Automated Extraction of Tree Adjoining Grammars from a Treebank for Vietnamese

Phuong Le-Hong[1,2], **Thi Minh Huyen Nguyen**[2], **Phuong Thai Nguyen**[2], **Azim Roussanaly**[1]
[1] LORIA, Nancy, France, [2] Vietnam National University, Hanoi, Vietnam
{lehong,azim}@loria.fr, huyenntm@hus.edu.vn, thainp@vnu.edu.vn

Abstract

In this paper, we present a system that automatically extracts lexicalized tree adjoining grammars (LTAG) from treebanks. We first discuss in detail extraction algorithms and compare them to previous works. We then report the first LTAG extraction result for Vietnamese, using a recently released Vietnamese treebank. The implementation of an open source and language independent system for automatic extraction of LTAG grammars is also discussed.

1 Introduction

Grammars in general and lexicalized tree adjoining grammars in particular are one of the most important elements in the natural language processing (NLP). Since the development of hand-crafted grammars is a time consuming and labor intensive task, many studies on automatic and semi-automatic grammar development have been carried out during last decades.

After decades of research in NLP mostly concentrated on English and other well-studied languages, recent years have seen an increased interest in less common languages, notably because of their growing presence on the Internet. Vietnamese, which belongs to the top 20 most spoken languages, is one of those new focuses of interest. Obstacles remain, however, for NLP research in general and grammar development in particular: Vietnamese does not yet have vast and readily available constructed linguistic resources upon which to build effective statistical models, nor reference works against which new ideas may be experimented.

Moreover, most existing research so far has been focused on testing the applicability of existing methods and tools developed for English or other Western languages, under the assumption that their logical or statistical well-foundedness guarantees cross-language validity, while in fact assumptions about the structure of a language are always made in such tools, and must be amended to adapt them to different linguistic phenomena. For an isolating language such as Vietnamese, techniques developed for flexional languages cannot be applied "as is".

The primary motivation to develop a system that can automatically extract an LTAG grammar for the Vietnamese language is the need of a rich statistical information and wide-coverage grammar which may contribute more effectively in the development of basic linguistic resources and tools for automatic processing of Vietnamese text.

We present in this article a system that automatically extracts lexicalized tree adjoining grammars from treebanks. We first discuss in detail the extraction algorithms and compare them to previous works. We then report the first LTAG extraction result for Vietnamese, using the recently released Vietnamese treebank. The implementation of an open source and language independent system for automatic extraction of LTAG grammars from treebanks is also discussed.

2 Previous works on extracting grammars from treebanks

There has been much work done on extracting treebank grammars in general and LTAG grammars in particular from annotated corpora, all of these works are for common languages. Xia developed the uniform method of grammar extraction for English, Chinese and Korean (Xia et al., 2000; Xia, 2001). Chiang developed a system for extracting an LTAG grammar from English Penn Treebank and used it for statistical parsing with LTAG (Chiang, 2000). Chen extracted TAGs from English Penn Treebank (Chen and Vijay-Shanker, 2000; Chen et al., 2006) and there are other works based on Chen's approach such as Johansen (Johansen, 2004) and Nasr (Nasr, 2004) for French, and Habash for Arabic (Habash and Rambow, 2004). Neumann extracted lexicalized tree grammars for English from English Penn Treebank and for German from NEGRA treebank (Neumann, 2003). Bäcker extracted an LTAG gramar for German, also from the NEGRA corpus and used it for supertagging (Bäcker and Harbusch, 2002). Park extracted LTAG grammars for Korean from Korean Sejong Treebank (Park, 2006).

3 Vietnamese treebank

Recently, a group of Vietnamese computational linguists has been involved in developing a treebank for Vietnamese (Nguyen et al., 2009), and it is also the first treebank on which our extraction system was used.

The construction of a Vietnamese treebank is a branch project of a national project which aims to develop basic resources and tools for Vietnamese language and speech processing[1]. The raw texts of the treebank are collected from the social and political sections of the Youth online daily newspaper. The corpus is divided into three sets corresponding to three annotation levels: word-segmented, POS-tagged and syntax-annotated set. The syntax-annotated corpus, a subset of the POS-tagged one, is currently composed of $10,471$ sentences ($225,085$ tokens). Sentences range from 2 to 105 words, with an average length of 21.75 words. There are $9,314$ sentences of length 40 words or less. The tagset of the treebank has 38 syntactic labels (18 part-of-speech tags, 17 syntactic category tags, 3 empty categories) and 17 function tags. For details, please refer to (Nguyen et al., 2009).

