THESIS APPROVAL

GRADUATE SCHOOL, KASETSART UNIVERSITY

Doctor of Engineering (Computer Engineering)

DEGREE

Computer Engineering | Computer Engineering

FIELD | **DEPARTMENT**

TITLE: Design and Development of a Lattice Structure Dependency Parser for Under-Resourced Languages

NAME: Mr. Sutee Sudprasert

THIS THESIS HAS BEEN ACCEPTED BY

THESIS ADVISOR

(Associate Professor Asanee Kawtrakul, D.Eng.)

THESIS CO-ADVISOR

(Associate Professor Punpiti Piamsa-Nga, D.Sc.)

DEPARTMENT HEAD

(Assistant Professor Kemathat Vibhatavanij, Ph.D.)

APPROVED BY THE GRADUATE SCHOOL ON _____

DEAN

(Associate Professor Gunjana Theeragool, D.Agr.)

THESIS

DESIGN AND DEVELOPMENT OF A LATTICE STRUCTURE DEPENDENCY PARSER FOR UNDER-RESOURCED LANGUAGES

SUTEE SUDPRASERT

A Thesis Submitted in Partial Fulfillment of
the Requirements for the Degree of
Doctor of Engineering (Computer Engineering)
Graduate School, Kasetsart University
2010

Sutee Sudprasert 2010: Design and Development of a Lattice Structure Dependency Parser for Under-Resourced Languages. Doctor of Engineering (Computer Engineering), Major Field: Computer Engineering, Department of Computer Engineering. Thesis Advisor: Associate Professor Asanee Kawtrakul, D.Eng. 80 pages.

Dependency representations have become fashionable again in various Natural Language Processing areas, such as Machine Translation, Information Extraction, Text Summarization, and Ontology. However, the development of a good dependency parser requires some resources such as training corpora or grammar rules and also morphosyntactic analysis tools as preprocessing tools. Unfortunately, many languages do not have a large training corpus nor reliable morphosyntactic analysis tools.

We present here a corpus-based approach for building a dependency parser especially for under-resourced languages. Dealing with unreliable morphosyntactic analysis tools, we propose a methodology for dependency parsing outputting all possible of morphosyntactic analysis by modifying the Eisner's algorithm ($O(n^3)$), a bottom-up dynamic programming chart parsing algorithm, that does not increase the time complexity order. For computing the parse score, we use Maximum Entropy Models and train the model with a small training corpus (716 sentences). Because the training corpus is very small, we also propose a method for adjusting the parse score by using a Dependency Insertion Grammar (DIG) induced from the corpus. The adjustment will be applied if invalid trees are produced by the statistical model. Moreover, the use of DIG can make it easier for us to observe language behavior and detect annotation errors through the induced DIG rather than looking into the corpus directly. We tested the system by using NAiST Thai Dependency Treebank as training data and the accuracy of the parsing results was 80% if sentences were word-segmented correctly and 85% if the sentences were also part-of-speech tagged.

_____ _____ __ / __ / __

Student's signature Thesis Advisor's signature

ACKNOWLEDGEMENTS

I would like to grateful thank to Assoc. Prof. Dr. Asanee Kawtrakul, my thesis adviser for advice, encouragement, and valuable comments for my thesis. She was the first one who introduced me to the field of natural language processing and taught me a lot about research methodology. I also would like to thank Prof. Dr. Christian Boitet and Dr. Vincent Berment, my French co-advisers from Franco-Thai NLP project, for their valuable comments, suggestion, and patiently correcting my English.

Finally, I would like to thank to all members of the NAiST Laboratory.

Sutee Sudprasert

March 2010

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF ABBREVIATIONS

MT	=	Machine Translation
CFG	=	context-free grammars
DIG	=	Dependency Insertion Grammar
DG	=	Dependency Grammar
DP	=	Dependency Parsing
CDG	=	Constraint Dependency Grammar
WCDG	=	Weighted Constraint Dependency Grammar
TAG	=	Adjoining Grammar
CCG	=	Combinatory Categorial Grammar
CSP	=	Constraint Satisfaction Problem
POS	=	part-of-speech
LM	=	language model

DESIGN AND DEVELOPMENT OF A LATTICE STRUCTURE DEPENDENCY PARSER FOR UNDER-RESOURCED LANGUAGES

INTRODUCTION

Projective and non-projective dependency structures have recently become quite fashionable again in various NLP areas, such as MT (Ding and Palmer, 2005), Information Extraction (Yakushiji *et al.*, 2005), Text Summarization (Gagnon and Sylva, 2005), and Ontology (Kim and Park, 2004). In these applications, an input text will be syntactically analyzed into a dependency structure that can reduce forms of ambiguity in the input text.

Figure 1 shows an example of MT using dependency structure as intermediate structure. An input sentence will be analyzed into dependency structure first, then the system transforms that structure into a dependency structure of a target language and generates a translation result. Note that the meaning of part-of-speech notions are presented in Appendix Part-of-speech.

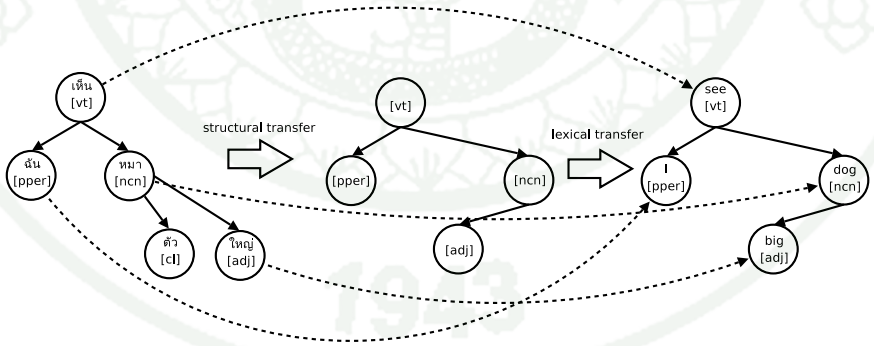
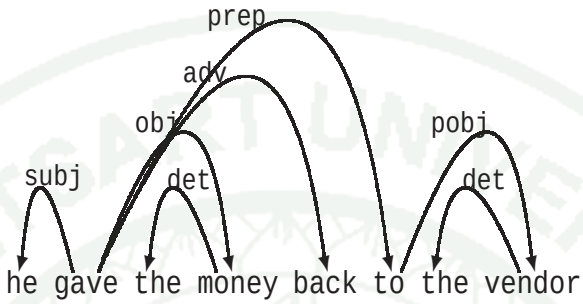


Figure 1 An example of machine translation using dependency structure as intermediate structure

By using the parser, the translation process will be easier than without it. There are several reasons for that (Boitet and Zaharin, 1988):

- they are more economical (they have less nodes) and hence perspicuous than constituent structures;

• they can represent some discontinuous constituents in a projective way. For example, in “he gave the money back to the vendor”, *gave… back* is a discontinuous constituent which cannot be represented by a constituent tree having a projective correspondence with the sentence. However, the following dependency tree is projective:



That is not always possible. For example, “Ces femmes, les hommes *n*e **l**es ont *p*as encore **t**ous comprises.” (“These women, the men did not yet understand them$_{women}$ all$_{men}$.”) has no reasonable projective dependency tree.

• they represent long-distance dependencies and predicate-argument relations (information needed in these applications) in a clearer way.

Figure 2 shows an example of discontinuous constituents represented by a dependency structure. In this case, there is no way to draw the correspondence lines between the nodes of the dependency tree and the words of the sentence (written as usual linearly) without any crossing pair of lines. The expression “tree without crossing lines” if often found in the literature but is faulty, as a tree can *always* be drawn on a plane without any crossing branches. The thing which is projective or not is the *correspondence* between the string and the tree, which should better be represented by “liaison elements”, as in entity-relation diagrams. We then say that the tree is “non-projective”.

Dependency representations date back to Tesnière (1959) and have been used extensively in NLP by Western and Eastern European and Japanese research groups since the early 1960’s, notably for Machine Translation (MT). Constituent or “phrase-based” representations have also been used, primarily for other applications such as Natural Language (NL-)based information retrieval, mainly because of their

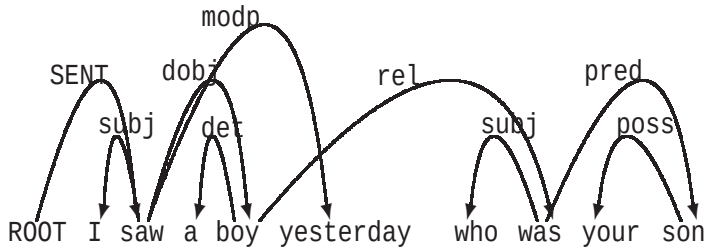


Figure 2 Example of discontinuous constituents: the noun phrase “a boy who was your son” and the verb phrase “saw yesterday” are “shuffled”.

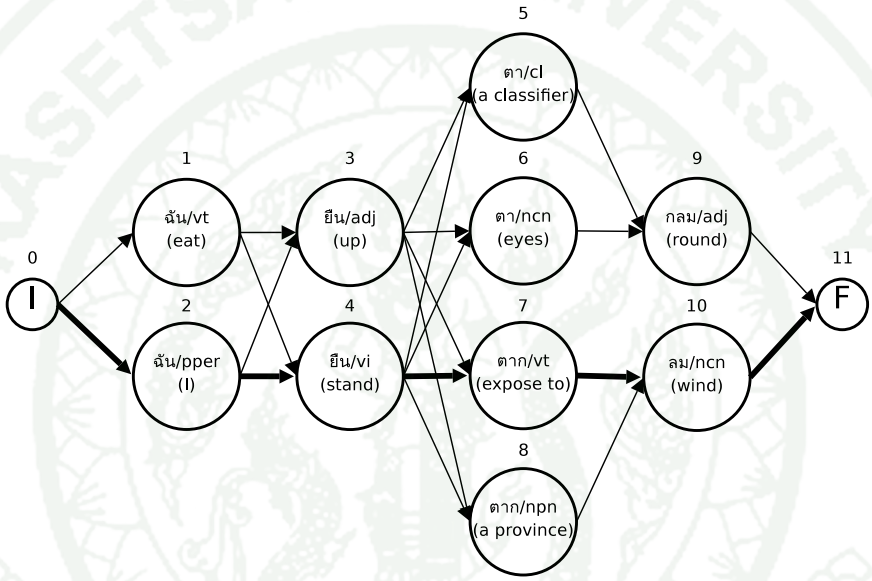


Figure 3 A morphosyntactic lattice that encodes all possible word segmentations and part-of-speech tagging results of a Thai text that means “I stand (to) expose to (the) air” (the bold lines represent the correct result).

good formal characterization by context-free grammars (CFG), and the existence of polynomial all-path algorithms (notably the CYK algorithm, for CFGs in Chomsky normal form, and Earley’s algorithm, for any CFG).

Written Thai, the language worked on this research, does not have word delimiters, and there are no reliable word segmenters and part-of-speech taggers for this language. Therefore, syntactic analysis of Thai should start from multiple results of a morphosyntactic lattice (see Figure 3), rather than from a dubiously disambiguated string (a dubiously disambiguated string means a string disambiguated by an unreliable morphosyntactic analyzer). Previous work is generally based on the assumption that

the input is a disambiguated morphosyntactic string. Unfortunately, this is not very realistic for a language like Thai with its potential ambiguities due to multiple word segmentation and part of speech tagging. This is why an extended parsing technique was proposed, able to handle as input a morphosyntactic lattice, with multiple ways of segmentation and part-of-speech tagging.

The research is presented here in order to deals with the above-mentioned problems. A possible solution for parsing a morphosyntactic lattice is proposed. To deal with the under-resourced situation, A combination of a "statistical" and an "expert" approach is invented in order to increase the parser accuracy by using a Dependency Insertion Grammar (DIG) (Ding and Palmer, 2004) which is automatically extracted from a tree-annotated corpus. A method for rescoring sub-trees with a language model is also proposed. This method is necessary when the Eisner's algorithm is applied to parse a morphosyntactic lattice. Furthermore, those extensions do not increase the time complexity order, it is still $O(n^3)$. The time complexity of k-best parsing is only increased by a multiplicative factor, $O(k \log k)$.

The contribution of this research are

1. a new dependency parsing algorithm for word lattices (Sudprasert *et al.*, 2009),

2. a Thai dependency treebank and its annotation guidelines (`http://naist.cpe.ku.ac.th/tred/`),

3. and treebank manipulating tools: Tree finder and DIG viewer (`http://naist.cpe.ku.ac.th/tred/digviewer/`)

OBJECTIVES

1. Develop an algorithm for dependency parsing an input lattice structure.
2. Improve of the quality of parsing output by automatically induced grammars.
3. Learn a DIG grammar from a very small tree-annotated corpus.
4. Learn a human-readable grammars that can be understood by linguists.

LITERATURE REVIEW

In this section, the background concerning Dependency Grammar (DG) and Dependency Parsing (DP) that are necessary for understanding the rest of the thesis is presented. For more detail about the state of the art in dependency-based parsing, please see Nivre (2005).

1. Dependency Grammar

The staring of the modern theoretical tradition of DG has often been referred to the work of Tesnière (1959). However, Covington (1984) argued that DG has been used since the Middle Ages. Western and Eastern European and Japanese research groups has used DG extensively in NLP since the early 1960's, notably for Machine Translation (MT).

The intuition behind the DG is simple that: in one sentence, all words depend on other words, except a unique word that does not depend on any other word, and is called the root of the sentence. An example of a simple DG analysis of the sentence "A boy runs' is shown below. Its dependency structure is demonstrated in Figure 4.

```
a depends on boy
boy depends on runs
run depends on nothing (root of the sentence)
```

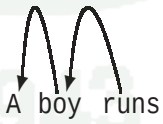


Figure 4 Example: dependency structure of the sentence "A boy runs".

The difference between constituency and dependency representations is the lack of phrasal nodes in the constituency representations. This can be seen by comparing the constituency representation of an English sentence in Figure 5, to the corresponding dependency representation in Figure 6.

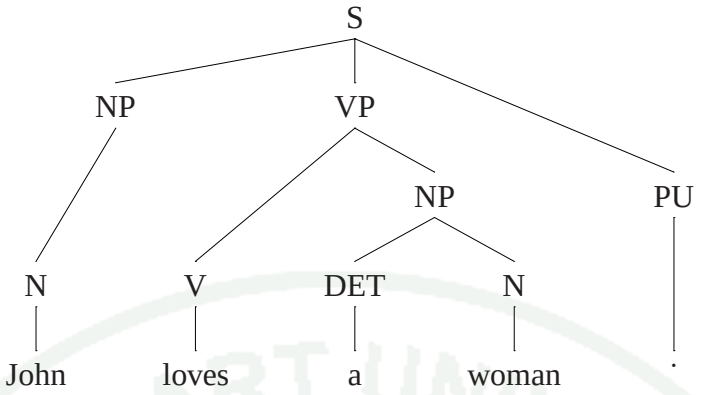


Figure 5 Constituent structure of English sentence “John loves a woman”

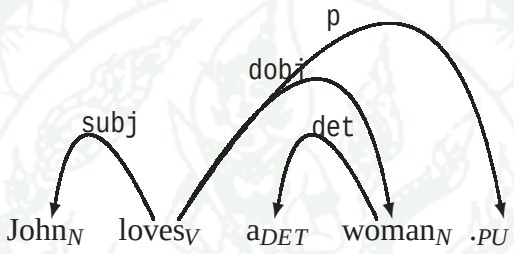


Figure 6 Dependency structure of English sentence “John loves a woman”

Normally, relations of dependencies are categorized into two types: complements and modifiers (adjunct). The modifiers are optional dependents that can be removed without disrupting the syntactic structure and the complements are obligatory dependents.

After the work of Tesnière there was a number of studies on dependency representations and various types of DG were proposed. The well known are Word Grammar (Hudson, 1990) and Meaning Text Theory (Mel’čuk, 1988) proposed a variety of computable syntactic dependency formalisms. Functional Dependency Grammar (Tapanainen and Järvinen, 1997) follows Tesnière’s model by distinguishing between dependency rules and rules for surface linearization. It adopts Tesnières notion of nuclei that are the primitive elements of FDG structures. Some researchers (Karlsson, 1990; Maruyama, 1990) were interested in treating the parsing of dependency structures as a constraint-based satisfaction problem that led to Constraint Dependency Grammar (CDG) and its descendent Weighted Constraint Dependency Grammar (WCDG).

Another important independency representation is Functional Generative Description (Sgall *et al.*, 1986), Based on the assumption of a language-independent underlying order, FDG represents a projective dependency tree and maps via ordering rules to the concrete surface realization. It is the core theoretical foundation of the Prague Dependency Treebank (Hajič *et al.*, 2000). Finally, there are some works (Joshi and Rambow, 2003; Clark *et al.*, 2001) that tried to produce dependency structures from mildly context-sensitive formalisms such as Adjoining Grammar (TAG) and Combinatory Categorial Grammar (CCG), which represent semantic dependencies and also handle non-projective structure naturally. However, those approaches were not often used for syntactic parsing because inheritance of mildly context-sensitive formalism leads to higher computing time. In worst case, the time complexity is $O(n^6)$ (Vijay-Shankar and Joshi, 1985). DIG also was inspired by TAG coming up with only one operation for derivation that is not a kind of adjoining operation. This makes the DIG simple to understand and easy to design an effective parsing algorithm. The details of DIG will be explained in MATERIALS AND METHODS chapter.

