ACSIJ Advances in Computer Science: an International Journal, Vol. 5, Issue 2, No.20 , March 2016
ISSN : 2322-5157
www.ACSIJ.org

ACSIJ
WWW.ACSIJ.ORG

On the automatic construction of LTAG Grammars from a Vietnamese Dictionary

Ha Phan Thi[1], Nam Ha Hai[2]

[1] Faculty of IT, Posts and Telecomunications Institute of Technology, VietNam
hathiphan@yahoo.com

[2] Faculty of IT, Posts and Telecomunications Institute of Technology,VietNam
namhh@ptit.edu.vn

Abstract

This paper presents a method for automatically extracting lexicalized tree-adjoining grammars (LTAG) from a Vietnamese dictionary. A system for automatically extracting LTAG grammars for Vietnamese from an electronic dictionary has been implemented for evaluation purpose. The experiment results of the proposed method are evaluated against the results of extracting LTAG grammars from VietTreeBank.

Keywords: LTAG; Lexicalized Tree Adjoining Grammar; Treebank, dictionary.

1.Introduction

Syntax analysis is a critical step in natural language processing pipeline. High quality syntax analysis will improve the performance of a natural language processing systems such as machine interpretation, text summarization, automatic Q&A systems.

Every syntax analyser needs a set of syntactic rules called linguistic grammar that is represented by a specific grammar formalism. Manual construction of grammars is a time consuming and tedious process. Therefore, much research has been carried out to solve the automatic or semi-automatic construction of grammars. Almost published research on the construction of grammars for natural language processing systems focused on popular languages such as English, French, Chinese... In general, there are two main approaches to the automatic construction of grammars. The first approach employs high-order grammar descriptions to generate the grammars called meta-grammar [1]. The second approach focuses on automatic extraction of grammars from a syntax-annotated corpus called Treebank. A method for automatic extraction of LTAG from VietTreeBank was proposed in [2]. The method proposed in this paper takes the second approach, which will automatically extract the LTAG grammars from a Vietnamese electronic dictionary. Dictionaries are usually developed by linguistic experts. This results in high accuracy in morphology, syntactic and sematic information. Each lexical item of a dictionary consists of three types of information: morphology information, syntactic information and sematic information. Using additional information from dictionaries is expected to enhance accuracy of the LTAG grammar extraction.

The rest of the paper is structured as follows: Section 1 provides background about the LTAG grammars; Section 2 introduces the Vietnamese dictionaries used in the proposed method; Section 4 describes the extraction algorithm based on dictionaries; Section 5 presents the experiment results and efficiency comparisons between VietTreeBank-based and dictionary-based LTAG grammar extraction methods; and the final section discusses conclusions and future directions

2. Tree-Adjoining Grammar – TAG

Tree-adjoining grammar is a grammar formalism proposed by Aravind Joshi in [3,4]. A TAG grammar is a 4-tuple G = <N,T,I,A>.

- N is a finite set of nonterminal symbols
- T is a finite set of terminal symbols
- I is a finite set of initial trees
- A is a finite set of auxiliary trees

Tree-adjoining grammars are classified as mildly context-sensitive grammars using trees as elementary units for rewriting rules. Much research on TAG has focused on formalism and applications to analysis of different natural languages such as English, French [5,6,7,8,9]. The elementary unit of a tree-adjoining

grammar is elementary trees. If every elementary tree has at least one leaf node wth a terminal symbol then the tree adjoining grammar becomes lexicalized tree adjoining grammar (LTAG).

Elementary Trees

A tree-adjoining grammar is composed of a set of elementary trees. There are two types of elementary tree: initial trees and auxiliary trees, which are the basic building blocks of the formalism. An initial tree has all inner nodes labeled with nonterminal symbols; and the leaf nodes are either labeled with terminal or nonterminal symbols, which are marked with the substitution marker ("↓", for instance). An auxiliary tree is defined as an initial tree, except that exactly one of its leaf nodes must be marked as a special node called foot node. The foot node must be labeled with a non-terminal symbol (`*', for instance), which is the same as the label of the root node.