[1]Project "Vietnamese Language and Speech Processing"

No.	Category	Description
1.	S	simple declarative clause
2.	VP	verb phrase
3.	NP	noun phrase
4.	PP	preposition phrase
5.	N	common noun
6.	V	verb
7.	P	pronoun
8.	R	adverb
9.	E	preposition
10.	CC	coordinating conjunction

Table 1: Treebank tags in examples.

The meanings of the tags that appear in this paper are listed in Table 1.

4 Extraction algorithms

In general, our work on extracting an LTAG grammar for Vietnamese follows closely the method of grammar extraction originally proposed by Xia (Xia, 2001). The extraction process has three steps: first, phrase-structure trees are converted into LTAG derived trees; second, the derived trees are decomposed into a set of elementary trees conforming to their three predefined prototypes; and third, invalid extracted elementary trees are filtered out using linguistic knowledge.

4.1 Building LTAG derived trees

The phrase structures in the Vietnamese treebank follow the English Penn Treebank bracketed style format which are not based on the LTAG formalism. They may have different formats from the LTAG derived trees which distinguish heads, arguments and adjuncts. Therefore, we first have to convert the phrase structures of the treebank into derived trees.

In this step, we first classify each node in a phrase-structure tree into three types, head, argument or modifier, and then build a derived tree by adding intermediate nodes so that at each level of the tree, the nodes satisfy exactly one of the following relations (Xia, 2001):

- *predicate-argument relation*: there are one or

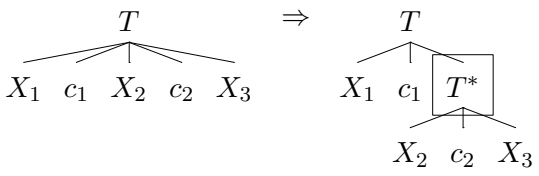

Figure 1: Conjunction groups transformation by Algorithm 1.

more nodes, one is the head, the rest are its arguments;

- *modification relation*: there are exactly two nodes, one node is modified by the other;
- *coordination relation*: there are exactly three nodes, in which two nodes are coordinated by a conjunction.

In order to find heads of phrases, we have constructed *a head percolation table* (Magerman, 1995; Collins, 1997) for the Vietnamese treebank. This table is used to select the head child of a node. In addition, we have also constructed *an argument table* to determine the types of arguments that a head child can take. The argument table helps explicitly mark each sibling of a head child as either an argument or an adjunct according to the tag of the sibling, the tag of the head child, and the position of the sibling with respect to the head child. Together with *the tagset table*, these three tables constitute the Vietnamese treebank-specific information that is required for the extraction algorithms[2].

Since the conjunction structures are different from the argument and modifier structures, we first recursively bracket all conjunction groups of a treebank tree by Algorithm 1 and then build the full derived tree for the resulting tree by Algorithm 2.

Figure 1 shows a tree with conjunction groups before and after being processed by Algorithm 1 where c_i are coordinating conjunctions and X_i are conjunction groups. Figure 2 shows a realisation of Algorithm 2 where A_i are arguments of the head child H of T and M_i are modifiers of H.

These two algorithms use the function INSERT-NODE$(T, \mathcal{L})$ shown in Algorithm 3 to insert an intermediate node between a node T and a list of its child

Algorithm 1 PROCESS-CONJ(T)