2. Dependency Parsing

2.1 Grammar-driven parsing

2.1.1 Context-free dependency grammars

In the earliest work on parsing with dependency representations Hays (1964) and Gaifman (1965), a dependency grammar was formulated in a way similar to the way CFGs were defined. The formulation composses of three rules: 1) determining dependents and their positions (left or right) of a given category, 2) giving every category belonging to a given word and 3) selecting a governor of a sentence from given list of all categories. This formulation can be transformed to lexicalized context-free grammar (as shown below) and is possible to be parsed with the standard context-free parsing algorithms (CKY, Earley, etc).

$$H \rightarrow L_1 \ldots L_m \; h \; R_1 \ldots Rn$$

$$H \in V_N; \quad h \in V_T; \quad L_1 ... L_m, R_1 ... R_n \in V_N^*.$$

where V_N and V_T are non-terminal and terminal vocabularies.

Such formulations are restricted to the derivation of projective dependency structures. There are frameworks that allow post-processing and introduce non-projective structures such as Sleator and Temperly (1991). They proposed a method to solve crossing links which occur because of coordinating conjunctions.

Many of these frameworks can be included to the notion of *bilexical grammars* of Eisner (2000) which consists of three elements:

1. A set V of words, called the *vocabulary* containing a distinguished symbol *ROOT*,

2. A set M of one or more *modifier roles* and

3. A pair of deterministic finite-state automata l_w and r_w that accepts some set of string over the alphabet $V \times M$. The l_w specifies the possible sequences of left dependents for w. The r_w specifies the possible sequences of right dependents for w.

For example, the bilexical grammars that recognize the dependency tree shown in Figure 6 can be written as

$$V = \{\mathtt{v}_1 = \mathtt{loves}_\mathtt{V}, \mathtt{v}_2 = \mathtt{woman}_\mathtt{N}\}$$

$$M = \{\mathtt{m}_1 = (\mathtt{John}_\mathtt{N}, \mathtt{subj}), \mathtt{m}_2 = (\mathtt{woman}_\mathtt{N}, \mathtt{dobj}), \mathtt{m}_3 = (\mathtt{a}_\mathtt{DET}, \mathtt{det}), \mathtt{m}_4 = (.\mathtt{PU}, \mathtt{p})\}$$

l_{loves_V} is an automata that accepts $\{(\mathtt{v}_1, \mathtt{m}_1)\}$

r_{loves_V} is an automata that accepts $\{(\mathtt{v}_1, \mathtt{m}_2), (\mathtt{v}_1, \mathtt{m}_4)\}$

l_{woman_N} is an automata that accepts $\{(\mathtt{v}_2, \mathtt{m}_3)\}$

The general parsing algorithm proposed by Eisner for bilexical

grammars is a bottom-up dynamic parsing algorithm. It proceeds by linking *spans* or sub-trees (where roots occur either leftmost or rightmost) instead of *constituents* (see more details in MATERIALS AND METHODS section 3.1.1). Therefore, the time complexity reduces from $O(n^5)$ to $O(n^3)$. Moreover, Eisner also shows how the framework of bilexical grammar, and the cubic-time parsing algorithm, can be modified to capture a number of different parsing frameworks and approaches. The previous are those of 1) Milward (1994) who proposed Dynamic Dependency Grammar and proved that the grammar can be recognized in $O(n^3)$, 2) Alshawi (1996) who used head automata (costed bidirectional finite state automata associated with the head of words of phrases) for representing a language model of a machine translation, and 3) Sleator and Temperly (1991) who developed Link Grammar.

2.1.2 Constraint dependency grammar

In the Constraint dependency grammar, the dependency parsing is characterized as a Constraint Satisfaction Problem (CSP). As usual, parsing algorithm must fight two fundamental forms of ambiguity: the lexical ambiguity (the ambiguity of an individual word or phrase used in different contexts to express two or more different meanings) and the structural ambiguity (the ambiguity of a phrase or sentence that there are more than one underlying structure). It is convincingly demonstrated that constraint-based techniques can effectively handle such ambiguities. This approach is also called “eliminative parsing”. Since sentences are analyzed by the successive eliminating representations, that violate constraints until the only valid representations remain. For example, suppose there are role constraints for *subject* and *adjective*.

Subject: the subject of a finite verb must be either a noun or a pronoun, it must agree with verb, and must have nominative case:

$$\begin{aligned}\Gamma_{subject}(w,w') \;\equiv\; & \mathbf{cat}(w') \in \{\mathtt{n},\mathtt{pro}\} \\ & \wedge\, \mathbf{arg}(w) = \mathbf{arg}(w') \\ & \wedge\, \mathbf{arg}(w') \in \mathtt{NOM}\end{aligned}$$

Adjective: an adjective may modify a noun and must agree with it:

$$\begin{aligned}\Gamma_{adjective}(w,w') \equiv\ & \mathbf{cat}(w) = \mathtt{n} \\ & \wedge \mathbf{cat}(w') = \mathtt{adj} \\ & \wedge \mathbf{arg}(w) = \mathbf{arg}(w')\end{aligned}$$

By following the these constraints, dependency trees with the structures conflicting to them will be eliminated. In practice, there are many constraints needed for building a dependency parsing based on CSP such as lexical constraints, valency constraints , role constraints, treeness constraints, etc (Duchier, 2000).

Maruyama (1990) was the first one who proposed a complete treatment of dependency parsing as a CSP and described parsing as a process of incremental disambiguation by generalizing the notion of tag to pairs consisting of a syntactic label and an identifier of the head node. This kind of representation is the fundamental of many different approaches to dependency parsing. Because it provides a way to reduce the parsing problem to a tagging or classification problem. Harper (1995) continued this line of research and proposed several algorithmic improvements. The FDG system (Tapanainen and Järvinen, 1997) is also classified as the approach of (Maruyama, 1990). It plus combines the eliminative parsing with a non-projective dependency grammar inspired by Tesnière (1959).

CSPs in general are NP-complete. It means relaxation have to be allowed to ensure reasonable computing time. In Maruyama (1990) and Harper and Helzerman (1995), the worst case of polynomial complexity is obtained only by considering local information in the application of constraints.

Parsing as solving a CSP is a rule-based approach. Therefore, it is impossible to avoid *under-generation* (no analysis satisfying all constraints) and *over-generation* (more than one satisfied analysis) situations. Heinecke *et al.* (1998) extends the CDG framework of Maruyama (1990) with graded (weighted) constraints, by assigning a score w ($0.0 \leq w \leq 1.0$) to each constraint, indicating the seriousness

of the violation of that constraint. The constraints are categorized into three types: hard constraints (score $= 0$), typical well-formedness conditions (score ≈ 0), weak constraints (score ≈ 1).

2.2 Data-driven parsing

2.2.1 Graph-based models

In graph-based models, the parsing models (Eisner, 1996; McDonald, 2006) conceptually consist of two parts: the permissible enumeration of all dependency trees according to a grammar or a constraint, and the selection of the most probable analysis according to a statistical model learned from a training corpus (see Figure 7 for an example).

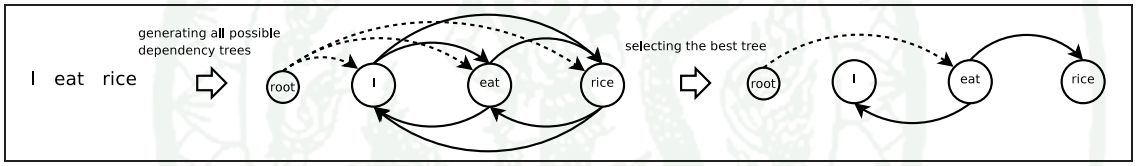


Figure 7 A concept of dependency parsing based on the graph-based models

An influential approach developed by Eisner (1996), are the threes probabilistic models (bigram lexical affinities, selectional preferences, and recursive generation) for dependency parsing that are evaluated by using supervised learning with data from the Penn Treebank. In the work of McDonald (2006), he applied discriminative estimation methods by using an online learning algorithm named MIRA (Crammer *et al.*, 2006), instead of using generative methods of Eisner (1996), to probabilistic dependency parsing. Moreover, he applied the Maximum Spanning Trees (MST) algorithm to find the best tree. McDonald's methods can handle both projective trees and non-projective trees (Edmonds, 1967; Chu and Liu, 1965).

Collins *et al.* (1999) applied the generative probabilistic parsing models of Collins (Collins, 1997 1999) to dependency parsing by using data from the Prague Dependency Treebank. This requires preprocessing to transform dependency structures into flat phrase structures and post-processing to extract dependency

structures from the phrase structures produced by the parser.

Another probabilistic approach of dependency parsing, that incorporates labeled dependencies, is extending an existing grammar-based model with a generative probabilistic model. In this approach, the grammar can be seen as a supertagging representing constraints on possible heads and dependents. Then the actual dependency links is determined from the supertag assignment. Examples of this approach are the CDG parser (Wang and Harper, 2004), the LTAG parser (Shen and Joshi, 2005), and the CCG parser (Clark and Curran, 2004).

2.2.2 Transition-based models

The approach is based on purely discriminative models of inductive learning in combination with a deterministic parsing strategy. The parsing model may define (McDonald and Nivre, 2007)

1. a set C of *parser configurations*, each of which defines a (partially built) dependency graph G

2. a set T of *transitions*, each a function $t : C \rightarrow C$

3. for every sentence $x = w_0, w_1, ..., w_n$,

    1. a unique *initial* configuration c_x

    2. a set C_x of *terminal* configurations.

A transition sequence $C_{x,m} = (c_x, c_1, ..., c_m)$ for a sentence x is a sequence of configurations such that $c_m \in C_x$, and for every $c_i (c_i \neq c_x)$, there is a transition $t \in T$ such that $c_i = t(c_{i-1})$. The dependency graph assigned to x by $C_{x,m}$ is the graph G_m defined by the terminal configuration c_m.

The deterministic discriminative approach was first proposed by Yamada and Matsumoto (2003). They used Support Vector Machines (SVM) to train

classifiers to predict the next action of a deterministic parser which is implemented in a form of shift-reduce parsing with three possible actions (*Shift*, *Right*, and *Left*). The parser processes the input from left to right repeatedly until there is no more dependencies are left. It means that up to $n-1$ passes over the input may be required to construct a complete dependency tree, giving the worst case time complexity of $O(n^2)$ (the worst case seldom occurs in practice).

Nivre *et al.* (2004) proposed a dependency parsing framework similar to Yamada and Matsumoto (2003), but the system can construct labeled dependency representations. Moreover, it can construct a complete dependency tree in a single pass over the data. They use memory-based learning to induce classifiers to predict the next parsing action based on conditional features.

3. Related Works

Our parsing algorithm falls into the data-driven graph-based parsing approach. Since, the algorithm used for selecting the best tree is based on Eisner (1996), a generative model with a cubic time parsing algorithm that based on a graph factorization. In this research, the Eisner's algorithm is modified in order to handle a lattice input and used DIG as a constraint for limiting the enumeration of all analyses.

McDonald (2006) and Jinshan *et al.* (2004) also developed their parsers based on Eisner's algorithm. McDonald focused on non-projective parsing with a discriminative learning algorithm, MIRA (Margin Infused Relaxed Algorithm). The accuracy of McDonald's parser is quite high i.e. 90% approximately for English, and was claimed to be state-of-the-art in this area. Jinshan *et al.* (2004) proposed a dependency parsing algorithm that is trainable with small data. The algorithm is similar to Eisner's for finding the best path from all the parsing results. Jinshan *et al.* (2004) trained the models by using only part-of-speech and head-dependent distance information.

As a pioneer, another kind of algorithm for data-driven parsing that makes decision in a shift-reduce parsing is the algorithms of Ratnaparkhi (1999). He first used Maximum Entropy Models to determine the action of a shift-reduce parser for

constituent structures. The accuracy of his parser was state-of-the-art in that time. Afterwards, Nivre and Scholz (2004) and Yamada and Matsumoto (2003) also applied the same idea to dependency parsing. The difference between those two works is that the former trained the model by using SVM while the latter used a memory-based learning.

McDonald and Nivre (2007) report that the average of accuracy of graph-based model and transition-based model for data-driven dependency parsing is similar. Hence, this research decided to start from Eisner's algorithm because of its dynamic programming character capable to extend in supporting multiple input and combining with grammatical rules.

Dependency grammars have a long history in the formal linguistics and computational linguistics communities. The first person who has been considered starting on modern dependency grammar is Tesnière (1959). After the work of Tesnière, there were many researches on dependency representations (grammars) and their relationships to other formalisms. The first researchers who studied the mathematical properties of DG were Hays (1964) and Gaifman (1965). Many dependency representations proposed after that were such as Functional Generative Description (Sgall *et al.*, 1986), Dependency Unification Grammar (Hellwig, 1986), Meaning Text Theory (Mel'čuk, 1988), Word Grammar (Hudson, 1990), Functional Dependency Grammar (Tapanainen and Järvinen, 1997), Dependency Insertion Grammar (Ding and Palmer, 2004), etc.

Now, many types of dependency representations were proposed with in less different abilities. Their generative power usually in the range between context-free to mildly context-sensitive (Joshi and Schabes, 1997). In this research, DIG was used, although it is not the most powerful dependency grammar. However, it is a simple one with a few components (2 types of elementary trees and 1 type of operation). Because of its minimalist, it is easy to integrate the DIG with statistical parsing models and it is also easy to validate them for the linguists who maintain the grammar.

In recent years, various researchers have started to build parsers with small

annotated corpora or without corpora. For example, Jinshan *et al.* (2004) proposed a method to build a dependency parser for Chinese. They used a purely statistical approach and relied only on part-of-speech information. Hwa *et al.* (2005) proposed a method for building a parser based on a parallel corpus, by inducing parse trees on the basis of parallel text. To do so, they need a parallel corpus and a reliable parser of the source language. However, all of them assumed that the morphosyntactic analysis work properly.

Previous works are generally based on the assumption that the input is a disambiguated morphosyntactic string. Unfortunately, this is not very realistic for a language like Thai because of its potential ambiguities due to word segmentation and part of speech tagging. This is why an extended parsing technique is proposed to handle an input as a morphosyntactic lattice, with multiple ways of segmentation and part-of-speech tagging.

The idea of parsing multiple versions of a given sentence has been proposed before by Tomita (1986). he proposed an efficient word lattice parsing algorithm that can be viewed as an extended LR parsing algorithm for context-free phrase structure grammars. Dependency parsing of word graphs has also already been proposed by Harper *et al.* (1993). The parser was an extended version of the CDG parser developed by Maruyama (1990). It actually was used to eliminate multiple sentence hypotheses produced by the speech recognizer. The complexity of the parser is $O(n^4)$, where binary constraints were used and the complexity will increase with the number of constraints. In a recent work, Collins *et al.* (2004) extended head-driven parsing models of Collins (Collins, 1999) to parse word lattices, in order to use simultaneously a language model and a parser for large-vocabulary speed recognition. The system experimented on the Wall Street Journal treebank shows better accuracy than the standard *n*-gram language model.

All previous researches put efforts on word lattice parsing intended to improve the accuracy of continuous speech recognition. In this work, a different method, a data-driven parser based on Eisner's algorithm (Eisner, 1996) is presented. By using the invented algorithm, the time complexity of the parser is $O(n^3)$, the same as Eisner's.

Moreover, the proposed parser can be augmented with DIG (Ding and Palmer, 2004) for eliminating wrong trees. This method can improve the accuracy of the parser for under-resourced languages.

MATERIALS AND METHODS

Materials

1. Computer

The parsing algorithms are implemented by using the Python and C programming language. The experiments are tested under the following computer specification:

1. Intel Xeon 64bit (Quad Core) Dual processor 2.00 GHz.
2. RAM 8 GB
3. Hard Disk (RAID-3) 650 GB
4. OS Debian GNU/Linux lenny/sid

2. Data

There are 3 linguistic resources that are needed for our system : a morphosyntactic dictionary, a part-of-speech (POS) tagged corpus, and a dependency treebank.

2.1 Morphosyntactic dictionary

Morphosyntactic dictionary is a list of pairs of a word and all its possible POS. It is used for creating a morphosyntactic lattice by mapping a given sentence with the items in the dictionary, thereby using a dynamic backtracking algorithm (Thumkanon, 2001). It will generate all possible word segmentations and part-of-speech tagging results of the given sentence; then the multiple results are encoded into a lattice. The morphosyntactic dictionary used in this research contains 16,816 entries. An example of items in the morphosyntactic dictionary is shown in Figure 8.

```
กระจอก            vi  adv  ncn
กระจอกงอกง่อย     vi  adj
กระจอกเทศ         ncn
กระจองหง่อง       adv
กระจองอแง         adv
กระจัง            ncn
กระจัดกระจาย      vi  adv
กระจัดพลัดพราย    vi  adj
```

Figure 8 An example of items in the morphosyntactic dictionary

2.2 POS tagged corpus

The POS tagged corpus contains sentences that were word-segmented and POS-tagged correctly. It is used for training the language model (LM) used for rescoring sub-trees. The size of POS tagged corpus is 40,494 sentences containing 49 tags, 18,396 words, and 548,431 tokens. Figure 9 shows a snapshot of the POS tagged corpus.

```
บาง/adj พันธุ์/ncn ดอก/ncn เล็ก/vi แต่/conj บาง/adj พันธุ์/ncn ก็/conj มี/vt ดอก/ncn ใหญ่/vi
แต่ละ/adj ใบ/ncn จะ/prev ประกอบด้วย/vcs ส่วน/ncn ของ/prep ก้าน/ncn ใบ/ncn และ/conj ตัว/ncn ใบ/ncn
ทุก/adj ๆ/punc เช้า/ncn คนงาน/ncn ที่/prep สวน/ncn จะ/prev ทยอย/vi ออก/vi เก็บเกี่ยว/vt ดอกไม้/ncn
ทั้งนี้/conj เพราะ/conj ครอบครัว/ncn นี้/det มี/vt อาชีพ/ncn รับ/vt จัด/vt สวน/ncn อยู่/vpost แล้ว/vpost
แต่/conj ถ้า/conj เป็น/vcs สี/ncn ที่/prel แปลก/vi ๆ/punc ราคา/ncn ก็/conj สูง/vi นิดหนึ่ง/adv
```

Figure 9 An example of the POS tagged corpus

2.3 Dependency treebank

In this work, we use NAiST Dependency treebank. The treebank consists of about 816 tree-annotated sentences that were prepared by linguists. A snapshot of the NAiST dependency treebank is shown in Figure 10.