Two rewriting operations

Trees in TAG can be combined using two operations: substitution and adjunction. Substitution operation substitutes a leaf node labeled with a symbol X of a tree α with a tree β whose root node is labeled with the same symbol X. The substitution operation is illustrated in Figure 1.

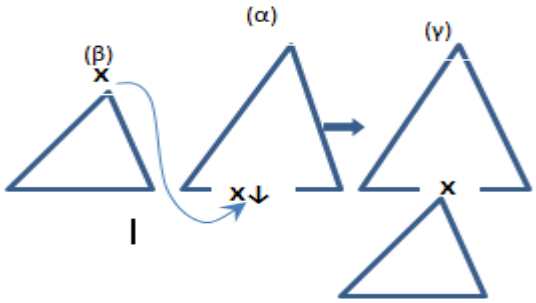

Figure 1. Illustration of substitution operation.

Adjunction operation inserts an auxiliary β with the root node labeled with a symbol X into another tree α at an inner node u labeled with a symbol X and the original sub tree of α rooted at node u is extracted from α and inserted below the foot node of β. The adjunction operation is not performed at nodes marked as substitution nodes of α. Figure 2 illustrates the adjunction operation.

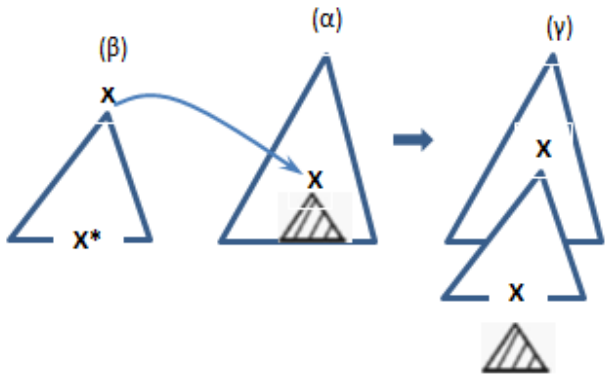

Figure 2. Illustration of adjunction operation.

Analysis and derived trees

The intermediate trees generated when applying substitution and adjunction operations are analytic trees. Full analysis tree are trees whose all leaf nodes are labeled with nonterminal symbols. Hence, syntactic analysis of a sentence starts from an elementary tree whose root node is an axiom and searches for a full analytic tree whose leaf nodes are corresponding to the words in the sentence.

Figure 3a shows an example of syntactic derivation of the sentence "John always laughs". If α_{John}, α_{always} and α_{laughs} are the trees for *John*, *always* và *laughs*, respectively, then this derivation uses two rewriting rules of LTAG formalism as follows:

- The tree α_{John} substitutes the leaf node that is labeled with symbol NP of the tree α_{laughs} to generate the analytic tree shown in Figure 3b*;*
- The auxiliary tree α_{always} is inserted at the node VP of the analytic tree derived from previous substitution step to generate the derived tree illustrated in Figure 3c.

Fore context-free grammar, rewriting rules can be derived by inspecting the syntactic trees. For TAG grammar, it is not possible to know the rewriting rules for generating an analytic tree by inspecting the tree. Therefore, in LTAG grammar, a special structure called derived tree is used to record the operations for generating analytic tree from elementary trees. Each node of a derived tree refers by name to an elementary tree. Each arc of a derived tree represents an adjunction operation using a dashed line or a substitution operation using a solid line. Besides, every node to which the

rewriting operations are applied is marked by a Gorn[1] address. The derived tree for the sentence *John always laughs* is illustrated in Figure 3c.