Require: A tree T
Ensure: T with conjunctions processed
1: **for** $K \in T.\texttt{kids}$ **do**
2: **if** IS-PHRASAL(K) **then**
3: $K \leftarrow$ PROCESS-CONJ(K);
4: **end if**
5: **end for**
6: $(\mathcal{C}_1, \ldots, \mathcal{C}_k) \leftarrow$ CONJ-GROUPS$(T.\texttt{kids})$;
7: **for** $i = 1$ to k **do**
8: **if** $\|\mathcal{C}_i\| > 1$ **then**
9: INSERT-NODE$(T, \mathcal{C}_i)$;
10: **end if**
11: **end for**
12: **if** $k > 2$ **then**
13: **for** $i = k$ **downto** 3 **do**
14: $\mathcal{L} \leftarrow \mathcal{C}_{i-1} \cup c_{i-1} \cup \mathcal{C}_i$;
15: $T^* \leftarrow$ INSERT-NODE$(T, \mathcal{L})$;
16: $\mathcal{C}_{i-1} \leftarrow T^*$;
17: **end for**
18: **end if**
19: **return** T;

nodes $\mathcal{L}$. This new node is a child of T, has the same label as T and has $\mathcal{L}$ as the list of its kids. The function CONJ-GROUPS$(\mathcal{L})$ returns k groups of components $\mathcal{C}_i$ of $\mathcal{L}$ which are separated by $k-1$ conjunctions $c_1, \ldots, c_{k-1}$. The function NEW-NODE(l) returns a new node with label l.

The Algorithm 2 uses several functions that are relatively self-explained. The function HEAD-CHILD(X) selects the head child of a node X according to a head percolation table. The head percolation table for the Vietnamese treebank is shown in the Table 4. The function IS-LEAF(X) checks whether a node X is a leaf node or not. The function IS-PHRASAL(X) checks whether X is a phrasal node or not.[3] The function ARG-NODES$(H, \mathcal{L})$ (respectively, MOD-NODES$(H, \mathcal{L})$) returns a list of nodes which are arguments (respectively modifiers) of a node H. The list $\mathcal{L}$ contains all sisters of H.

For example, Figure 3 shows the phrase structure of a sentence extracted from the Vietnamese treebank "*Họ sẽ không chuyển hàng xuống thuyền vào*

[2]To our best knowledge, this is the first time such tables are published for the Vietnamese treebank.

[3]A phrasal node is defined to be a node which is not a leaf or a preterminal. This means that it must have two or more children, or one child that is not a leaf.

Algorithm 2 BUILD-DERIVED-TREE(T)

Require: A tree T whose conjunctions have been processed
Ensure: A derived tree whose root is T

if (not IS-PHRASAL(T)) **then**
  return T;
end if
$H \leftarrow$ HEAD-CHILD(T);
if not IS-LEAF(H) **then**
  for $K \in T.\texttt{kids}$ **do**
    $K \leftarrow$ BUILD-DERIVED-TREE(K);
  end for
  $\mathcal{A} \leftarrow$ ARG-NODES$(H, \mathcal{L})$;
  $\mathcal{M} \leftarrow$ MOD-NODES$(H, \mathcal{L})$;
  $m \leftarrow \|\mathcal{M}\|$;
  if $m > 0$ **then**
    $\mathcal{L} \leftarrow \{H\} \cup \mathcal{A}$;
    $T^* \leftarrow$ INSERT-NODE$(T, \mathcal{L})$;
  end if
  $(M_1, M_2, \ldots, M_m) \leftarrow \mathcal{M}$;
  for $i = 1$ **to** $m - 1$ **do**
    $\mathcal{L} \leftarrow \{M_i, T^*\}$;
    $T' \leftarrow$ INSERT-NODE$(T, \mathcal{L})$;
    $T^* \leftarrow T'$;
  end for
end if
return T;

ngày mai."[4] The head children of phrases are circled.

The derived tree of the sentence given by Algorithm 2 is shown in Figure 4, the inserted nodes are squared.

4.2 Building elementary trees

At this step, each derived tree is decomposed into a set of elementary trees. The recursive structures of the derived tree are factored out and will become auxiliary trees, the remaining non-recursive structures will be extracted as initial trees.

Extracted elementary trees fall into one of three prototypes according to the relation between the anchor and other nodes, as shown in Figure 5.

The extraction process involves copying nodes from the derived tree for building elementary trees. The result of extraction process is three sets of elementary trees: $\mathcal{S}$ contains spine trees, $\mathcal{M}$ contains modifier trees and $\mathcal{C}$ contains conjunction trees.

[4]They will not deliver the goods to the boat tomorrow.