2.3.1 Sources and Characteristics

We have tried to collect sentences from texts in various domains and genres such as agricultural news, encyclopedia, and health care.

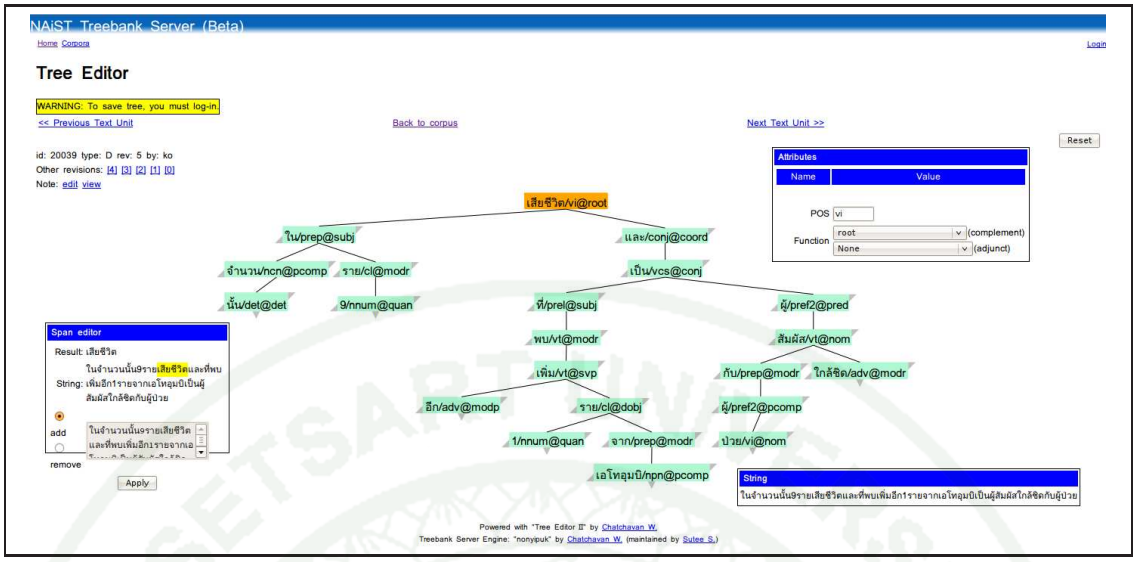


Figure 10 A snapshot of NAiST dependency treebank

2.3.2 Annotation Standard

Since our lab is pioneering the building of a Thai Dependency treebank, there was no presence annotation guidelines for this task. At first, we had to create the guidelines by adopting guidelines available for other languages such as Danish Treebank, Prague Treebank, and Penn Treebank (Even the Penn Treebank is phrase-based but it was annotated with grammatical functions which can be used to label the edges of dependency trees).

Having investigated the existing guidelines and compared them to Thai, we designed our annotation standard (`http://naist.cpe.ku.ac.th/tred/static/guideline/Thai_Dependency_Guideline_v.1.4.pdf`). It consists of 30 grammatical functions which are divided into two main types i.e. complement (12) and adjunct (18), which could represent all syntactical patterns of Thai writing. For more detail please see in the Appendix on Grammatical Functions.

We have annotated the sentences using Tred (`http://naist.cpe.ku.ac.th/tred` (Wacharamanotham *et al.*, 2007)), a web-based tree editing component which has a full set of facilities for manipulating the corpus such as graphical tree editor, revision control, permission control, and querying tools. For

internal use, the annotated trees are restored in JSON format (JSON is a lightweight computer data interchange format). It is more concise than XML, and also easy to use in Ajax web application programming which has been used for implementing our tools. For external interchange, the dependency trees are stored in the format which is used in MSTParser (`http://sourceforge.net/projects/mstparser/`). Each sentence is represented by 3 or 4 lines and sentences are separated by a new line.

The general format is:

$$\begin{array}{cccc} w_1 & w_2 & \ldots & w_n \\ p_1 & p_2 & \ldots & p_n \\ l_1 & l_2 & \ldots & l_n \\ d_1 & d_2 & \ldots & d_n \end{array}$$

where,

- w_1 ... w_n are the n words of the sentences (tab delimited).
- p_1 ... p_n are the part-of-speech tags for each word.
- l_1 ... l_n are the labels (grammatical functions) of the incoming edge to each word.
- d_1 ... d_n are integers representing the position of each words parent.

For example, the dependency tree of sentence “John loves a woman” shown in Figure 6 can be represented in

John	loves	a	woman
N	V	DET	N
subj	root	det	dobj
2	0	3	2

2.3.3 Tree editor

The tree editor is an important component of treebank preparing process. It is a web-based application that can be run on any operating systems via

web browser (`http://naist.cpe.ku.ac.th/tred/corpora/list`). However, the program can only be run on Firefox version 3, because of the use of the SVG (Scalable Vector Graphics) language for drawing the tree which is now supported only by Firefox.

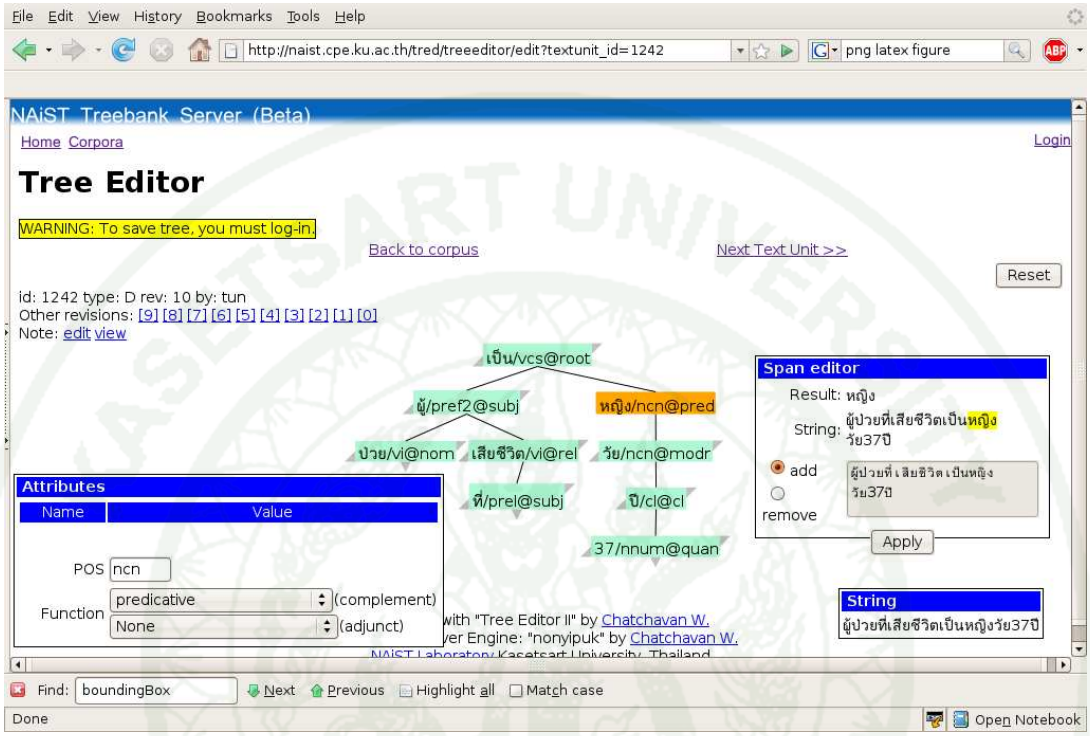


Figure 11 A screen shot of the tree editor tool

The tree editor (see figure 11) consists of 4 widgets: tree drawing, attributes editor, span editor, and string display. The tree drawing widget is the main part of the program which is used for manipulating the structure of the tree (only moving the nodes in the tree). Adding a new node or deleting some nodes in the tree is done by using the span editor. The attribute editor is used for editing the part-of-speech and grammatical function of each word. The last widget, string display, shows the flat string corresponding to the tree (by flattening the tree projectively).

All changes of the tree are saved with information proper to the editor such as history of revisions which is useful for collaborative work, when more than one linguist works on the same tree. Moreover, each linguist can put some questions or comments for annotating the tree, these notes will be used later as discussion topics.

2.3.4 Attributes search

Tred provides two interfaces for retrieving the trees: Attributes search and DIGs viewer. Attributes search is a simple interface: one enters included and excluded keywords i.e. surface words, parts-of-speech and grammatical functions, and it returns the trees matching the keywords. In figure 12 is a screen shot of the program.

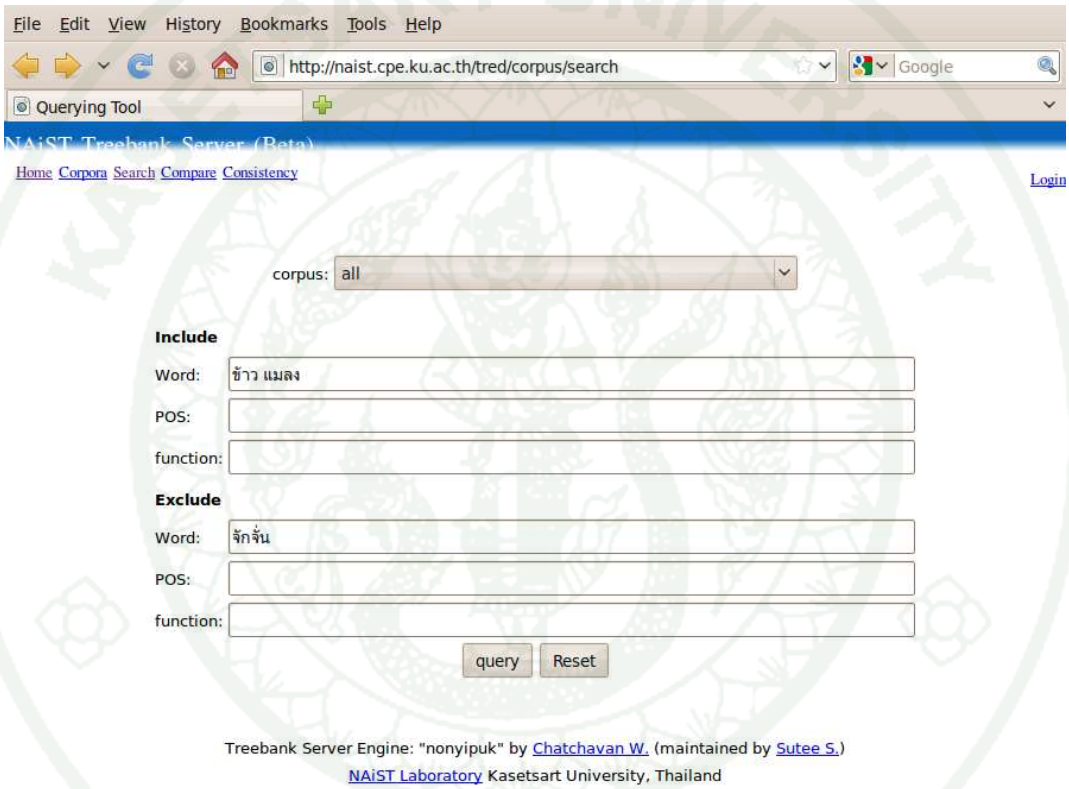


Figure 12 A screen shot of keywords searching interface.

2.3.5 DIGs viewer

The another querying tool is the DIG viewer tool (`http://naist.cpe.ku.ac.th/tred/digviewer/`). We can use this tool for retrieving the trees in the treebank which correspond to an elementary DIG tree. In other words, that is like using a elementary tree as a keyword for finding the trees.

At first, the DIG viewer shows all heads (word or word with its

part-of-speech) of the three special forms of elementary trees (as Figure 13). Users can see elementary trees by clicking on a head in the list, and then the elementary trees will be shown beside the clicked head (as Figure 14). If users click on a elementary tree, details such as local word ordering, original tree (the tree in treebank from which the elementary was extracted) , and link to the original tree, will be shown in another window (see Figure 15). The original tree is drawn with two colors green and yellow, to indicate complement and adjunct relations, respectively. The nodes colored blue in the original constitute the extracted elementary tree.

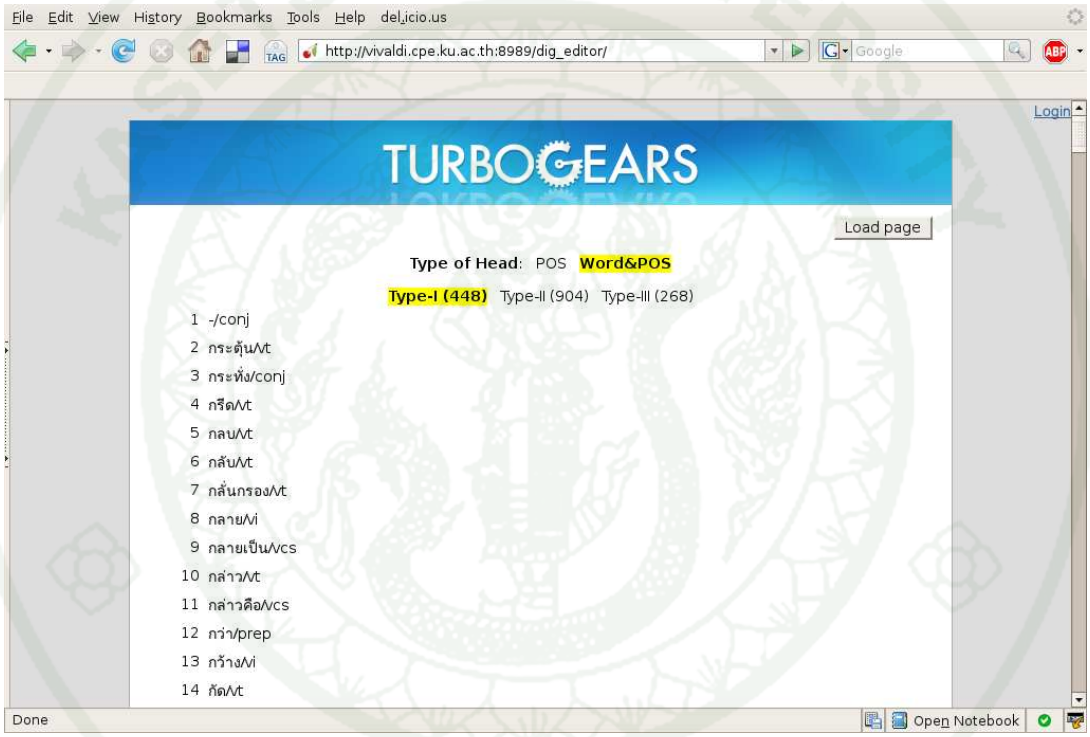


Figure 13 A screen shot of the DIG viewer (1).

The DIG viewer can be used not only for viewing all extracted elementary trees from the treebank, but also for detecting errors in the treebank, since the three special forms of the elementary were designed to be linguistically sound. If odd elementary trees appear, then the treebank must contain some errors. By using the DIG viewer, it is easy to find the errors and go to the original tree in the treebank in order to correct it. For example, the elementary trees for “vi” (intransitive verb) should not have object grammatical function (“pobj”, “dobj”, and “iobj”). If one of them appears, it means that the part-of-speech tagging is wrong (it should be “vt” transitive verb) or that the grammatical function annotation is incorrect.

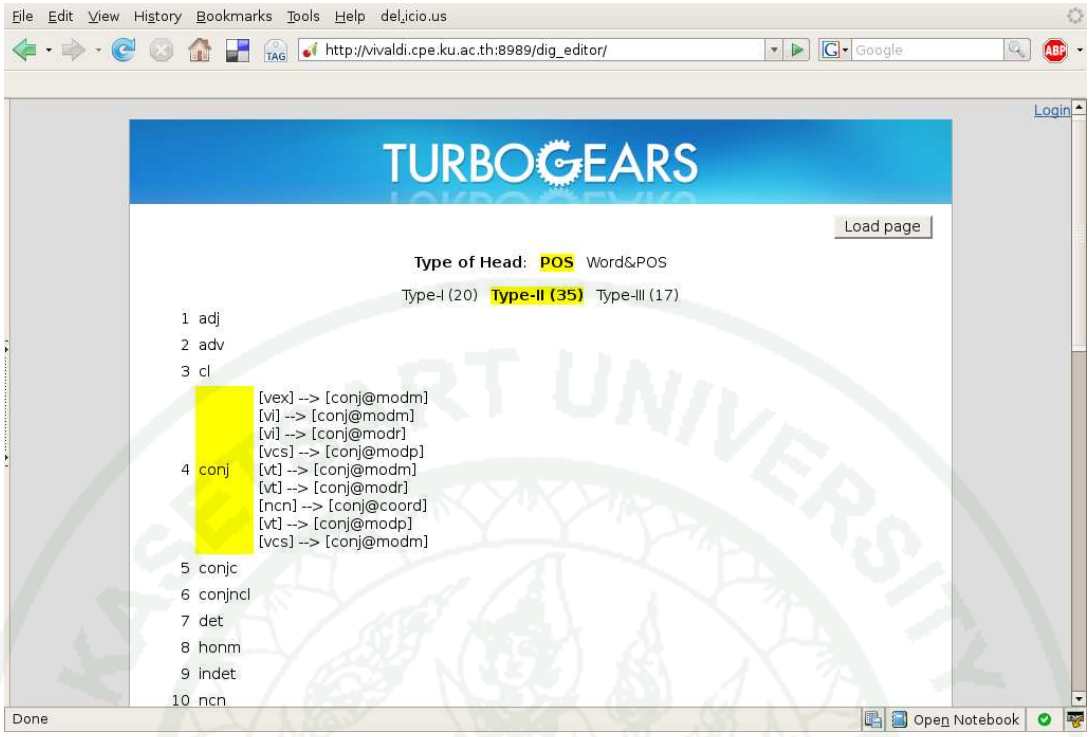


Figure 14 A screen shot of the DIG viewer (2).