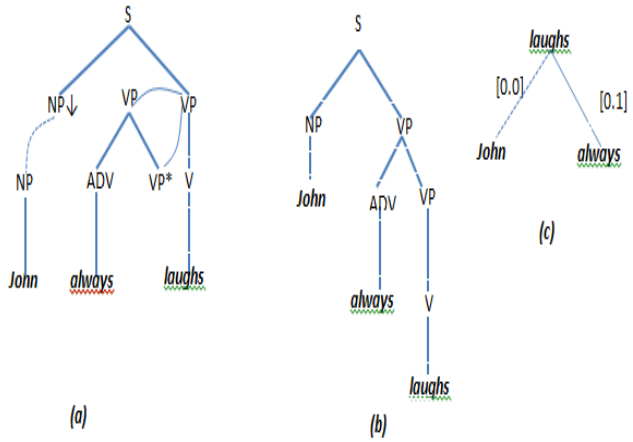

Figure 3. Example of derivation with substitution and adjunction in TAG grammar.

When building TAG grammar for a natural language, some principles are used. First, TAG grammar is lexicalized: every elementary tree has a leaf node attached to a lexical unit called lexical anchor. Second, each initial tree in LTAG grammar represents a projection component of an anchor, which supplements the anchor word. Third, elementary trees are minimal. An initial tree must contains an anchor word that is the central word of the primary component of the sentence and all mandatory projected components of the anchor word [8]. All the auxiliary components, which are recursively added, will be constructed by using adjunction operation on the auxiliary trees. When constructing a sentence, the substitution is corresponding to attaching arguments to a predicate and the adjunction is equivalent to adding auxiliary components. Therefore, a derived tree represents semantic dependent relationships among words in the tree. This explains the wide acceptance of using derived trees as an interface between syntax and semantics in sematic approaches in LTAG grammar. LTAG grammar is classified as mildly context-sensitive grammar. Therefore, its generation capability is stronger than context-free grammar; that makes LTAG grammar easily transform to unified grammar formalisms. LTAG grammar formalism is suitable for linguistic applications as the properties of LTAG grammar allow for naturally describing syntactic symptoms. LTAG grammar has been selected to model Vietnamese grammar where LTAG syntactic analyzer has been fine tuned for Vietnamese.

3. Vietnamese Dictionary

Vietnames machine readable dictionary[2] developed by the project KC.01.01/06-10 contains 35.000 word item with 41700 meanings. The corpus model is based on LMS standard developed by ISO/TC 37/Sc 4. LMF is organised into packages that allow for specifying linguistic information at different levels.

Each lexical item of a dictionary consists of three types of information: morphology information, syntactic information and sematic information. Morphology information describes the word structure. Syntactic information describes word types and sub-types; sub-categorization frame; arguments of predicates, syntactic functions and components of parameters. Semantic information describes logic constraints. The dictionary uses XML encoding for the sake of information exchange between different systems, language comparison research and future updates.

Observations on structure of the Vietneamse dictionary shows that each lexical item takes either of the two forms: The first form, the word is not a verb then there is only information about word types and sub-types. Figure 4 shows the XML description of the noun "*đế quốc*" in the dictionary. The second form, the word is a verb then there is information about word types and sub-types and predicate-argument relationships. Figure 5 shows the XML description of the verb "*đi*" in the dictionary.

```
<Entry>
        <HeadWord>đế quốc</HeadWord>
                <Morphology>
                <WordType>compound
word</WordType>
        </Morphology>
        <Syntactic>
                <Category>A</Category>
                <SubCategory>Ap</SubCategory>
```

[1] Gorn address is recursively defined as follows: address of the root node is 0, k^{th} child node k of a node with address j takes address j.k.

[2] http://vlsp.vietlp.org:8080/demo/?page=vcl

```
        </Syntactic>
        <Semantic>
        ……
        </Semantic>
</Entry>
```

Figure 4. XML description of a noun.

```
1.      <Entry>
2.   <HeadWord>đi</HeadWord>
3.      <Morphology>
4.      <WordType>simple word</WordType>
5.      </Morphology>
6.      <Syntactic>
7.      <Category>V</Category>
8.      <SubCategory>Vt</SubCategory>
9.      <SubcategorizationFrame
              val="Sub+V+Obj"/>
10.     <SyntacticArgument>
11.     <feat att="syntacticFunction" val="Sub"/>
12.     <feat att="syntacticConstituent" val="NP"/>
13.     </SyntacticArgument>
14.     <SyntacticArgument>
15.     <feat att="syntacticFunction" val="Obj"/>
16.     <feat att="syntacticConstituent" val="PP"/>
17.     </SyntacticArgument>
18.     <Before>R: đang</Before>
19.     </Syntactic>
20.     <Semantic>
        ……. …..
21.     </Semantic>
22.     </Entry>
```

Figure 5. XML description of a verb.