Algorithm 3 INSERT-NODE$(T, \mathcal{L})$

Require: A tree T and its children list $\mathcal{L}$
Ensure: A new child node T^* of T whose kids are $\mathcal{L}$

$T^* \leftarrow$ NEW-NODE$(T.\texttt{label})$;
$T^*.\texttt{kids} \leftarrow \mathcal{L}$;
$T.\texttt{kids} \leftarrow T.\texttt{kids} \setminus \mathcal{L}$;
$T.\texttt{kids} \leftarrow T.\texttt{kids} \cup \{T^*\}$;
return T^*;

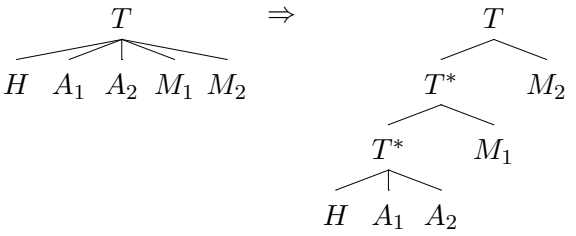

Figure 2: An example of derived tree realisation

To build elementary trees from a derived tree T, we first find the head path[5] $\{H_0, H_1, \ldots, H_n\}$ of T. For each parent P and its head child H, we get the list $\mathcal{L}$ of sisters of H and determine the relation between H and $\mathcal{L}$. If the relation is coordination, a conjunction tree will be extracted; if the relation is modification, a modifier tree will be extracted; otherwise, the relation is predicate-argument and a spine tree will be extracted. Algorithm 4 shows the extraction algorithm.

Algorithm 5 shows the function for building a spine tree. The function MERGE-LINK-NODES(T) merges all link nodes of a spine tree into one node (see Figure 7). Algorithms 6 and 7 are functions which respectively build modifier and conjunction trees.

For example, from the derived tree shown in Figure 4, 9 trees are extracted by algorithms as shown in Figure 6 and Figure 7.

[5]A *head path* starting from a node T in a derived tree is the unique path from T to a leaf node where each node except T is the head child of its parents. Here $H_0 \equiv T$ and H_j is the parent of its head child H_{j+1}. A node on the head path is called a *link node* if its label is the same as that of its parent.

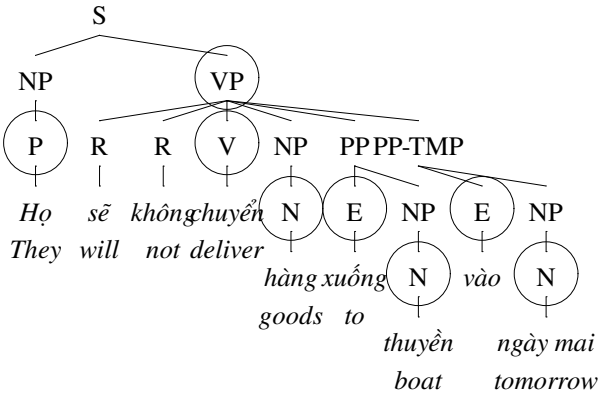

Figure 3: A treebank tree.

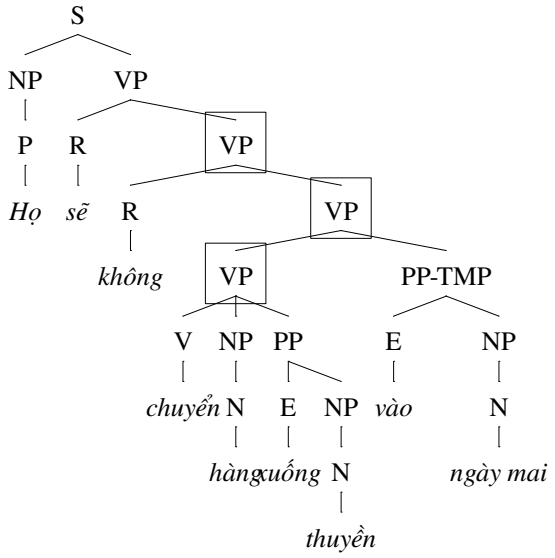

Figure 4: The derived tree of the treebank tree in Figure 3.