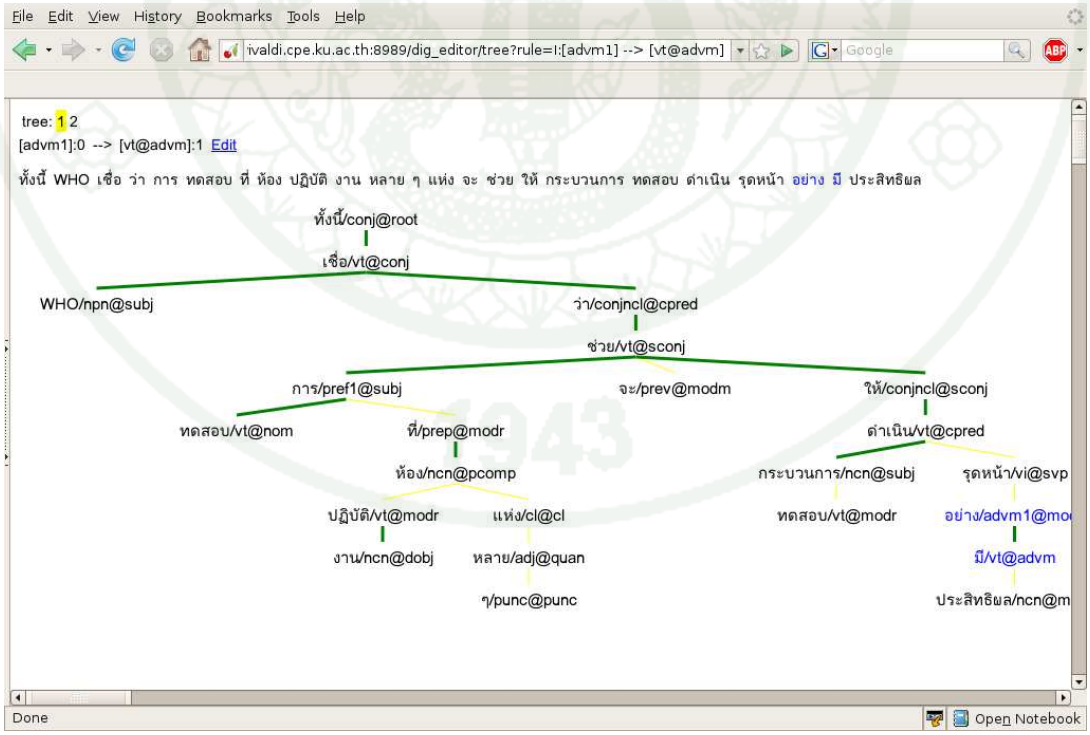


Figure 15 A screen shot of the DIG viewer (3).

Methods

1. System Overview

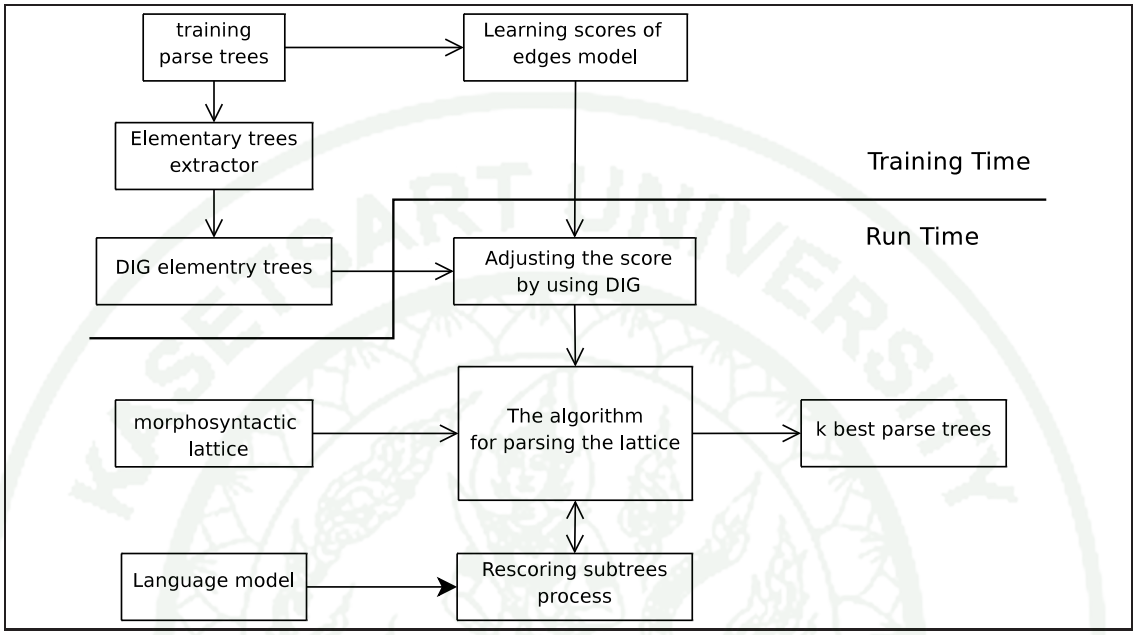


Figure 16 Training and run time of the parsing system.

The diagram in Figure 16 depicts graphically the training time (dependency scores learning and DIG elementary trees learning) and the run time (parsing a lattice input) of the parser described in this thesis. It shows the different steps that take place during training time, i.e. the learning of the elementary trees and dependency scores from a treebank, and run time, i.e. using the learned grammar and the learned scores to parse (*k*-best parse) a lattice input. More specifically, at training time, the elementary trees are extracted from a treebank and the dependency scores are learned by using a machine learning model that uses the treebank as training data. At run time, the dependency scores are adjusted by DIG and are used by the parsing algorithm to produce the *k* best parse trees. In addition, sub-trees produced in the parsing process are rescored by using LM, in order to promote the sub-trees which also have good morphosyntactic analysis.

In the following sections, we will describe the basis of DIG and each of the modules in the larger system.

2. Dependency Insertion Grammar

2.1 Basis of DIG

A grammar expressed in the DIG formalism (Ding and Palmer, 2004) consists of two parts: elementary trees and insertion operation. Each node in an elementary tree consists of: a lexical item, a corresponding part-of-speech and a local word ordering. The elementary trees are of two types: Type-A and Type-B. The difference between them (see Fig. 17) is that the root node of a Type-A tree is lexicalized and is the head of the tree, while the root of a Type-B tree is not lexicalized but one of the lexicalized nodes is the head of the tree.

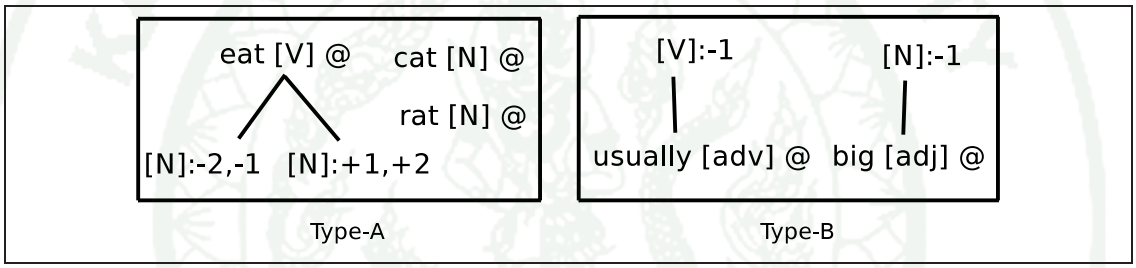


Figure 17 Type-A and Type-B elementary trees. Note: the @ symbol marks the head node of the tree.

Insertion is the only operation used for DIG derivation: when an unlexicalized node of an elementary tree is of the same type, i.e. category, as the head node of another elementary tree, the two can be unified into a single node and a new elementary tree can be built (see Fig. 18).

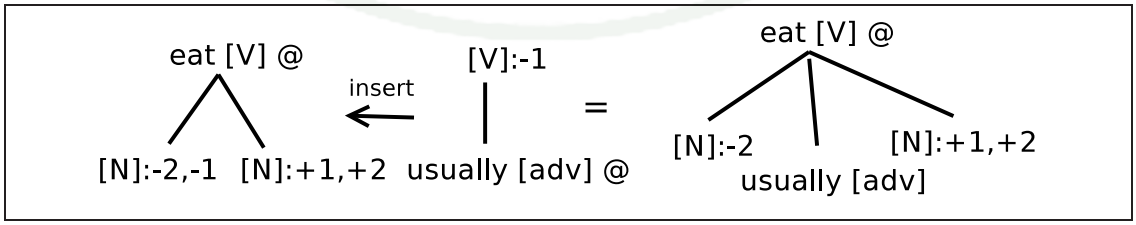


Figure 18 The insertion operation in DIG. -2,-1, +1, and +2 are relative positions.

2.2 The extended forms of elementary trees

The DIG formalism itself does not impose any constraints on the shape of the elementary trees, as long as they satisfy the requirements of Type-A and Type-B elementary trees. Therefore, given a corpus, there can be a number of elementary trees, each of which covers the corpus. To say that some elementary trees cover the corpus implies that each dependency tree in the corpus can be generated by combining the elementary trees by the insertion operation.

Like Xia (1999) who proposed a method for extracting LTAGs (Lexicalized Tree-Adjoining Grammars) from a bracketed corpus, we need to define extended forms of elementary trees in order to ensure that the extracted grammar is both compact and linguistically sound.

We consider extended elementary trees of the following forms (see Figure 19):

- Type-I tree: a type A tree with only complement functions (syntactic arguments, i.e. strongly bound complements).
- Type-II tree: a type B tree with only two nodes, the unique arc bearing an adjunct function (circumstantial complements).
- Type-III tree: a combination of Type-I and Type-II.

Given the fact that our tree-annotated corpus is very small, it is unavoidable to run into the data sparseness problem. In order to minimize this problem, we used relative direction instead of relative position i.e. all positions ≤ -1 be reduced to left (L) and position $\geq +1$ will be reduced to right (R). We call this the "relaxation of relative positions".

Figure 20 shows an elementary tree corresponding to "eat [VT]" (transitive verb). The left-hand side tree is a normal elementary tree of Type-I, waiting for "PPER"

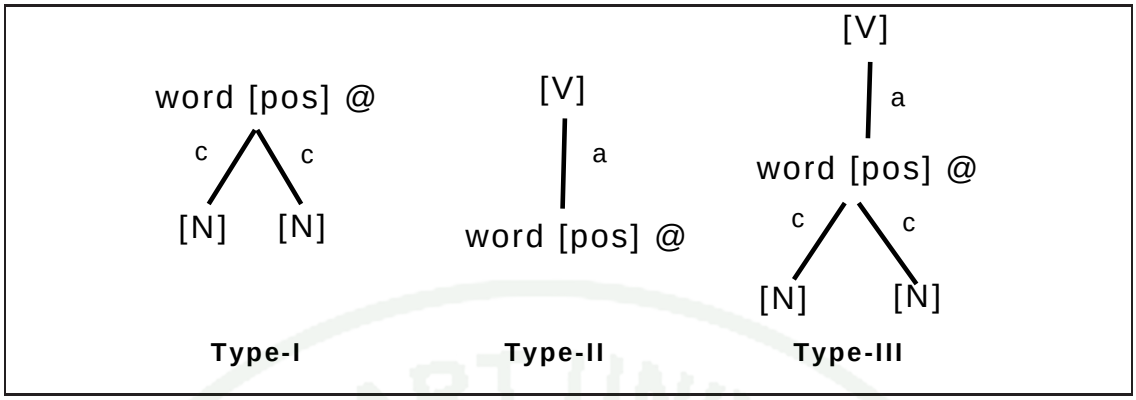


Figure 19 Three forms of elementary trees. Note: the“c” and “a” functions stand respectively for “complement” and “adjunct”.

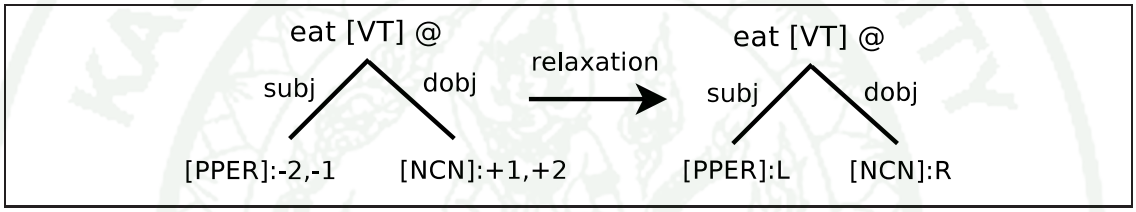


Figure 20 An example of the relaxation of DIG

(personal pronoun) at position -2 or -1 and “NCN” (common noun) at position +1 or + 2. When the constraints are relaxed, the elementary tree is changed to the right-hand side tree.

2.3 Extracting elementary trees from the treebank

The code of in Algorithm 1 is used for extracting elementary trees of the above three mentioned forms from the treebank. The algorithm starts from the root node and traverses each node in order to recognize patterns of the extended forms.

Given a training dependency tree, the algorithm traverses each node of the tree (line 2). At any visited node *x*, if the inward edge has a complement function (c) then a Type-I tree will be built where the visited node is the root and the part-of-speech of its children which have complement functions (line 3-6, 25). If an adjunct relation (a) on an outward edge is found, there are two possible cases. (1) If a child of a visited node *y* doesn’t have any children, a Type-II tree will be built immediately by using the

Algorithm 1 $ExtractDIG(T)$

Input: T a parse tree object
Output: a set of elementary trees

```
1  ElementaryTrees ← [ ]
2  for all x in T.nodes() do
3   T1 ← new_tree(x.word, x.pos)
4   for all y in x.children() do
5    if y.function = 'c' and x.function = 'c' then
6     T1.connect_at_root(null, y.pos)
7    else if y.function = 'a' then
8     if y.children() = [ ] then
9      T2 ← new_tree(null, x.pos)
10     T2.connect_at_root(y.word, y.pos)
11     ElementaryTrees.append(T2)
12    else
13     T3 ← new_tree(null, x.pos)
14     tmp ← new_tree(y.word, y.pos)
15     for all z in y.children() do
16      if z.label = 'c' then
17       tmp.connect_at_root(null, z.pos)
18      end if
19     end for
20     T3.connect_at_root(tmp)
21     ElementaryTrees.append(T3)
22    end if
23   end if
24  end for
25  ElementaryTrees.append(T1)
26 end for
27 return ElementaryTrees
```

part-of-speech of the visited node x as the root and its child node y as the child (line 8-11). (2) Otherwise, if a child of the visited node y has children, the algorithm will do like above and construct a Type-I tree in order to produce a sub-tree (line 14-19). A Type-III tree will be built by using the part-of-speech of the visited node x as the root, connecting the produced sub-tree tmp to the root. If the child of the visited node y does not have any children, a Type-II tree will be built instead (line 20-21).

Figure 21 shows an example of DIG elementary trees extracted from the annotated-tree text "I ate boiled rice with my friend".

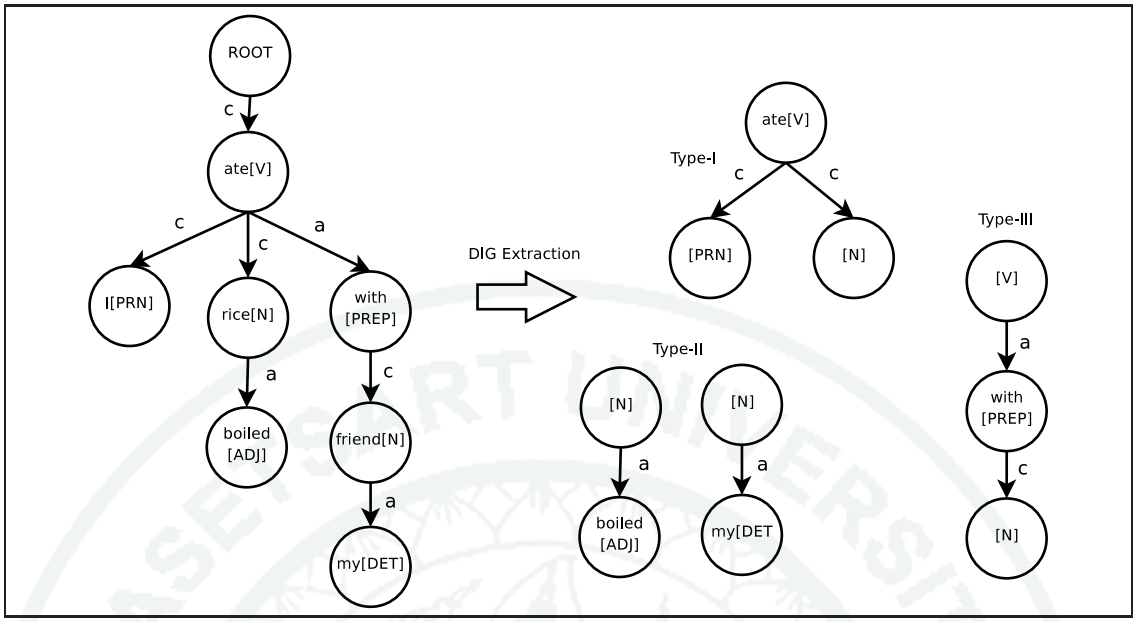


Figure 21 An example of extracting the extended forms of elementary trees from a parse tree.