4. Elementary Tree Construction Algorithm

An algorithm for constructing the elementary trees for LTAG grammar has been developed based on predicate-argument relationships embodied in morphology and syntactic information of lexical items in the dictionary. The algorithm is fine tuned for the Vietnamese dictionary. Followings are the fundamental steps of the algorithm:

Step 1: For every lexical item, construct a mapping table that maps word types to corresponding syntatic components.

Step 2. For each lexical item, check if there exists sub-categorization frame in the syntactic tag (<*Syntactic*>):

- If exists, construct three types of elementary trees: Type-1 tree is the elementary tree that contains word phrase, word type and lexicon (e.g. "(*VP (V đi)*)"); Type-2 rree is the elementary that contains word phrase, word type, lexicon and folowing arguments (e.g. "(*VP (V đi) (+PP)*)"). The type-2 tree exists only if folowing arguments exist. Type-3 tree is the elementary tree that contains syntactic components of a lexical item (e.g. "*(S (+NP) (VP (V đi) (+PP)))*")

- If not exists, construct trees that contains word phrase, word type and lexicon (Type-1 tree)

The details of the algorithm is presented in the Algorithm 1.

Algorithm 1 BuildTree(Lexical Item)
SyntacticArgument[]: Array of arguments of a verb
Headword: Lexicon
SubcategorizationFrame: Syntactic frame of a lexical item that is a verb
SyntacticComponents {NP,PP,AP,NP,VP,NP,RP,QP} correspond to wordtypes
Input: Lexical Item
Ouput: Spin Elementary Tree
Begin

//BUILD TYPE-3 TREE

1 **Begin**
2 **If** (*SubcategorizationFrame*!=∅)
3 **Begin**
4 *SyntacticArgument* ← (Each argument from left-to-right ∈ *SubcategorizationFrame*)
5 *Tree* = *S*+ syntactic component corresponding to head argument of headword
6 (*SyntacticArgumen*[0]) + syntactic component corresponding to word type + word type + *headword*
7 Construct the tree *T* in *Type-3Tree*, the beginning part of each tree *T* is *Tree* and its end part is a string *str* that contains the labels of syntactic components of the arguments after *headword*. Each argument may contains more than one component labels so there might be

more than one strings str. Therefore, *Type-3Tree* might have more than one tree *T*.

8 **End;**

9 **End.**

//BUILD TYPE-2 TREE

1 **Begin**

2 **If** (*SubcategorizationFrame*!=∅)

3 **Begin**

4 *SyntacticArgumen* ← (Each argument from left- to-right ∈ *SubcategorizationFrame*)

5 **if** (*SyntacticArgument*[1]!=∅)

6 **Begin**

7 *Tree* = Syntactic component corresponding to word type + word type + *headword*

8 Construct tree *T* in *Type-2Tree*, the beginning part of each tree *T* is *Tree* and its end part is a string *str* that contains the labels of syntactic components of the arguments after *headword*. Each argument may contains more than one component labels so there might be more than one strings *str*. Therefore, *Type-2Tree* might have more than one tree *T*.

9 **End;**

10 **End;**

11 **End.**

//BUILD TYPE-1 TREE

1 **Begin**

2 *Type-1Tree*= Label of syntactic component + word type + *headword*

3 **return** (*Type-1Tree*∪ *Type-2Tree* ∪ *Type-3Tree*)

4 **End.**

Applying the Algorithm 1 for lexical item ”đi” described in Figure 4 results in:

Type-3 Tree= (S (+NP) (VP (V đi); str_1= (+PP)));
Tree3 Tree= (S (+NP) (VP(V đi) (+PP))).