4.3 Filtering out invalid trees

Annotation errors are inevitable for any treebank. The errors in parse trees will result in wrong elementary trees. An elementary tree is called invalid if it does not satisfy some linguistic requirement. We have construct some linguistic rules for filtering out invalid elementary trees. For example in Vietnamese, an adjective (or an adjectival phrase) can be an argument of a noun (or a noun phrase), however, they must be always on the right of the noun. Thus if there is an adjective on the left of a noun of an extracted spine tree, the tree is invalid and it must be filtered out.

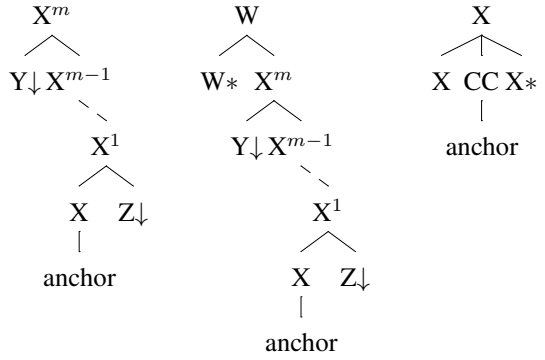

Figure 5: Prototypes of spine trees (predicate-argument relation) and auxiliary trees (modification and coordination relation).

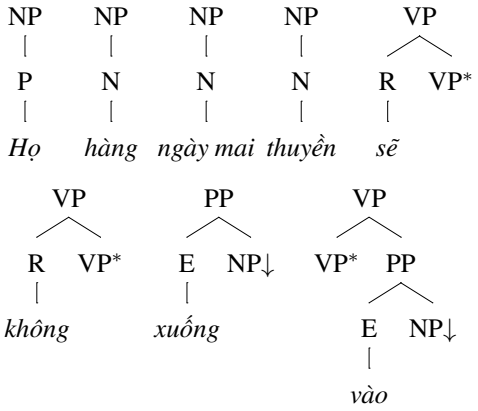

Figure 6: Extracted elementary trees.

4.4 Comparison with previous work

As mentioned above, our approach for LTAG extraction follows the uniform method of grammar extraction proposed by Xia (Xia, 2001). Nevertheless, there are some differences between our design and implementation of extraction algorithms and that of Xia.

First, in the building derived tree step, we first recursively bracket all conjunction groups of the tree before fully bracketing the arguments and modifiers of the resulting tree. We think that this approach is easier to understand and implement since conjunction structures are different from argument and modifier structures. Second, in the elementary tree decomposition step, we do not split each node in the derived tree into the top and bottom parts as it was done in the approach of Xia. In our implementation, the nodes are directly copied to build extracted trees. Third, the tree extraction process is broken into func-

Algorithm 4 BUILD-ELEMENTARY-TREES(T)

Require: T is a derived tree
Ensure: Sets $\mathcal{S}, \mathcal{M}, \mathcal{C}$ of elementary trees.

if (not IS-PHRASAL(T)) **then**
  return ;
end if
$\{H_0, H_1, \ldots, H_n\} \leftarrow$ HEAD-PATH(T);
$ok \leftarrow$ `false`;
$P \leftarrow H_0$;
for $j \leftarrow 1$ **to** n **do**
  $\mathcal{L} \leftarrow$ SISTERS(H_j);
  if $|\mathcal{L}| > 0$ **then**
    `Rel` $\leftarrow$ GET-RELATION($H_j, \mathcal{L}$);
    if `Rel` $=$ `Coordination` **then**
      $\mathcal{C} \leftarrow \mathcal{C} \cup$ BUILD-CONJ-TREE(P);
    end if
    if `Rel` $=$ `Modification` **then**
      $\mathcal{M} \leftarrow \mathcal{M} \cup$ BUILD-MOD-TREE(P);
      if $j = 1$ **then**
        $\mathcal{S} \leftarrow \mathcal{S} \cup$ BUILD-SPINE-TREE(P);
        $ok \leftarrow$ `true`;
      end if
    end if
    if `Rel` $=$ `Argument` **then**
      if not ok and not IS-LINK-NODE(P) **then**
        $\mathcal{S} \leftarrow \mathcal{S} \cup$ BUILD-SPINE-TREE(P);
        $ok \leftarrow$ `true`;
      end if
    end if
  else
    if not IS-LINK-NODE(P) and IS-PHRASAL(P) **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup$ BUILD-SPINE-TREE(P);
    end if
  end if
  $P \leftarrow H_j$;
end for