2.4 Formatting of elementary trees file

It is necessary to mindfully design the format used for storing the elementary trees into a file because the format should not only be parsed by a machine, but should also read by humans. Storing elementary trees into a readable format will let the linguists observe the behavior of the language and also detect errors of annotation by identifying odd elementary trees which is easier than to observe the treebank directly.

A DIG file consists of two sections: elementary tree section and local word ordering section. Figure 22 shows some parts of the DIG file. The Elementary Section consists of elementary trees that are indexed by the heads (a head can only be a part-of-speech or a word with a part-of-speech) and type (I, II, or III). The float number which comes after the elementary trees indicates the conditional probability of the elementary tree given its head. As for the Word Ordering Section, the word ordering of nodes of the elementary trees are stored in this part. The list of numbers which comes after each the word ordering contains the reference IDs of the tree in the training corpus. It is used to get back to the original tree in order to analyze where corresponding the errors come from when some the elementary tree is incorrect.

```
### Elementary Tree Section ###
...
I:vi
   [vi] -> [pref1@subj],[conjncl@sconj]:0.004566
   [vi] -> [ncn@subj],[pref1@subj]:0.004566
...
II:adj
   [cl] -> [adj@modp]:0.027473
   [adj] -> [adj@modr]:0.013736
   [pref1] -> [adj@modr]:0.010989
...

### Word Ordering Section ###
...
[adj] -> [adj@modr]
   -1 -> 0:1.000000: #10549,10578,10685,10735
[adj] -> [adv@modp]
    1 -> 0:0.500000: #10450,10592
   -1 -> 0:0.500000: #10506,10626
```

Figure 22 An example of elementary trees file.

3. Dependency Parsing Techniques

The dependency parsing technique we use is based on the assumption that the score (probability) of a dependency tree is the sum of the scores of all edges in the tree, and that *a best parse tree* is one with *the highest score*. This approach consists of two processes: searching for a best tree among all possible trees and computing the scores of edges.

3.1 Searching the best tree

3.1.1 Eisner's algorithm

In the traditional approach, the input is a disambiguated morphosyntactic string $w_1 \ldots w_N$ where N is the number of words in the string. A $N \times N$ Dependency Matrix, DM, is built, where $DM[i, j]$ contains the best relation *rel* between

Algorithm 2 Eisner's algorithm

Input: DM a dependency matrix size $n \times n$; $n > 1$
Output: a score of the best parse tree

```
1  for all p,d,c in {1..n} × {⇐,⇒} × {0,1} do
2    C[p,p,d,c] ← 0.0
3  end for
4  for m = 1 to n do
5    for p = 1 to n − m do
6      t ← p + m
7      for r = p to t do
8        if r < t then
9          C[p,t,⇐,0] ← max(C[p,t,⇐,0],C[p,r,⇒,1] + C[r + 1,t,⇐,1] + DM[t,p]) /* create left incomplete items */
10         C[p,t,⇒,0] ← max(C[p,t,⇒,0],C[p,r,⇒,1] + C[r + 1,t,⇐,1] + DM[p,t]) /* create right incomplete items */
11         C[p,t,⇐,1] ← max(C[p,t,⇐,1],C[p,r,⇐,1] + C[r,t,⇐,0]) /* create left complete items */
12       else if r > p then
13         C[p,t,⇒,1] ← max(C[p,t,⇒,1],C[p,r,⇒,0] + C[r,t,⇒,1]) /* create right complete items */
14       end if
15     end for
16   end for
17 end for
18 return C[(1,n,⇒,1)]
```

w_i and w_j, with a score s.

$$DM[i,j] = \begin{cases} (rel, s) & \text{if } i \neq j \\ \text{empty} & \text{otherwise} \end{cases} \tag{1}$$

For example, if the input is "I love you" and the scores of dependencies are computed as following (Here, we do not take relation *rel* into account for simplicity)

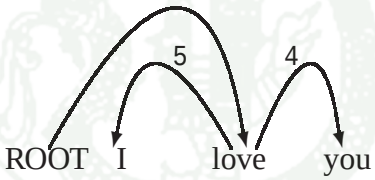
head		**dependent**		**score**
I	$\longrightarrow$	love	=	0
love	$\longrightarrow$	I	=	5
I	$\longrightarrow$	you	=	1
you	$\longrightarrow$	I	=	1
love	$\longrightarrow$	you	=	4
you	$\longrightarrow$	love	=	0

then, a DM of the sentence “I love you” is

		1	2	3
		I	love	you
1	I	0	5	1
2	love	0	0	0
3	you	1	4	0

where a row index represents position of a head and a column index represents postion of a dependent.

One may use a brute-force algorithm that generates all possible dependency trees and then selects the highest score tree. Here the highest score tree is



Unfortunately, the brute-force algorithm has exponential computing time and cannot be used for parsing a long length sentence in practice.

In fact, we can apply Eisner’s algorithm (Eisner, 1996) (See (McDonald, 2006; Eisner, 1996) for more details) to find the best projective dependency tree among all possible results in the dependency matrix within $O(n^3)$. Eisner’s algorithm is a bottom-up parsing algorithm just like the CKY parsing algorithm: it finds optimal subtrees for substrings of increasing length. The idea is to parse the left and right dependents of a word independently, and combine them later. That requires only two additional binary variables to specify the direction of the item and whether the item is complete, i.e., *d* and *c* which will be explained more in the following.

In the dynamic programming table C of Eisner's algorithm, $C[p,t,d,c]$ stores the score of the best subtree from position p to position t, with direction d and complete value c. The variable d indicates the direction of the subtree (whether it gathers left ($\Leftarrow$) or right ($\Rightarrow$) dependents). The variable c indicates whether a subtree is complete ($c = 1$, no more dependents) or not ($c = 0$, needs to be completed).

The pseudo-code of Eisner's algorithm is shown in Algorithm 2. The pseudo-code compute only the score of a best parse tree. We must also store back pointers so that it is possible to reconstruct a best tree from the chart item that spans the entire sentence. In this work, we assume that the output is an unlabeled dependency tree, therefore *DM* contains only scores.

Consider line 9 in Algorithm 2. It finds the best score for an incomplete left subtree from position p to t, $C[p,t,\Leftarrow,0]$. We need to find an index r ($p \leq r < t$) that gives the best (maximum) possible score for combining two complete subtrees, $C[p,r,\Rightarrow,1]$ and $C[r+1,t,\Leftarrow,1]$. The score of the tree obtained by combining these two complete trees is the score of these subtrees plus the score of the chosen dependency relation from v_i to v_j. This is guaranteed to be a score of the best subtree because we are considering all possible combinations by enumerating all values of r. By forcing the root node to be at the left-hand side of the sentence, the score of the best tree for the sentence is $C[1,n,\Rightarrow,1]$.

3.1.2 Extended parsing technique for handling a lattice structure

In order to extend this parsing technique for handling lattice structures, we use exactly the same Dependency Matrix as in (1), but we add one more condition to check whether there is a non-empty path from w_i to w_j in the lattice, which is $\mathrm{acc}(i,j)$ in (2). Algorithm 3 shows the adaptation of Eisner's algorithm to find a best projective dependency tree.

Algorithm 3 Modified Eisner's algorithm for handling a lattice input

Input: DM a dependency matrix size $n \times n$; $n > 1$
Output: a score of the best parse tree

```
1  for all p,d,c in {1..n} × {⇐,⇒} × {0,1} do
2    C[p,p,d,c] ← 0.0
3  end for
4  for m = 1 to n do
5    for p = 1 to n − m do
6      t ← p + m
7      for r = p to t do
8        for all q in get_next(r) do
9          if r < t and IsLegal(p,r,q,t) then
10           C[p,t,⇐,0] ← max(C[p,t,⇐,0], C[p,r,⇒,1] + C[q,t,⇐,1] + DM[t,p])
11           C[p,t,⇒,0] ← max(C[p,t,⇒,0], C[p,r,⇒,1] + C[q,t,⇐,1] + DM[p,t])
12         end if
13       end for
14       if r < t and IsLegal(p,r,r,t) then
15         C[p,t,⇐,1] ← max(C[p,t,⇐,1], C[p,r,⇐,1] + C[r,t,⇐,0])
16       end if
17       if r > p and IsLegal(p,r,r,t) then
18         C[p,t,⇒,1] ← max(C[p,t,⇒,1], C[p,r,⇒,0] + C[r,t,⇒,1])
19       end if
20     end for
21   end for
22 end for
23 return C[1,n,⇒,1]
```

The DM matrix will be (2)

$$DM[i,j] = \begin{cases} (rel, s) & \text{if } i \neq j \text{ and } \mathrm{acc}(i,j) \\ \text{empty} & \text{otherwise} \end{cases} \tag{2}$$

The modification to Eisner's original algorithm consists just in adding the condition *IsLegal* (line 9, 14 and 17 in Algorithm 3) and call to the function *get_next* (in line 8) to validate the built subtrees along the lattice structure. The call $get\_next(n)$ returns all next adjacent nodes of n, and

$$IsLegal(p,r,q,t) = acc(p,r) \wedge acc(q,t) \tag{3}$$

In order to illustrate this, let's take a look at Fig. 23, showing

the parsing processes respectively of Eisner's algorithm (Algorithm 2) and ours (Algorithm 3).

Figure 23a shows how to find the best incomplete sub-tree corresponding to the sub-string $p...t$ of Eisner's algorithm. The best incomplete sub-tree for sub-string $p...t$ is the combination of sub-trees $p...r$ and $r+1...t$ plus a dependency between w_p and w_t, which has received the highest score when $p \leq r < t$.

If the input is a morphosyntactic lattice, not every point of sub-trees can be combined to generate a new sub-tree. Only a combination which is in a *trajectory* can be generated. We call *trajectory* any sequence of directly linked vertices in a lattice beginning with I and ending with F. The p_{th} *trajectory* has the form $T_p = I \rightarrow w_{p1} \rightarrow ... \rightarrow w_{pl_p} \rightarrow F$. By looking at Figure 23b, we can see that the selectable sub-strings need to contain a link between a starting node and an ending node. For example, sub-string 1...2 cannot be selected, because there is no path from 1 to 2 (validated by *IsLegal* condition). Moreover, when we try to combine two sub-trees, we also have to check that there is an arc connecting the last node of the first sub-tree, r, to the first node of the second sub-tree, q (limited by *get_next* function).

The computing time is increased by testing the condition and looking for all nodes q directly connected from r. Both can be done in constant time equaling the branching factor (maximum number of outward edges of each node in the lattice). Hence we are still in $O(n^3)$ where n is the number of nodes in the lattice. Hence, there is no increase in time complexity.

3.2 Computing the scores of edges

In fact, a function for computing the scores of edges can easily be estimated by using machine learning models such as Maximum Entropy Models or SVM. But in our case where the training corpus is very small, the use of a machine learning model alone may lead the parser to produce invalid parse trees. In addition, we assumed that dependencies in a sentence are independent of each other, hence the score of a dependency tree is the sum of the scores of all its edges. However, relying

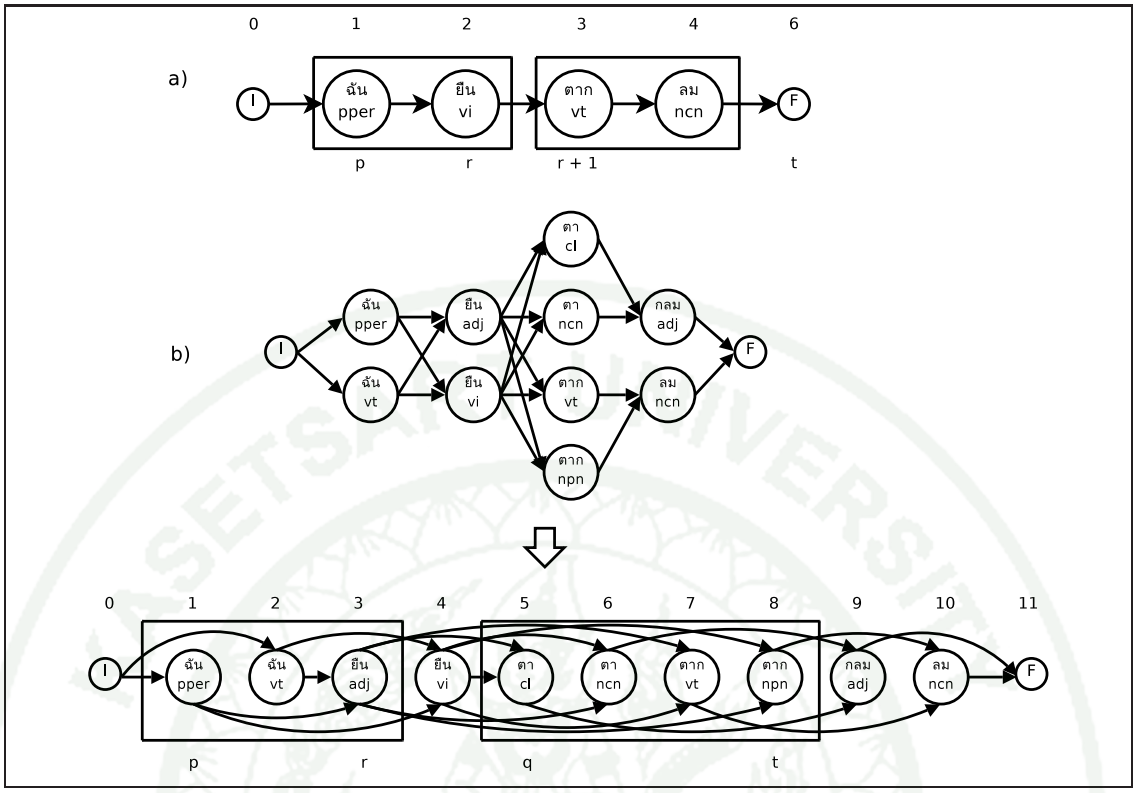
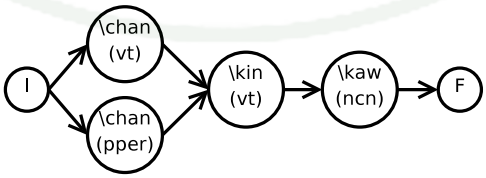


Figure 23 An example of parsing with a) a correct morphosyntactically analyzed text and b) a morphosyntactic lattice

solely on the score of edges is not appropriate to compute the score of dependency trees for each possible output from a morphosynatic analyzer (which are encoded in a lattice structure). Because it is possible that the parser will select a parse tree that has the highest score but the parse tree is not necessarily to be a correct morphosynatic analysis. See an example of parsing a simple sentence "*chan*(I) *kin*(eat) *kaw*(rice)". It can be encoded into a morphosyntactic lattice



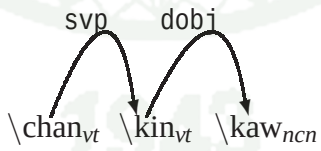
that *chan* has two possible parts-of-speech i.e. personal pronoun (pper) and transitive verb (vt) while *kin* and *kaw* have one possibility, transitive verb and common noun (ncn) respectively. In this context, *chan* should be personal pronoun. But the parser

Algorithm 4 *K*-best version of the modified Eisner's algorithm for lattice structure combining a DIG and a language model

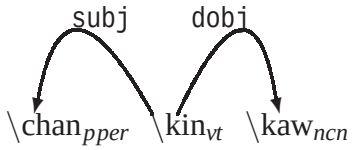
Input: DM a dependency matrix size $n \times n$; $n > 1$, $k \geq 1$, $b \geq 0$, and $e > 0$
Output: a list of k-best parse tree

for all p,d,c **in** $\{1..n\} \times \{\Leftarrow, \Rightarrow\} \times \{0,1\}$ **do**
  $C[p,p,d,c] \leftarrow [\,]$
  for $i = 1$ to k **do**
    $C[p,p,d,c].append(\text{new } TreeObject())$
  end for
end for
for $m = 1$ to n **do**
  for $p = 1$ to $n - m$ **do**
    $t \leftarrow p + m$
    for $r = p$ to t **do**
      for all q **in** $get\_next(r)$ **do**
        if $r < t$ and $IsLegal(p,r,q,t)$ **then**
          $C[p,t,\Leftarrow,0] \leftarrow merge_{\leq k}(C[p,t,\Leftarrow,0], dig\_mult_{\leq k}(C[p,r,\Rightarrow,1], C[q,t,\Leftarrow,1], DM[t,p], b, e)$
          $C[p,t,\Rightarrow,0] \leftarrow merge_{\leq k}(C[p,t,\Rightarrow,0], dig\_mult_{\leq k}(C[p,r,\Rightarrow,1], C[q,t,\Leftarrow,1], DM[p,t], b, e)$
        end if
      end for
      if $r < t$ and $IsLegal(p,r,r,t)$ **then**
        $C[p,t,\Leftarrow,1] \leftarrow merge_{\leq k}(C[p,t,\Leftarrow,1], dig\_mult_{\leq k}(C[p,r,\Leftarrow,1], C[r,t,\Leftarrow,0], 0, b, e)$
      end if
      if $r > p$ and $IsLegal(p,r,r,t)$ **then**
        $C[p,t,\Rightarrow,1] \leftarrow merge_{\leq k}(C[p,t,\Rightarrow,1], dig\_mult_{\leq k}(C[p,r,\Rightarrow,0], C[r,t,\Rightarrow,1], 0, b, e)$
      end if
    end for
  end for
end for
return $C[1,n,\Rightarrow,1]$

will produce



instead of



This is because, in Thai, subjects are often omitted that makes the score of which the first word is transitive verb and also is a root of a sentence is very high. But if we look in morphosyntactical context, "$\backslash chan_{vt}$ $\backslash kin_{vt}$ $\backslash kaw_{ncn}$" is invalid. (Note that *svp* is for annotating serial verb modification.)