Type-2 Tree= (VP (V đi); str_1= (+PP)), T=(VP(V đi) (+PP)). Type-2 Tree contains (VP (V đi) (+PP)).

Type-1 Tree= (VP (V đi)).

The Algorithm 1 constructs initial trees from a Vietnamese machine-readable dictionary. The description of auxiliary components of a lexical item in the dictionary is not sufficient to construct auxiliary trees.

5. Results and Discussion

In the experiment, two sets of elementary trees have been generated for a number of verbs.. The first set of initial trees has been generated using the method proposed in this paper. The second set of elementary has been generated from VietTreeBank. Trees from the two sets of elementary generated for each verb were compared to each other in terms on intersection and bias. Table 1 shows details of the experiment results.

Table 1. Comparison of Spin elementary trees generated from dictionary and those of VietTreeBank

Experiment Step	Number of trees from dictionary	Intersected Word (Both Word)	Number of trees from dictionary that has anchor word intersected (allXml Tree)	Number of trees from VietTreeBank that has anchor word intersected (allBank Tree)	Number of intersected trees (Both Tree)	Similarity Ratio of LTAG compared to VietTreeBanbank	Similarity Ratio of LTAG compared to the dictionary	Average number of words per intersected tree
Common word type	56386	1469	6355	3701	1481	40.02%	23.30%	0.999
Detailed word type (Vt,Vu)	59243	1469	6963	3701	892	24.10%	12.81%	1.65

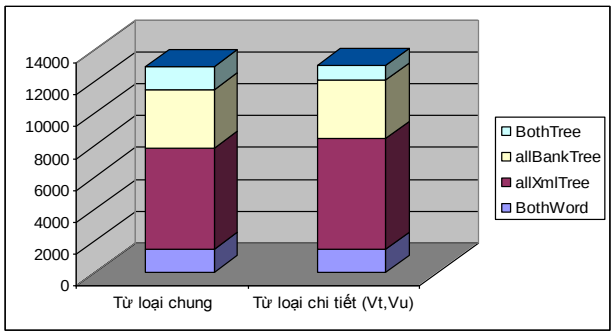

Figure 6. Comparison diagram on elementary trees.

Figure 6 shows the significant difference of intersected elementary trees generated from the dictionary and VietTreeBank. In the experiment, the common and detailed word type have been captured from 1469 anchor words as verbs. The experiment indicates that word types and subtypes in VietTreeBank are not consistently annotated for lexical items. Therefore, the labels of word types for lexicons in VietTreeBank need to be standardized. Statistics in Table 1 indicates that the number of intersected words is approximately equal to the number of intersected initial trees for common word type category. The proportions of the intersected initial trees to the trees from the dictionary or VietTreeBank are small. The initial trees derived from the dictionary do not provide syntactic information as rich as those of VietTreeBank do. The elementary trees of VietTreeBank do not cover those of the dictionary. In the dictionary, there is only sub-categorization frame for verbs not for other predicates (noun, adjective, preposition). This results in small ratio of intersection of trees. Therefore, sub-categorization frames of the dictionary need to be enriched.

The annotation errors are unavoidable in big treebanks. The errors occur in syntactic analysis trees make elementary trees invalid. An elementary tree is considered as invalid if it does not hold for a certain linguistic requirement. For Vietnamese, invalid elementary trees can be removed using grammar rules. For example, in Vietnamese, an adjective (or adjective phrase) cannot be the next central node for a verb etc. Therefore, an elementary tree is considered invalid if there exists an adjective, a noun or a preposition being central node of a verb phase or other types of phrase. Another case of invalid tree in Vietnamese is that an initial tree is considered invalid if its central node has more than four mandatory arguments (see Figure 7). In VieTreeBank, there are some elementary trees that have more than 4 arguments. Whereas, the maximum number of arguments of an elementary tree constructed from the dictionary is three.