Algorithm 5 BUILD-SPINE-TREE(T)

Require: T is a derived tree
Ensure: a spine tree

$T_c \leftarrow$ COPY(T);
$P \leftarrow T_c$;
$H \leftarrow$ `NULL`;
repeat
  $H \leftarrow$ HEAD-CHILD(P);
  $\mathcal{L} \leftarrow$ SISTERS(H);
  if $|\mathcal{L}| > 0$ **then**
    `Rel` $\leftarrow$ GET-RELATION($H, \mathcal{L}$);
    if `Rel` $=$ `Argument` **then**
      for $A \in \mathcal{L}$ **do**
        BUILD-ELEMENTARY-TREES(A);
        A.`kids` $\leftarrow \emptyset$;
        A.`type` $\leftarrow$ `Substitution`;
      end for
    else
      for $A \in \mathcal{L}$ **do**
        P.`kids` $\leftarrow P$.`kids` $\setminus A$;
      end for
    end if
  end if
  $P \leftarrow H$;
until ($H =$ `NULL`)
return MERGE-LINK-NODES(T_c);

Algorithm 6 BUILD-MOD-TREE(T)

Require: T is a derived tree
Ensure: a modifier tree

$T_c \leftarrow$ COPY(T);
$H \leftarrow$ HEAD-CHILD(T_c);
H.`kids` $\leftarrow \emptyset$;
H.`type` $\leftarrow$ `Foot`;
$M \leftarrow$ MODIFIER(H);
$T' \leftarrow$ BUILD-SPINE-TREE(M);
if $|M$.`kids`$| > 1$ **then**
  BUILD-ELEMENTARY-TREES(M);
end if
$M \leftarrow T'$;
return T_c;

tions, each function builds a type of elementary trees and they can be called mutually by each other to repeat the extraction process for the subtrees whose roots are not yet visited. In spite of using recursive functions, our extraction algorithms are carefully designed so that there is no redundant or repeating function calls: each node is assured to be visited one time. The "divide and conquer" approach in algorithm design has been shown to be efficient and easy to optimise.

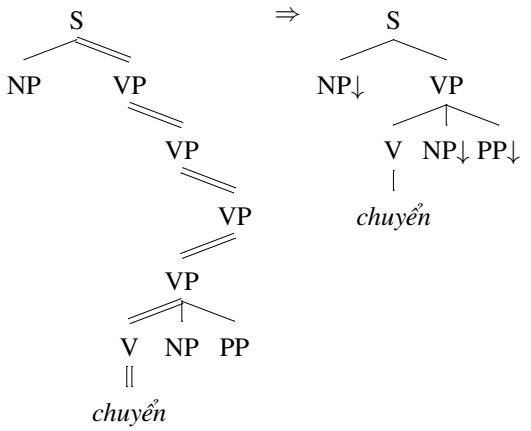

Figure 7: Merge link nodes to get a spine tree. The head path of the tree is marked by double lines.

Algorithm 7 BUILD-CONJ-TREE(T)

Require: T is a derived tree
Ensure: a conjunction tree
1: $T_c \leftarrow$ COPY(T);
2: $H \leftarrow$ HEAD-CHILD(T_c);
3: BUILD-ELEMENTARY-TREES(H);
4: $K \leftarrow$ COORDINATOR(H);
5: BUILD-ELEMENTARY-TREES(K);
6: H.`kids` $\leftarrow \emptyset$;
7: H.`type` $\leftarrow$ `Foot`;
8: K.`kids` $\leftarrow \emptyset$;
9: K.`type` $\leftarrow$ `Substitution`;
10: **return** T_c;

Category	**Original tags**	**Tags in G_2**
noun phrases	NP/WHNP	NP
adjective phrases	AP/WHAP	AP
adverbial phrases	RP/WHRP	RP
preposition phrases	PP/WHPP	PP
clauses	S/SQ	S

Table 2: Some tags in the Vietnamese treebank tagset are merged into a single tag.

5 Experiments

We ran extraction algorithms on the Vietnamese treebank and extracted two treebank grammars. The first one, G_1, uses the original tagset of the treebank. The second one, G_2, uses a reduced tagset, where some tags in the treebank are merged into a single tag, as shown in Table 2. The grammar G_2 is smaller than G_1 and it is presumable that the sparse data problem is less severe when G_2 is used. Furthermore, it was shown that the size of the extracted grammar is important for Lightweight Dependency Analysis (LDA) and supertagging (Bangalore and Joshi, 1999).