In order to overcome these problems, we will introduce two more methods for computing the score of a parse tree: adjusting the score of edges computed and filtering out invalid trees by applying the DIG and the rescoring sub-trees by using a Language Model (LM).

In addition, as our parsing algorithm is inspired by Eisner's algorithm that allows for *k*-best extensions, we can also extend our adapted algorithm to compute the *k*-best trees. With the *k*-best extension, if the function f that computes a new score by merging two sub-trees is monotonic, the complexity of the parsing algorithm will be increased by a multiplicative factor, $O(k\ log\ k)$ (Huang and Chiang, 2005).

We will present a method that efficiently computes the score of the best tree for the morphosyntactic lattice, and is a monotonic function. That will increase the time complexity of *k*-best parsing only by a multiplicative factor $O(k \log k)$.

The pseudo-code of the complete parsing algorithm is shown in Algorithm 4. In the algorithm, there are three parts that are different from Algorithm 3. First, items of table *C* are list of *TreeObject* elements, each containing a tree structure with its score (line 2-5). Second, $dig\_mult_{\leq k}$ is used to find the *k*-best trees of all multiplications between two lists of trees (line 13,14,18,21). Third, we replace *max* with $merge_{\leq k}$ (line 13,14,18,21). The detail of $dig\_mult_{\leq k}$ and $merge_{\leq k}$ will be described in the sections 3.2.2 and 3.2.3 respectively.

3.2.1 The scores of the edges

The score of an edge measures the probability of the dependency relation established between two words. This score has been used by many researchers and can be computed in various ways, for example, by using machine learning

Basic features (4 categories, 18 types)			
Category	Feature number	Feature type	Feature values (Number of values)
A	1	lexeme of x_i (head)	(40,000)
	2	POS of x_i	ncn (common noun), vt (transitive verb),... (51)
	3	generalized POS of x_i	N (noun), V (verb),... (13)
	4	lexeme of y_i (dependant)	same as feature number 1
	5	POS of y_i	same as feature number 2
	6	generalized POS of y_i	same as feature number 3
B	7	distance between x_i and y_i	1,2,3,4,5,[6-10],[11-15],[16-inf) (8)
	8	position of y_i referred x_i	left, right (2)
C	9	POS of x_{i-1}	same as feature number 2
	10	generalized POS of x_{i-1}	same as feature number 3
	11	POS of x_{i+1}	same as feature number 2
	12	generalized POS of x_{i+1}	same as feature number 3
	13	POS of y_{i-1}	same as feature number 2
	14	generalized POS of y_{i-1}	same as feature number 3
	15	POS of y_{i+1}	same as feature number 2
	16	generalized POS of y_{i+1}	same as feature number 3
D	17	POS between x_i and y_i	(51!)
	18	generalized POS between x_i and y_i	(13!)

Figure 24 Features (basic features)

methods such as Maximum Entropy Models (Uchimoto *et al.*, 1999), Support Vector Machines (Kudo and Matsumoto, 2000), and MIRA (McDonald, 2006) or conditional probabilistic models (Eisner, 1996; Jinshan *et al.*, 2004), to estimate the score from linguistic features of various kinds of the two words.

In our work, we used Maximum Entropy Models for learning the score *s*. The features for training the model used here are similar to the first-order features used in (McDonald, 2006). But, we added more back-off features by adding a new tag set which is a POS (part-of-speech) generalization. For example, the seven tags of noun i.e., 'NCN', 'NCT','NNUM', 'NORM', 'NPN', 'NTIT', and 'NLAB', and a personal pronoun, 'PPER' will be reassigned to 'N'. The generalized part-of-speech has 18 different tags. Moreover, we discard 5-gram prefix feature (The 5-gram of a surface word will used as a feature if the word is longer than 5 characters, for instance the 5-gram feature of "general" is "gener".) because it is not appropriate for Thai because Thai is isolating languages and consists of more than 60 characters (excluding symbolic characters). Therefore it is highly possible that words having the same 5-gram prefix are not related. The model is used to estimate dependency probabilities. These probabilities will then be used as scores. In the implementation, we take the logarithm of the probabilities to avoid floating overflow.

Combined features (9 categories, 100 types)		
Combination type	Category	Feature set
Bigram features: related to the information of head and the information of its dependent	(A_1,A_2,B)	$A_1 = \{1,2,3,(1,2),(1,3)\}, A_2 = \{4,5,6,(4,5),(4,6)\}$, $B = \{7,8,\varepsilon\}$
Surround features: related to the POS surrounding head and its dependent	$(\{2\},\{5\},C_1,C_2,B)$ $(\{3\},\{6\},C_3,C_4,B)$	$C_1 = \{9,11\}, C_2 = \{13,15\}$ $C_3 = \{10,12\}, C_4 = \{14,16\}$
In Between POS features: the POS features for all the words in-between the head and its dependent	$(\{2\},\{5\},D_1,B)$ $(\{3\},\{6\},D_2,B)$	$D_1 = \{17\}$ $D_2 = \{18\}$

Figure 25 Features (combined features)

The features used in this work are listed in Figures 24 and 25.

We use the Maximum Entropy Modeling Toolkit for Python (`http://www.homepages.inf.ed.ac.uk/s0450736/maxent_toolkit.html`) for implementing the computing score model. The model is a two-class classifier, deciding whether a pair of words should have a dependency relation or not. In this work, we focus primarily on unlabeled dependencies, but the model can be extended to assign grammatical functions to the dependency structure by using a single-stage (joint labeling) or two-stage method (see (McDonald, 2006) for more detail).

3.2.2 Adjusting the score of edges and filtering out invalid trees

Since our corpus is small (resource-poor languages), the use of the scores of edges mentioned above is not enough. Hence, we propose the use of DIG for adjusting the scores. We adjust the scores of edges by checking whether the tree satisfies a given DIG. If it does not, the score of the edge used to build the tree is decreased and if the number of unsatisfactoriness is greater than a constant *b*, the tree will be filtered out. In order to filter out the invalid tree, we build larger subtrees and derive their corresponding elementary trees as the same time. If the elementary trees cannot be derived, we will adjust the score of the built dependency trees. The score will be decreased by a positive constant *e* if the insertion of two smaller elementary trees fails. An example of adjusting the score of edges is shown in Fig. 26.

Figure 26 gives an example of computing the score of a new subtree

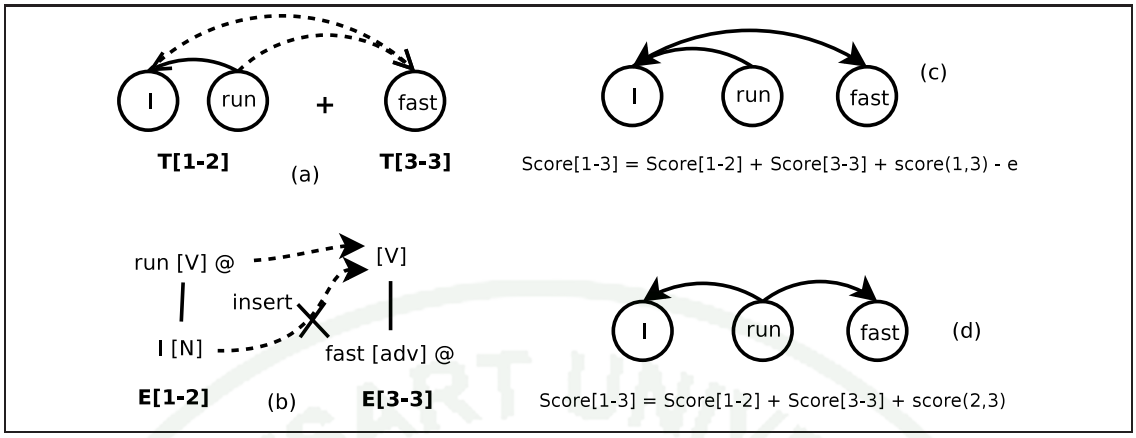


Figure 26 An example of adjusting the scores of edges by using DIG for "I run fast".

where there are two possibilities to build a new subtree from $T[1-2]$ and $T[3-3]$: connecting "I" to "fast" and "run" to "fast" (see 26a). But, the insertion between their elementary trees satisfies only the connection from "run" to "fast" (as 26b), hence the score of the tree constructed from connecting "fast" to "I" is decreased by the constant e (as 26c), while the score of connecting "fast" to "run" is computed normally (as 26d).

If all possible elementary trees corresponding to each substring are kept, the time complexity of the parsing algorithm will be $O(g^n n^3)$, where g is the maximal number of corresponding elementary trees per a smallest substring (a word). In fact, we do not need to keep all possible complete elementary trees. We can keep only the status of the elementary trees corresponding to each word, since the parsing algorithm considers only building dependency relation between two words.

In fact, when the insertion is performed, only the category and relative position of the elementary tree of the inserted tree are considered. Consider again Figure 26b. It shows that the insertion of "I" and "run" into elementary trees correspond to "fast". We do not need to know the elementary trees of "I run". We consider only what the elementary trees of "fast" are waiting for (in this case "fast" is waiting for "V"). Therefore, the time complexity becomes $O(gn^3)$.

The operation of adjusting the scores of edges and filtering out the invalid trees are embedded into the operation $dig\_mult_{\leq k}$ (see Algorithm 5) i.e. the

modified version of $mult_{\leq k}$ (Huang and Chiang, 2005), the multiplication operation that produces the k best trees of all multiplications between two lists of trees. The time complexity of the operation $mult_{\leq k}$ is $O(k \log k)$.

Algorithm 5 $dig\_mult_{\leq k}(C_1, C_2, score, b, e)$

Input: C_1 and C_2 are two lists of *TreeObject*s, $k > 0$, $score \geq 0$, $b \geq 0$, and $e > 0$
Output: a list of *TreeObject*s

```
1  results ← [ ]
2  for all t1,t2 in mult≤k(C1,C2) do
3    if badness(t1,t2) ≤ b then
4      tree ← combine(t1,t2)
5      if check_dig(t1,t2) then
6        tree.score ← t1.score + t2.score + score − e
7      else
8        tree.score ← t1.score + t2.score + score
9      end if
10     results.append(tree)
11   end if
12 end for
13 return results
```

In Algorithm 5, the $badness(t_1, t_2)$ function returns the number of dependencies that do not satisfy the DIG if t_1 and t_2 are combined, and $check\_dig(t_1, t_2)$ returns $true$ if combination of t_1 and t_2 is satisfied by DIG, otherwise $false$. Finally, $combine(t_1, t_2)$ returns a new sub-tree which is a combination of t_1 and t_2. The b constant is used for limiting the value "*badness*" (the number of dependencies not satisfying DIG) that is allowed to occur in the parse trees. The b will be increased dynamically by one if any parse tree cannot be generated.

By adding the operation of adjusting the score of edges and filtering out invalid trees processes into the k-best parsing, the multiplicative factor is still $O(k \log k)$. However, the best sub-tree is no longer optimal because the penalty will be only applied to the k-best sub-trees in each step, so that it is possible that another sub-tree that is not penalized will have a higher score than a sub-tree in the k-best list which is penalized. Therefore, the value of k has an effect on the parsing accuracy: if we increase k, the search space of finding the best sub-trees is also increased. Even though this method does not guarantee the optimum solution, the parsing accuracy can be improved with a small k (see chapter Experiments for the detail of the experiments).

3.2.3 Rescoring sub-trees by using a language model

Rescoring sub-trees by using a LM is very important for finding the best parse tree from all possible segmented words and part-of-speech tagged sentences, since the use of the score of dependencies does not guarantee that the tree with the highest score will correspond to the best word segmentation and best part-of-speech tagging of the sentence (as mentioned in section 3.2). We use a LM to help the parser select a good parse tree which is also a good morphosyntactically analyzed text. Here, we use a trigram model on the part-of-speech as our the LM.

$$P(t_{1,n}) = \prod_{i=1}^{n} P(t_i|w_i)P(t_{i-2}, t_{t-1}|t_i) \tag{4}$$

where t_i and w_i are part-of-speech and word at position i in a given sentence respectively. n is the length of the sentence.

In this work, we use the trigram model because it is fast, simple and easy to implement. Other methods could also give us a reasonable score of morphosyntactic analyzed input, such as Conditional Random Fields, that are theoretically better than the trigram model, but more complex to implement.

Table 1 The ambiguity of the training corpus for LM

ambiguity	**words**	**tokens**
1	16,262 (88.40%)	239,810 (43.73%)
2	1,789 (9.73%)	144,429 (26.34%)
3	270 (1.47%)	94,036 (17.15%)
4	56 (0.30%)	28,241 (5.15%)
5	14 (0.08%)	19,919 (3.63%)
6	5 (0.03%)	21,996 (4.01%)

The trigram model was trained on 40,494 part-of-speech tagged sentences containing 49 tags, 18,396 words and 548,431 tokens. Table 1 shows the detail of the training corpus related to the part-of-speech ambiguity that we found.

The accuracy of the word segmenter and of the part-of-speech

tagger by using the trigram model is about 95% and 90% respectively.

Algorithm 6 $merge_{\leq k}(C_1, C_2)$

Input: C_1 and C_2 are two lists of *TreeObject*s and $k > 0$.
Output: a list of *TreeObject*s

```
1  R ← [ ]
2  L1 ← rescore_by_lang_model(C1)
3  L2 ← rescore_by_lang_model(C2)
4  while len(L1) > 0 and len(L2) > 0 and len(R) < k do
5    if L1.first ≥ L2.first then
6      R.append(L1.first)
7      L1.remove(L1.first)
8    else
9      R.append(L2.first)
10     L2.remove(L2.first)
11   end if
12 end while
13 while len(L1) > 0 and len(R) < k do
14   R.append(L1.first)
15   L1.remove(L1.first)
16 end while
17 while len(L2) > 0 and len(R) < k do
18   R.append(L2.first)
19   L2.remove(L2.first)
20 end while
21 return rescore_by_edges_score(R)
```

We added a rescoring process into the function $merge_{\leq k}$, which takes two sorted lists of length k (or fewer) as input, and outputs the top k in sorted order of the $2k$ elements. For parsing a single input, the elements (sub-trees) are sorted by the score of edges, but here, for parsing a morphosyntactic lattice, the elements are sorted by the score of LM instead. This can be done in $O(k \log k)$ then the overall multiplicative factor ($dig\_mult_{\leq k}$ and $merge_{\leq k}$ operations) is still $O(k \log k)$. The pseudo-code is shown in Algorithm 6.

In fact, the rescoring process can be added into the final parse trees, but we may have to set k extremely high in order to find the true best parsing (taking the LM into account).

RESULTS AND DISCUSSION

Results

1. Evaluation Methods

To evaluate our methods, we set up three experiments. In the first, we assume that inputs are correctly word segmented and part-of-speech tagged. In this experiment, we can directly compare our method to the others. In the second experiment, we will not assume that the inputs are perfect, but we will convert the inputs into morphosyntactic lattices to test our method. For the others, the inputs will be analyzed by the existing morphosyntactic tools. Finally, we study the oracle parse (Shen, 2006), or the best parse among the top *k* parses in order to measure the performance of the *k*-best parsing.

For the experiments, we used 716 sentences of the NAiST treebank for training and 100 other sentences for testing. We measured Dependency Precision (DP), Complete Rate (CR) and Root Accuracy (RA) to evaluate the parsing results. We measured only the correctness of the dependency structures, without considering the grammatical functions.

$$\begin{aligned} \text{DP} &= \frac{\text{number of correct dependencies}}{\text{total number of reference dependencies}} \\ \text{CR} &= \frac{\text{number of complete parse trees}}{\text{total number of sentences}} \\ \text{RA} &= \frac{\text{number of correct root nodes}}{\text{total number of sentences}} \end{aligned}$$

In the second and third experiments, we also measured the correctness of the morphosyntactic analysis. We used Token Precision (TP), Token Recall (TR), and Token F-measure (TF).

$$\begin{aligned} \text{TP} &= \frac{\text{number of correct tokens}}{\text{total number of reference tokens}} \\ \text{TR} &= \frac{\text{number of correct tokens}}{\text{total number of hypothesis tokens}} \\ \text{TF} &= \frac{2 \times \text{TP} \times \text{TR}}{\text{TP} + \text{TR}} \end{aligned}$$

Here, the number of correct tokens means the number of the tokens that are correctly word-segmented and part-of-speech tagged.

2. Extracted elementary trees

The extended forms of elementary trees were extracted from the training corpus, 716 sentences, there are 8,172 tokens, of which 1,996 are type-I, 4,342 are type-II and 1,834 are type-III.

In the elementary trees extracted using the algorithm described in section 2.3, there are 981 different types of elementary trees (175, 400 and 406 for Type-I, Type-II and Type-III, respectively), and 497 of them appear only once. Some of these elementary trees are abnormal structures, especially those of low number of occurrences in the corpus.

Obviously, the extracted elementary trees do not cover all the words. Therefore, if the lexicalized elementary trees (the head is a lexicon with its part-of-speech) can not be found, the unlexicalized elementary trees (the head is a part-of-speech) will be matched instead. Specifically, the unlexicalized elementary trees which have a high number of occurrences in the corpus (> 3) will be used in order to avoid using noisy elementary trees.

3. Parsing with perfect inputs

Although parsing with perfect inputs are not the main focus of this work, observing the accuracy of parsing with a perfect input can help investigate the performance of combining DIG with a data-driven parsing method, and also with

other parsers. Here, we use the MSTparser (`http://sourceforge.net/projects/mstparser`), a statistical dependency parser freely available on the web. The MSTparser was trained with the parameters $k = 5$ and $N = 10$, as reported in (McDonald, 2006) yielding a good accuracy and training the model in reasonable time.