The list of initial trees derived from VietTreeBank that cannot be derived from the dictionary provides linguistic knowledge to filter out certain elementary trees that are invalid for grammar rules extracted from VietTreeBank. For instance, following trees

are invalid: (VP (A tạm)); (S (VP (N nói)) (+NP)); (VP (N tai nạn) (+n));(VP (N nước)).

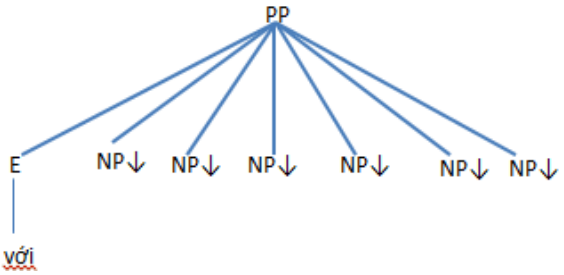

Figure 7. An example of invalid initial tree.

6. Conclusions

This paper presents a method for automatically constructing LTAG grammars from a Vietnamese machine-readable dictionary, which can be applied to Vietnamese syntactic analysis problem. The elementary trees constructed from the Vietnamese dictionary also provide linguistic knowledge to filter out invalid elementary trees extracted from VietTreeBank. The experiment indicates the need for richer sub-categorization frames for the cases other than verbs in VietTreeBank. The experiment also shows that labels for word types and subtypes used in VietTreeBank need to be standardized using common criteria. The results show potentials for using Vietnamese dictionaries together with VietTreeBank to improve the quality of automatic LTAG grammars construction systems.

References

[1] A. Kinyon and C. A. Prolo, *A classification of grammar development strategies*, In Proceedings of the Workshop on Grammar Engineering and Evaluation, pages 43—49, Taipei, Taiwan, 2002.

[2] Lê Hồng Phương, Nguyễn Thị Minh Huyền, Nguyễn Phương Thái, Phan Thị Hà, *Trích rút tự động văn phạm LTAG cho tiếng Việt*, Tạp chí Tin học và Điều khiển học, T.26. S2. 153-171, 2010.

[3] A. K. Joshi and Y. Schabes, *Handbooks of Formal Languages and Automata, chapter Tree Adjoining Grammars*, Springer-Verlag, 1997.

[4] A. K. Joshi, L. S. Levy, and M. Takahashi, *Tree adjunct grammars*, Journal of the Computer and System Sciences, 10:136–165, 1975.

[5] A. Abeillé, *Treebanks - Building and Using Parsed Corpora*, Dordrecht: Kluwer Academic Publishers, 2003.

[6] C. Doran, B. Hockey, A. Sarkar, and B. Srinivas, *Evolution of the XTAG system*, In A. Abeillé and O. Rambow, editors, Tree adjoining grammars, pages 371–404. Stanford CSLI, 2000.

[7] E. V. de la Clergerie, B. Sagot, L. Nicolas, and M.-L. Guénot, *FRMG: évolution d'un analyseur syntaxique TAG du franc¸ais* ", In Workshop ATALA de IWPT 2009, Paris, 2009.

[8] Y. Parmentier, *SemTAG: Une plate-forme pour le calcul sémantique à partir de grammaires d'arbres adjoints*, PhD thesis, Université Henri Poincaré, Nancy I, 2007.

[9] R. Frank, *Phrase Structure Composition and Syntactic Dependencie* , MIT Press, Boston, 2002.

First Author: Dr. Phan Thi Ha is currently a lecturer in the Department of Information Systems at Posts and Telecommunications Institute of Technology in Vietnam. She received a B.Sc.in Math & Informatics, a M.Sc. in Mathematic Guarantee for Computer Systems and a PhD. in Information Systems in 1994, 2000 and 2013, respectively. Her research interests include machine learning, natural language processing and mathematics applications.

Second Author: Dr. Ha Hai Nam is an Associate Professor in the Department of Information Systems at Posts and Telecommunications Institute of Technology in Vietnam. Dr. Nam studied for a PhD. in Computer Science at Newcastle University where he explored the application of optimisation techniques to smart graphics. His research interests include optimisation, machine learning and distributed computing.