We count the number of elementary trees and tree templates. The sizes of the two grammars are in Table 3. Recall that a template is an elementary tree without the anchor word.

Type	**# of trees**	**# of templates**
G_1	**46,382**	**2,317**
Spine trees	24,973	1,022
Modifier trees	21,309	1,223
Conjunction trees	100	72
G_2	**46,102**	**2,113**
Spine trees	24,884	952
Modifier trees	21,121	1,093
Conjunction trees	97	68

Table 3: Two LTAG grammars extracted from the Vietnamese treebank.

There are 15,035 unique words in the treebank and the average number of elementary trees that a word anchors is around 3.07. We also count the number of context-free rules of the grammars where the rules are simply read off the templates in an extracted LTAG. The extracted grammar G_1 and G_2 respectively has 851 and 727 context-free rules.

In order to evaluate the coverage of the Vietnamese treebank, we count the number of extracted tree templates with respect to size of the treebank. Figure 8 shows the number of templates converges very slowly as the size of the corpus grows, implying that there are many unseen templates. This experiment also implies that the size of the current Vietnamese treebank is not large enough to cover all the grammatical templates of the Vietnamese language.

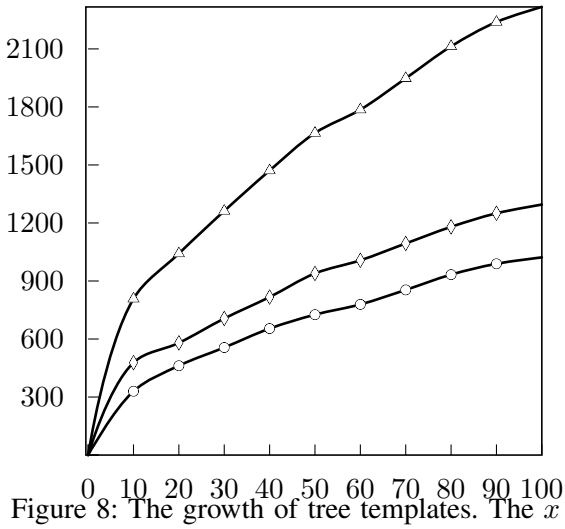

Figure 8: The growth of tree templates. The x axis shows the percentage of the corpus used for extraction, the y axis shows the number of extracted templates ($\triangle$), initial templates (o) and auxiliary templates ($\diamond$).

We have developed a software package that implements the presented algorithms for extracting an LTAG for Vietnamese. The software is written in the Java programming language and is freely distributed under the GNU/GPL license[6]. The software is very efficient in term of extraction speed: it takes only 165 seconds to extract the entire grammar G_1 on an ordinary personal computer. It is very easy to extend the software for use to extract LTAGs from treebanks of other languages since the language-specific information is intensionally factored out of the general framework. In order to use the software on a treebank of a language, user needs to provide the treebank-specific information for that language: a tagset, a head percolation table, and an argument table.

[6] *http://www.loria.fr/∼lehong/tools/vnLExtractor.php*

6 Conclusions

We have presented a system that automatically extracts LTAGs from treebanks. The system has been used to extract an LTAG for the Vietnamese language from the recently released Vietnamese treebank. The extracted Vietnamese LTAG covers the corpus, that is the corpus can be seen as a collection of derived trees for the grammar and can be used to train statistical LTAG parsers directly.

The number of templates extracted from the current Vietnamese treebank converges slowly. This implies that there are many new templates outside the corpus and the current Vietnamese treebank is not large or typical enough to cover all the grammatical templates of the Vietnamese language.

Preliminary experimental parsing results using the LLP2 LTAG parser (Crabbé et al., 2003) show a high complexity of Vietnamese parsing in term of number of parses produced. For example, a test involving 70 sentences of length 15 words or less, parsed using an extracted LTAG grammar gives an average number of parses of 49.6 for a sentence, in which 14 sentences having unique parse. In future work, we plan to evaluate and extend the coverage and performance of both the grammar and parser for Vietnamese in greater detail.