For our parsing algorithm, there are three parameters which can affect the performance i.e. k, e, and b. Therefore, we set up another experiment for observing the effect of these parameters by letting them vary.

Having experimented, we found that increasing k does not improve the accuracy. Moreover, the accuracy dropped at some higher k. The idea of using b does not seem to work in this case, because the accuracy improves if b is disabled (set to ∞). The e that can improve the accuracy from the baseline ($e = 0$ and $b = \infty$) is $0 < e \leq 2$.

From the experiments, we should prioritize the score of edges computed from a data-driven model rather than weighting the score by DIG for parsing with perfect inputs.

Table 2 Results comparing our systems with MSTparser where the input is perfect (for MAX_{DIG}, we set $k = 1$, $b = \infty$ and $e = 2$).

	DP	**CR**	**RA**
MAX (baseline)	86.03	21.00	90.00
MST_{proj}	83.40	15.00	**94.00**
MAX_{DIG}	**88.66**	**27.00**	92.00

Results of parsing with perfect inputs are shown in Table 2. We use subscript *DIG* to denote the use of DIG while *MAX* represents our learning model i.e. Maximum Entropy Models and MST_{proj} is MSTparser using projective parsing algorithm for training the model.

For overall performance, MAX_{DIG} is the best one. It confirms that the use of the DIG can improve the parsing accuracy of a data-driven parsing model. The accuracy

of MST_{proj} is lower than the baseline, even if MSTparser uses the learning model, MIRA (Crammer *et al.*, 2006), that is theoretically better than the model used in the baseline. We think that this is due to the 5-gram prefix features used in MSTParser that does not make sense for Thai, and the simplified part-of-speech features that was added into the baseline model. Also, we see that the better performance of MIRA via the root accuracy of MST_{proj} is higher than the others, because MIRA learns the scores of edges by using a whole dependency tree rather than each pair of dependency nodes.

4. Parsing with morphosyntactic lattices

In this experiment, we created a morphosyntactic lattice by mapping a dictionary using a dynamic backtracking algorithm (Thumkanon, 2001), and generating all possible word segmentations and part-of-speech tagging results. Here, we assume there is no unknown word in the input text, in order to assure that there is a correct *homophrase* in the lattice. For the other methods that cannot take the lattice structure as input, we used input texts that were morphosyntactically analyzed by the analyzer instead. We used the trigram model trained by the same corpus that was used in the LM rescoring process.

Like in the previous experiment, we also observe the effect of varying the parameters. As we expected, the result is the opposite to those obtained in the experiment with perfect inputs. The parsing accuracy improved when k increased and was stable when $k \geq 10$ (k varied from 1 to 20). The accuracy is highest when $b = 0$ and $4 \leq e \leq 5$.

The experiment shows that we can trust the use of DIG more than in the score computed by the learning model, unlike in the previous experiment, because when the input is a lattice, the effect of the independence assumption is more evident, and many invalid dependencies are produced. Hence, the use of DIG plays an important role in this experiment.

Table 3 compares the results. We use the superscript $*$ to indicate methods taking a lattice as input, and superscript d to indicate that the input is a dubiously

Table 3 Results comparing our systems with the MSTparser where the input is the lattice or the text analyzed by the morphosyntactic analyzer (for MAX^*_{DIG} and $MAX^*_{DIG\text{-}LM}$, we set $k = 10$, $b = 0$ and $e = 5$).

	DP	**CR**	**RA**
MAX^d (baseline)	68.01	4.00	75.00
MAX^d_{DIG}	68.39	5.00	77.00
MST^d_{proj}	65.99	4.00	78.00
MAX^*	66.42	4.00	76.00
MAX^*_{DIG}	68.39	4.00	79.00
MAX^*_{LM}	73.43	5.00	79.00
$MAX^*_{DIG\text{-}LM}$	**74.32**	**6.00**	**81.00**

disambiguated string. We also used subscript the *LM* to indicate the use of a language model for rescoring.

The $MAX^*_{DIG\text{-}LM}$ method is the best one, and its accuracy is far better than that of the parsers used on inputs produced by the used morphosyntatical analyzer. Moreover, the accuracy of morphosyntactic analysis of the results is also improved, as shown in Table 4.

Table 4 Results comparing the accuracy of morphosyntactic analysis

	TR	**TP**	**TF**
trigram model	88.64	87.34	87.99
MAX^*	87.12	88.08	87.59
MAX^*_{DIG}	88.11	88.88	88.49
MAX^*_{LM}	92.86	93.27	93.06
$MAX^*_{DIG\text{-}LM}$	**93.15**	**93.49**	**93.32**

The MAX^* method is the worst: the accuracy of parse trees and the accuracy of morphosyntactic analysis are lower than the baseline. That is similar to the MAX^*_{DIG} method, which shows that using only the DIG can slightly improve the parse accuracy only, but does not improve the morphosyntactic analysis, because the score of morphosyntactic analysis is not taken into account. The use of DIG can improve the parsing accuracy, but it is not enough. By contrast, MAX^*_{LM} uses only the LM but

obviously improves the accuracy of morphosyntactic analysis and that of the parse trees.

The results show that if we perform morphosyntatic analysis and syntactic analysis simultaneously by using their information to help each other (here we use the score of morphosyntactic analysis to rescore the parse trees), the accuracy of that combination is better than that of the usual sequence (morphosyntactic analysis followed by syntactic analysis).

Table 5 Speed of parsing with morphosyntactic lattices.

num. of words	num. of nodes	num. of all paths	execution time (sec)
1-5	7.39	15.42	0.17020
6-10	15.17	518.87	2.5216
11-15	23.55	7,418.24	9.1375
16-20	30.89	488,910.57	30.6499
21-25	39.58	6,063,285.00	93.3398
26-30	47.25	54,095,126.00	328.7566

Table 5 shows speed of parsing comparing to number of words, number of nodes and number of paths in the input lattices. The results shows that the speed of our parsing model depends on the number of nodes (in order of $O(n^3)$) in a input lattice as we mentioned before.

5. *K*-best parsing

The next experiment is on *k*-best parsing. We use the same algorithm with the best parameter settings ($b = 0$ and $e = 5$) as in the previous section, and we also study the oracle parse, or the best parse, among the top 10 parses. The result is shown in Table 6. Note that the MSTparser which we used cannot produce the *k* best parse trees, hence no result is given for it.

For the oracle parse, the MAX^*_{DIG} method becomes the best one. In addition, if we increase *k* to 100, MAX^* also outperforms MAX^*_{LM} and $MAX^*_{DIG\text{-}LM}$. It shows that the models using a LM for rescoring worse in the oracle parse.

Table 6 The accuracy of oracle parse in the 10-best parses

	DP	**CR**	**RA**
MAX^{d}	72.44	7.00	89.00
MAX^{d}_{DIG}	73.03	9.00	90.00
MAX^{*}	74.43	12.00	84.00
MAX^{*}_{DIG}	**78.02**	**13.00**	**93.00**
MAX^{*}_{LM}	76.03	11.00	84.00
$MAX^{*}_{DIG\text{-}LM}$	76.42	11.00	84.00

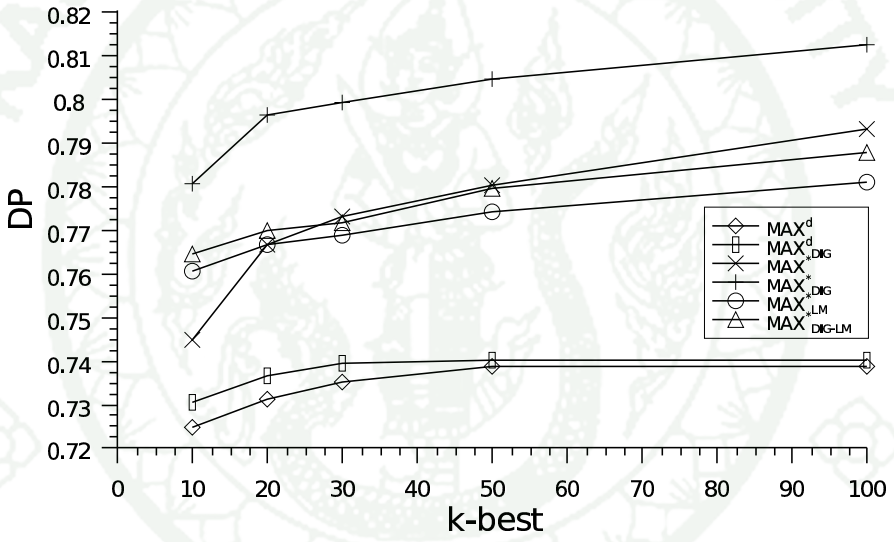


Figure 27 DP of the *k*-best oracle on the test data

We notice that the methods using a LM for rescoring produce a lot of parse trees having duplicated patterns of morphosyntactic analysis. This is because the rescoring by the LM method lets the parsing process consider the score of LM first and the score of dependencies later. In other words, the text of highest score of LM is first selected and as many corresponding parse trees as possible will be generated. It leads to imbalanced decision-making by the parser, that too much emphasizes the score of LM. This problem occurs only in the *k*-best parsing, it does not affect the best parsing. Therefore, the rescoring method should not be used in *k*-best parsing.

Figures 27 and 28 show the DP and the TF of the oracle on *k*-best parsing where

k is 10, 20, 30, 50 and 100.

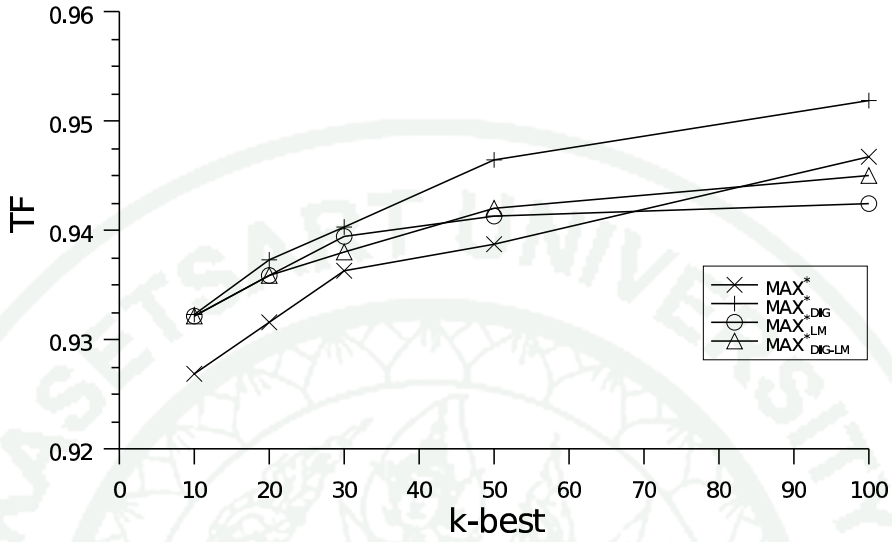


Figure 28 TF of the *k*-best oracle on the test data

The DP and the TF of oracle in 100-best parsing with the MAX^*_{DIG} method are 81.21% and 95.18%, respectively, and both tend to continuously increase if *k* increases even more. That is similar to other methods for which the input is a morphosyntactic lattice. Unlike the methods where the input is a dubiously disambiguated string, the DP of oracle tends to improve at first, but slightly increases when *k* increases even more and becomes saturated when $k \geq 50$: the accuracy of the morphosyntactic analysis cannot improve anymore. In other words, the parsing accuracy is first limited by the accuracy of morphosyntactic analysis of the input. Conversely, if we increase *k* when inputs are morphosyntactic lattices, the search space for finding both the best morphosyntactic analysis and the best parse tree is enlarged. Hence, the chance to find a best parse tree among *k*-parse trees is higher than when the inputs are dubiously disambiguated strings.

Clearly, the use of a morphosyntactic lattice as input, in case of the *k*-best parsing, significantly and decisively improves the accuracy (relative to the oracle parse) of the morphosyntactic analysis and the parsing process.

Discussion

The limitation of the proposed parser caused by three types of input.

Noun-phrase with multiple nouns: If a subject or an object of a sentence is modified by nouns, the parser would select the wrong subject or object. The modifier is usually selected instead of the true subject or object.

Relative pronoun omission: In Thai, relative pronouns, clue words for indicating relative clauses, are usually omitted. The parser will select the root incorrectly by promoting the verb of the relative clause instead of selecting the verb of the main clause.

Spoken language: In the spoken language, some constituents of sentences are omitted such as subject, object and main verb. The errors will occur if the main verb, the root, is omitted. However, the parser can handle the sentences that subject and object are omitted.

CONCLUSION AND RECOMMENDATION

Conclusion

Syntactic parsers are very important for development of NLP applications such as machine translation, information extraction, text summarization, etc. In this work, we built a dependency parser that suits to under-resourced languages (no huge treebank and no good morphosyntactic analyzer), hence we proposed a dependency parsing method for languages without reliable morphosyntactic analysis tools. For this, we represent the input as a morphosyntactic lattice structure, and apply an adaptation of Eisner's algorithm to find projective dependency trees. From the experimental results, our method performs better than the results working on dubiously disambiguated strings. In addition, we also show the use of Dependency Insertion Grammar (DIG) to adjust the scores computed by the statistical parsing models, and the rescore of the parse trees by the LM, and a *k*-best extension of the parser.

The results show that our methods can significantly improve the parsing accuracy. The highest parsing accuracy (DP) reported in this paper is 74.32% which is 6.31% improvement compared to the models taking as inputs the results of unreliable morphosyntactic analysis tools. Even if the obtained accuracy is not enough for high-level NLP applications such as MT, Information Extraction and Text Summarization, it is still useful for a corpus preparing process that requests a parser which can produce reasonable *k*-best parse trees in order to let annotators start from the best tree among *k*-best trees rather than from a doubtful parse tree.

This work is the first step of development of a dependency parser for under-resourced languages. There are many opened problems which the solution could improve the parsing models in the future. For instance, the proper combination of the morphosyntactic process and the syntactic process can improve the performance of each other and the use of linguistic knowledge can improve the data-driven parsing model.

Recommendation

1. Language Independent

Although we proposed a dependency parsing method for Thai, the proposed method can be applied to other languages with a little effort. Before applying our parsing method to a new language, we need at least one method to classify between the complement and the adjunct dependencies in order to be able to extract elementary trees.

In addition, tuning of feature selection for the learning model is also necessary, because each language has its own specific behavior. If a proper feature space is selected, the parsing accuracy can be significantly improved as shown in our experiments: the less performance learning method beats up the better one which the proper feature space is obtained.

2. Components

The parsing system in this study is based on the algorithm (Algorithm 4) that can effectively decode the highest score projective dependency tree among all weighted dependencies established in the lattice structure. Therefore, the parsing accuracy of the system depends on a method used for computing the score (weight) of the dependencies. In this work, The proposed method for computing the score consists of three processes and each process works independently of each other. Hence, each process can be substituted by others that works in the same manner.

The first process is the computing the score of edges (dependencies) by using machine learning model. Here, Maximum Entropy Models is used. It can be replaced by any learning methods that can also produce the reasonable score such as MIRA, Support Vector Machine, etc.

The second process is the adjusting the score and filtering out the invalid parse trees. In this report, we proposed the use of DIG. In fact, other methods that can

linguistically validate parse trees can be used in place or one may try to use other grammars instead of the DIG.

The last process is the rescoring parse trees by using LM. The trigram model based on part-of-speech is used to represent the LM. Essentially, there are methods that can also be used for represent the LM, i.e. Conditional Random Fields.

It is believed the parsing accuracy can be further improved and it depends on the methods used in each process.

LITERATURE CITED

Alshawi, H.. 1996. Head Automata and Bilingual Tiling: Translation with Minimal Representations, pp. 167–176. *In* **Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL '96)**.

Boitet, C. and Y. Zaharin. 1988. Representation Trees And String-Tree Correspondences, pp. 59–64. *In* **Proceedings of the 4th international conference on Computational Linguistics (COLING 1988)**. Budapest, Hungary.

Chu, Y. J. and T. H. Liu. 1965. On the shortest arborescence of a directed graph. **Science Sinica**. 14:1396–1400.

Clark, S. and J. Curran. 2004. Parsing the WSJ using CCG and log-linear models, p. 103. *In* **Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics**. Barcelona, Spain.

Clark, S., J. Hockenmaier and M. Steedman. 2001. Building deep dependency structures with a wide-coverage CCG parser, pp. 327–334. *In* **Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL '02)**. Philadelphia, Pennsylvania.

Collins, C., B. Carpenter and G. Penn. 2004. Head-driven parsing for word lattices, p. 231. *In* **Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics**. Barcelona, Spain.

Collins, M.. 1997. Three generative, lexicalised models for statistical parsing, pp. 16–23. *In* **Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics**. Madrid, Spain.

Collins, M.. 1999. **Head-driven statistical models for natural language parsing**. Ph.D. thesis. Computer and Information Science, University of Pennsylvania.

Collins, M., L. Ramshaw, J. Hajič and C. Tillmann. 1999. A statistical parser for Czech, pp. 505–512. *In* **Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics**. College Park, Maryland.

Covington, M.. 1984. **Syntactic theory in the High Middle Ages: Modistic models of sentence structure**. Cambridge University Press.

Crammer, K., O. Dekel, J. Keshat and S. Shalev-Shwartz. 2006. Online Passive-aggressive algorithms. **Journal of Machine Learning Research**. 7:551–585.

Ding, Y. and M. Palmer. 2004. Synchronous Dependency Insertion Grammars: A Grammar Formalism for Syntax-Based Statistical MT. *In* **Proceedings of the workshop on Recent Advances in Dependency Grammars, The 20th International Conference on Computational Linguistics (COLING 2004)**. Geneva, Switzerland.

