it is more powerful than CFG. So TAG is a mildly context sensitive language. On the other hand $L_5=\{a^n b^n c^n d^n e^n\}$ is not a Tree Adjoining language, but it is context sensitive. So it follows that L(CFG) < L(TAG) < L(CSG).

Definition 1(Tree Adjoining Grammar): A TAG is a 5-tuple $G = (V_N, V_T, S, I, A)$ where V_N is a finite set of non-terminal symbols, V_T is a finite set of terminals, S is a distinguished nonterminal, I is a finite set of trees called initial trees and A is a finite set of trees called auxiliary trees. The trees in I U A are called elementary trees.

2 LTAG and Properties of LTAG

In LTAG, each word is associated with a set of elementary trees. Each elementary tree represents a possible tree structure for the word. An elementary tree may have more than one lexical item. There are two kinds of elementary trees, initial trees and auxiliary trees. Elementary trees can be combined through two operations, substitution and adjunction. Operations are substitution and adjunction. Former is used to attach an initial tree, and later is used to attach an auxiliary tree.

V.V. Das and J. Stephen (Eds.): CNC 2012, LNICST 108, pp. 383–386, 2012.

The key Properties of LTAG are

- Extended Domain Locality
- Factoring Recursion from the domain of Dependencies (FRD), thus making all dependencies local (Joshi and Schabes, 1997 [5]).

3 Extending Dotted Symbols and Dotted Tree Concept

Use of dots in LTAG is basically same as that proposed by Earley (1970) for his algorithm for CFG. We mimic the same idea here. Dot on left side of a non-terminal indicates that the tree has not been explored yet. Right side dot indicates that all its children are already explored.

Adjunction builds a new tree from an auxiliary tree β (with root/foot node X) and a tree α (with internal node X). The sub-tree at internal node X in α is excised and replaced by β; the excised sub-tree is then attached to the foot node of β.

The most common usage for substitutions on initial trees, but substitution may also be done at frontier nodes of auxiliary and derived trees. Substitution takes place on non-terminal nodes of the frontier of a tree (usually an initial tree). The node marked for substitution is replaced by the tree to be substituted.

4 Proposed Algorithm

The algorithm uses two basic data structures: state and states set.

Definition 2: A state s is defined as a 5-tuple, [a, cur_it, pos, parent, lchild where a: is the name of the dotted tree, cur_it: is the address or element of the dot in the tree a, pos: is the position of the dot; parent: is the parent element of the cur_it; For start node it is ϕ, lchild: is the left child of the cur_it; For leaf node it is ϕ.

A state set S is defined as a set of states. The states sets will be indexed by an integer: Si with i ϵ N. The presence of any state in states set i will mean that the input string $a_1 \ldots a_i$ has been recognized. Algorithm for *state set creation* is

```
Let G be an LTAG,
Let a1…anbe the input string,
/* Push initial state (α0,s',L,ϕ,S) to stateset 0
ENQUEUE(α0,s',L,ϕ,S) {Dummy}stateset 0
For(i=1 to LENGTH(sentence) do
    For each state in stateset i do
        If (incomplete (sentence) ) and any Operation is
possible
            PREDICTOR(state)
        If (incomplete (sentence)) and any Operation is
not possible
            SCANNER(state)
        Else
            COMPLETOR(state)
        End
```

```
    End
End

Algorithm Predictor
  For each state cur_it as root in stateset(i) and for
all GRAMMAR_RULE

     Case 1: Dot is on the left side of a NT

        If NT is not a leaf

       ENQUEUE(tree,cur_it,L,P,lc) {Predictor}

       /*Do Adjunction Operation

       /*Add all cur_it rooted element to stateset(i)

            Move dot to immediate left child

        Else

       ENQUEUE(tree,cur_it,L,P,lc)              {Predictor}
       /*Substitution Operation

        End

     Case 2: Dot is on the left side of a Terminal

       ENQUEUE(tree,cur_it,R,P,ϕ) {Predictor}

       /*Move dot to right of the terminal

  End

Algorithm Scanner
   /*Increment stateset index
   For word (j) in input sentence
       Find elementary tree for the word
       ENQUEUE(tree,root,L,ϕ,lc) {Scanner}
   End
Algorithm Completer
  For each state that all left tree and all child
explored
  Case 1: Dot is on the right side of a NT
           If a sibling exist
             ENQUEUE(tree,sibl,L,P,nlc) {Completer}
             /*Move dot to left of immediate sibling
           Else
             ENQUEUE(tree,P,R,GP,cur_it) {Completer}
             /*Move dot to right of the parent
           End
      Case 2: If Dot is on right of a Terminal
             ENQUEUE(tree,root,R,GP,cur_it) {Completer)
  End
```

4.1 Complexity

The basic idea and method of the proposed algorithm is from the Earley Parsing Technique and the average complexity is of the proposed work is not changed than Earley Parsing even after change. On analysing it shows $O(|G|n^3)$ in average behavior in time and $O(|G|n)$ in space where |G| is the length of input grammar.

5 Conclusion

We design a new Earley parser based algorithm for LTAG. It works in lesser complexity than any of the existing TAG parser. It is easy to implement and complex data structure of existing Earley algorithm for TAG has modified to a simple one. It combines the advantages of both TAG and Earley parsing. Worst case behavior is also adaptable.

References

1. Aho, A.V., Sethi, R., Ullman: Compilers: principles, techniques, and tools. Addison-Wesley (2002)
2. Shen, L., Joshi, A.K.: Statitical LTAG Parsing. Ph.D. thesis, University of Pennsylvania (2006)
3. Joshi, A.K., Schabes, Y.: Tree-adjoining grammars. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 69–124. Springer (1997)
4. McDonald, R., Crammer, K., Pereira, F.: Online large-margin training of dependency parsers. In: Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics, ACL (2005)
5. Shen, L., Joshi, A.K.: An SVM based voting algorithm with application to parse reranking. In: Proceedings of the 7th Conference on Computational Natural Language Learning (2003)
6. Frost, R., Hafiz, R., Callaghan: Modular and Efficient Top-Down Parsingfor Ambiguous Left-Recursive Grammars. In: ACL-SIGPARSE, 10th International Workshop on Parsing Technologies (IWPT), pp. 109–120 (2007)
7. Chiang, D.: Statistical Parsing with an Automatically-Extracted Tree Adjoining Grammar. In: Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics, ACL (2000)