We are currently experimenting the extraction of a French LTAG from a French treebank (Abeillé et al., 2003). We also plan to compare quantitatively syntactic structures of French and Vietnamese. We believe that a quantitative comparison of the two grammars may reveal interesting relations between them since, due to historical reason, by being in contact with the French language, Vietnamese was enriched not only in vocabulary but also in syntax by the calque of French grammar.

Acknowledgement

This work has been carried on in the framework, and with the support of the project QT-09-01, Vietnam National University of Hanoi.

References

Anne Abeillé, Lionel Clément, and François Toussenel. 2003. Building a treebank for French. In *Treebanks:*

Tags	Direction	Priority List
S	Left	S VP AP NP
SBAR	Left	SBAR S VP AP NP
SQ	Left	SQ VP AP NP
NP	Left	NP Nc Nu Np N P
VP	Left	VP V A AP N NP S
AP	Left	AP A N S
RP	Right	RP R T NP
PP	Left	PP E VP SBAR AP QP
QP	Left	QP M
XP	Left	XP X
YP	Left	YP Y
MDP	Left	MDP T I A P R X
WHNP	Left	WHNP NP Nc Nu Np N P
WHAP	Left	WHAP A N V P X
WHRP	Left	WHRP P E T X
WHPP	Left	WHPP E P X
WHXP	Left	XP X

Table 4: Head percolation rules for the Vietnamese treebank.

Building and Using Parsed Corpora. Kluwer, Dordrecht.

Jens Bäcker and Karin Harbusch. 2002. Hidden Markov model-based supertagging in a user-initiative dialogue system. In *Proceedings of TAG+6*, pages 269–278, Universita di Venezia.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

John Chen and K. Vijay-Shanker. 2000. Automated extraction of TAGs from the Penn treebank. In *Proceedings of the Sixth International Workshop on Parsing Technologies*.

John Chen, Srinivas Bangalore, and K. Vijay-Shanker. 2006. Automated extraction of tree-adjoining grammars from treebanks. *Natural Language Engineering*, 12(3):251–299.

David Chiang. 2000. Statistical parsing with an automatically-extracted tree adjoining grammar. In *ACL'00*, pages 456–463, Morristown, NJ, USA.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of ACL*.

Benoît Crabbé, Bertrand Gaiffe, and Azim Roussanaly. 2003. Représentation et gestion de grammaires d'arbres adjoints lexicalisées. *Traitement Automatique des Langues*, 44(3):67–91.

Nizar Habash and Owen Rambow. 2004. Extracting a tree adjoining grammar from the Penn Arabic treebank. In *Proceedings of TALN'04*, Morocco.

Ane-Dybro Johansen. 2004. Extraction des grammaires LTAG à partir d'un corpus étiquetté syntaxiquement. Master's thesis, Université Paris 7.

David M. Magerman. 1995. Statistical decision tree models for parsing. In *Proceedings of ACL*.

Alexis Nasr. 2004. *Analyse syntaxique probabiliste pour grammaires de dépendances extraites automatiquement*. Habilitation à diriger des recherches, Université Paris 7.

Günter Neumann. 2003. A uniform method for automatically extracting stochastic lexicalized tree grammar from treebank and HPSG. In *Treebanks: Building and Using Parsed Corpora*. Kluwer, Dordrecht.

Phuong Thai Nguyen, Luong Vu Xuan, Thi Minh Huyen Nguyen, Van Hiep Nguyen, and Phuong Le-Hong. 2009. Building a large syntactically-annotated corpus of Vietnamese. In *Proceedings of the 3rd Linguistic Annotation Workshop, ACL-IJCNLP*, Singapore.

Jungyeul Park. 2006. Extraction of tree adjoining grammars from a treebank for Korean. In *COLING ACL'06 Student Research Workshop*, pages 73–78, Morristown, NJ, USA.

Fei Xia, Martha Palmer, and Aravind Joshi. 2000. A uniform method of grammar extraction and its applications. In *Proceedings of the joint SIGDAT conference on empirical methods in NLP and very large corpora*, pages 53–62, Morristown, NJ, USA.

Fei Xia. 2001. *Automatic grammar generation from two different perspectives*. Ph.D. thesis, University of Pennsylvania.