Ding, Y. and M. Palmer. 2005. Machine translation using probabilistic synchronous dependency insertion grammars, pp. 541–548. *In* **Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL-05)**. Association for Computational Linguistics, Ann Arbor, Michigan.

Duchier, D.. 2000. Constraint Programming For Natural Language Processing. *In* **Lecture Notes, ESSLLI 2000**.

Edmonds, J.. 1967. Optimum Branchings. **Journal of Research of the National Bureau of Standards**. 71B:233–240.

Eisner, J.. 1996. Three New Probabilistic Models for Dependency Parsing: An Exploration, pp. 340–345. *In* **Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)**. Copenhagen.

Eisner, J.. 2000. Bilexical Grammars and Their Cubic-Time Parsing Algorithms. *In* H. Bunt and A. Nijholt, eds., **Advances in Probabilistic and Other Parsing Technologies**. pp. 29–62. Kluwer Academic Publishers.

Gagnon, M. and L. D. Sylva. 2005. Text Summarization by Sentence Extraction and Syntactic Pruning. *In* **Proceedings of the 3rd Computational Linguistics in the North East (CliNE-05)**. Université du Québec en Outaouais, Gatineau, Canada.

Gaifman, H.. 1965. Dependency Systems and Phrase-Structure Systems. **Information and Control**. 8(3):304–337.

Hajič, J., A. Böhmová, E. Hajičová and B. Vidová-Hladká. 2000. The Prague Dependency Treebank: A Three-Level Annotation Scenario. *In* A. Abeillé, ed., **Treebanks: Building and Using Parsed Corpora**. pp. 103–127. Amsterdam: Kluwer Academic.

Harper, M. and R. Helzerman. 1995. Extensions to constraint dependency parsing for spoken language processing. **Computer Speech and Language**. 9:187–234.

Harper, M., L. Jamieson, C. Zoltowski and R. Helzerman. 1993. Semantics and Constraint Parsing of Word Graphs, pp. 63–66. *In* **Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing**. Minneapolis, Minnesota.

Hays, D.. 1964. Dependency Theory: a Formalism and Some Observations. **Language**. 40(4):511–525.

Heinecke, J., J. Kunze, W. Menzel and I. Schröder. 1998. Eliminative parsing with graded constraints, pp. 526–530. *In* **Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics**. Montreal, Quebec, Canada.

Hellwig, P.. 1986. Dependency Unification Grammar, pp. 195–198. *In* **Proceedings of the 11th coference on Computational linguistics**. Bonn, Germany.

Huang, L. and D. Chiang. 2005. Better k-best parsing. *In* **Proceedings of the International Workshop on Parsing Technologies (IWPT)**. Vancouver, B.C., Canada.

Hudson, R.. 1990. **English Word Grammar**. Blackwell.

Hwa, R., P. Resnik, A. Weinberg, C. Cabezas and O. Kolak. 2005. Bootstrapping parsers via syntactic projection across parallel texts. **Natural Language Engineering**. 11(3):311–325.

Jinshan, M., Z. Yu, L. Ting and L. Sheng. 2004. A Statistical Dependency Parser of Chinese under Small Training Data. *In* **Proceedings of the 1st International Joint Conference on Natural Language Processing (IJCNLP)**. Sanya City, Hainan Island, China.

Joshi, A. and O. Rambow. 2003. A Formalism for Dependency Grammar Based on Tree Adjoining Grammar. *In* **Proceedings of the Conference on Meaning-Text Theory**. Paris, France.

Joshi, A. and Y. Schabes. 1997. Tree-Adjoining Grammars. *In* **Handbook of Formal Languages**, vol. 3. pp. 69–124. Springer, Berlin, New York.

Karlsson, F.. 1990. Constraint grammar as a framework for parsing running text, pp. 168–173. *In* **Proceedings of the 13th conference on Computational linguistics**. Helsinki, Finland.

Kim, J. J. and J. C. Park. 2004. Annotation of Gene Products in the Literature with Gene Ontology Terms Using Syntactic Dependencies, pp. 528–534. *In* **Proceedings of the 1st International Joint Conference on Natural Language Processing (IJCNLP)**. Sanya City, Hainan Island, China.

Kudo, T. and Y. Matsumoto. 2000. Japanese Dependency Structure Analysis Based on Support Vector Machines, pp. 18–25. *In* **Proceedings of the Joint SIGDAT conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000)**. Hong Kong.

Maruyama, H.. 1990. Structural disambiguation with constraint propagation, pp. 31–38. *In* **Proceedings of the 28th annual meeting on Association for Computational Linguistics**. Association for Computational Linguistics, Pittsburgh, Pennsylvania, USA.

McDonald, R.. 2006. **Discriminative Training and Spanning Tree Algorithms for Dependency Parsing**. Ph.D. thesis. Computer and Information Science, University of Pennsylvania.

McDonald, R. and J. Nivre. 2007. Characterizing the Errors of Data-Driven Dependency Parsing Models. *In* **Proceedings of Conference on Empirical Methods in Natural Language Processing and Natural Language Learning (EMNLP-CoNLL)**. Prague, Czech Republic.

Mel'čuk, I.. 1988. **Dependency Syntax: Theory and Practice**. The SUNY Press.

Milward, D.. 1994. Dynamic dependency grammar. **Linguistics and Philosophy**. 17:561–605.

Nivre, J.. 2005. Dependency Grammar and Dependency Parsing. MSI report 05133. School of Mathematics and Systems Engineering, Växjö University.

Nivre, J., J. Hall and J. Nilsson. 2004. Memory-Based Dependency Parsing, pp. 49–56. *In* **Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL-2004)**. Boston, Massachusetts, USA.

Nivre, J. and M. Scholz. 2004. Deterministic dependency parsing of English text, p. 64. *In* **Proceedings of the 20th international conference on Computational Linguistics (COLING '04)**. Geneva, Switzerland.

Ratnaparkhi, A.. 1999. Learning to Parse Natural Language with Maximum Entropy Models. **Machine Learning**. 34(1-3):151–175.

Sgall, P., E. Hajicová and J. Panevová. 1986. **The Meaning of the Sentence in its Semantic and Pragmatic Aspects**. Springer. 1st edn.

Shen, L.. 2006. **Statistical LTAG Parsing**. Ph.D. thesis. Computer and Information Science, University of Pennsylvania.

Shen, L. and A. K. Joshi. 2005. Incremental LTAG parsing, pp. 811–818. *In* **Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing**. Vancouver, B.C., Canada.

Sleator, D. and D. Temperly. 1991. Parsing English with a link grammar, pp. 277–292. *In* **Proceedings of the third International Workshop on Parsing Technologies (IWPT)**.

Sudprasert, S., A. Kawtrakul, C. Boitet and V. Berment. 2009. Dependency Parsing with Lattice Structures for Resource-Poor Languages. **IEICE Transactions on Information and Systems**. E92-D(10):2122–2136.

Tapanainen, P. and T. Järvinen. 1997. A non-projective dependency parser, pp. 64–71. *In* **Proceedings of the 5th Conference on Applied Natural Language Processing**. Association for Computational Linguistics, Washington, D.C.

Tesnière, L.. 1959. **Éléments de syntaxe structurale**. Editions Klincksieck, Paris.

Thumkanon, C.. 2001. **A Statistical Model for Thai Morphological Analysis**. Master's thesis. Computer Engineering, Kasetsart University.

Tomita, M.. 1986. An Efficient Word Lattice Parsing Algorithm for Continuous Speech Recognition. *In* **Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing**. Tokyo, Japan.

Uchimoto, K., S. Sekine and H. Isahara. 1999. Japanese dependency structure analysis based on maximum entropy models, pp. 196–203. *In* **Proceedings of the 9th conference on European chapter of the Association for Computational Linguistics**. Association for Computational Linguistics, Bergen, Norway.

Vijay-Shankar, K. and A. Joshi. 1985. Some computational properties of Tree Adjoining Grammars, pp. 82–93. *In* **Proceedings of the 23rd annual meeting on Association for Computational Linguistics**. Association for Computational Linguistics, Morristown, NJ, USA.

Wacharamanotham, C., M. Suktarachan and A. Kawtrakul. 2007. The Development of Web-based Annotation System for Thai. *In* **Proceedings of the 7th International Symposium on Natural Language Processing**. Pattaya, Chonburi, Thailand.

Wang, W. and M. Harper. 2004. A Statistical Constraint Dependency Grammar (CDG) Parser, pp. 42–49. *In* **Proceedings of the ACL Workshop Incremental Parsing: Bringing Engineering and Cognition Together**. Association for Computational Linguistics, Barcelona, Spain.

Xia, F.. 1999. Extracting Tree Adjoining Grammars from Bracketed Corpora. *In* **the 5th Natural Language Processing Pacific Rim Symposium (NLPRS-99)**. Beijing, China.

Yakushiji, A., Y. Miyao, Y. Tateisi and J. Tsuji. 2005. Biomedical Information Extraction with Predicate-Argument Structure Patterns, pp. 60–69. *In* **Proceedings of the First International Symposium on Semantic Mining in Biomedicine**. Hinxton, Cambridgeshire, UK.

Yamada, H. and Y. Matsumoto. 2003. Statistical Dependency Analysis With Support Vector Machines, pp. 195–206. *In* **Proceedings of the 8th International Workshop on Parsing Technologies**. Nancy, France.

APPENDICES

Appendix A

Part-of-speech

Noun

1. proper noun (npn)
2. cardinal number (nnum)
3. ordinal number marker (norm)
4. label noun (nlab)
5. common noun (ncn)
6. collective noun (nct)
7. title noun (ntit)

Pronoun

1. personal pronoun (pper)
2. demonstrative pronoun (pden)
3. indefinite pronoun (pind)
4. possessive pronoun (ppos)
5. reflexive pronoun (prfx)
6. reciprocal pronoun (prec)
7. relative pronoun (prel)
8. interrogative pronoun (pint)

Verb

1. intransitive verb (vi)

2. transitive verb (vt)
3. causative verb (vcau)
4. complementary state verb (vcs)
5. existential verb (vex)
6. pre-verb (prev)
7. post-verb (vpost)
8. honorific marker (honm)

Determiner

1. determiner (det)
2. indefinite determiner (indet)

Adjective

1. adjective (adj)

Adverb

1. adverb (adv)
2. adverb marker 1 (adm1)
3. adverb marker 2 (adm2)
4. adverb marker 3 (adm3)
5. adverb marker 4 (adm4)
6. adverb marker 5 (adm5)

Classifier

1. classifier (cl)

Conjection

1. conjection (conj)
2. double conjection (conjd)
3. noun clause conjection (conjncl)

Preposition

1. preposition (prep)
2. co-preposition (prepc)

Interjection

1. interjection (int)

Prefix

1. Prefix 1 (pref1)
2. Prefix 2 (pref2)
3. Prefix 3 (pref3)

Particle

1. affirmative (aff)
2. particle (part)

Negative

1. negative (neg)

Punctuation

1. punctuation (punc)

Idiom

1. idiom (idm)

Passive Voice Marker

1. passive voice marker (psm)

Symbol

1. symbol (sym)

Appendix B

Grammatical Functions

We classify Thai grammartical functions into two main groups i.e. complements and adjuncts. There are 12 complements and 17 adjuncts.

Complements

1. subject (subj)
2. clausal subject (csubj)
3. direct object (dobj)
4. indirect object (iobj)
5. prepositional object (pobj)
6. prepositional complement (pcomp)
7. subject or object predicative (pred)
8. clausal predicative (cpred)
9. conjunction (conj)
10. subordinating conjunction (sconj)
11. nominalizer (nom)
12. adverbalizer (advm)

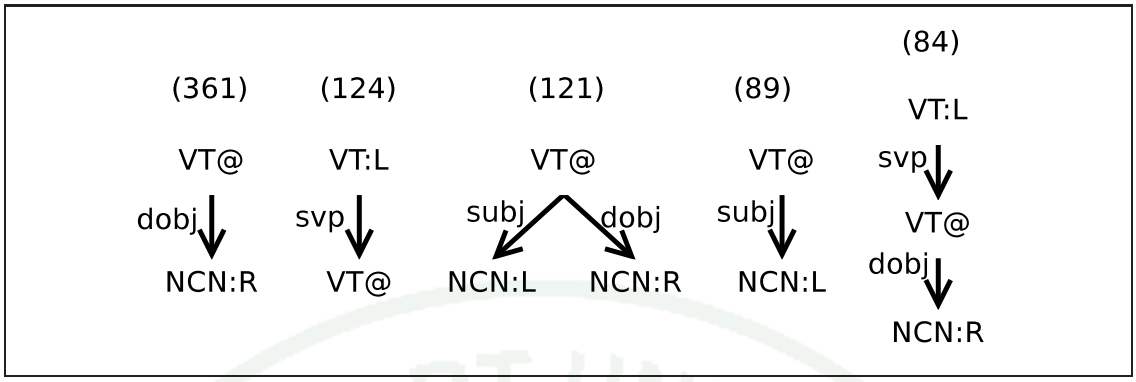
Adjuncts

1. parenthetical modifier (modp)
2. restrictive modifier (modr)
3. mood modifier (modm)
4. aspect modifier (moda)
5. locative modifier (modl)

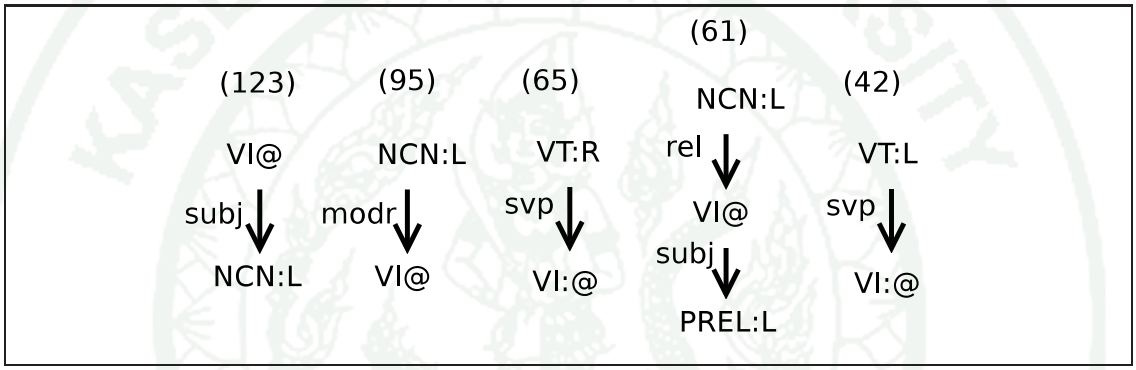
6. parenthetical apposition (appa)
7. restrictive apposition (appr)
8. relative clause modification (rel)
9. determiner (det)
10. quantifier (quan)
11. classifier (cl)
12. coordination (coord)
13. negation (neg)
14. punctuation (punc)
15. double preposition (dprep)
16. parallel serial verb (svp)
17. sequence serial verb (svs)

Appendix C

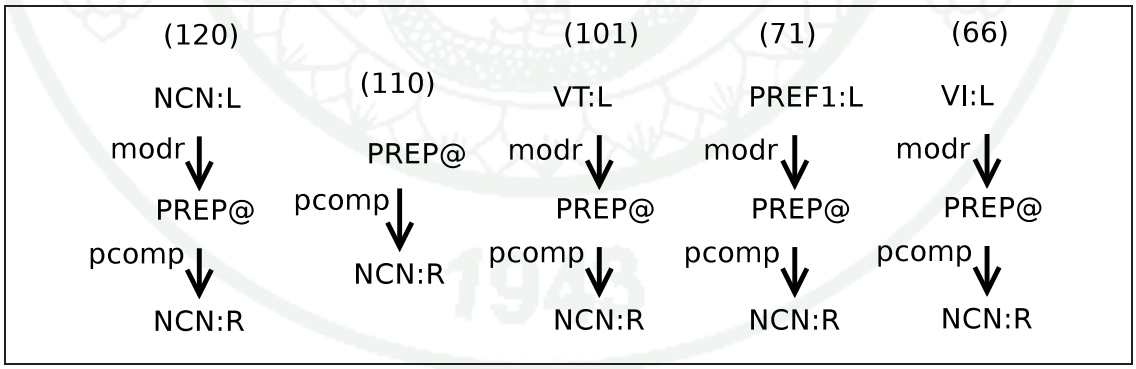
Extracted Elementary Trees from NAIST Treebank



Appendix Figure C1 Top 5 of the most occurrence relaxed elementry trees of transitive verb (vt)



Appendix Figure C2 Top 5 of the most occurrence relaxed elementry trees of intransitive verb (vi)

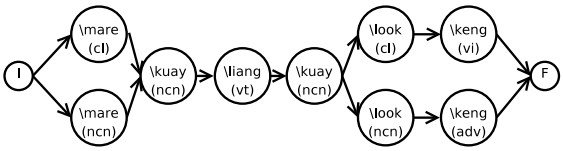


Appendix Figure C3 Top 5 of the most occurrence relaxed elementry trees of preposition (prep)

Appendix D

K-best Parsing Results

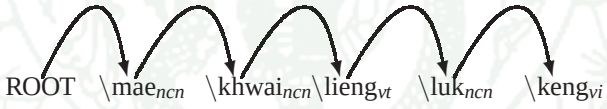
Input: “\mare(mother) \kuay(buffalo) \liang(take care) \look(kid) \keng(well)”



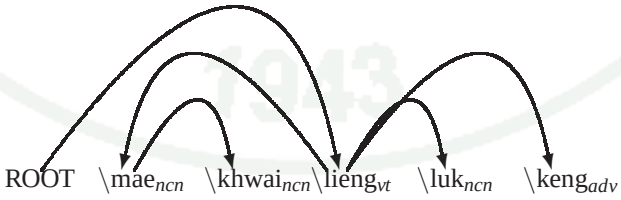
Output: 5-best unlabeled parse trees

1. score=2.863449

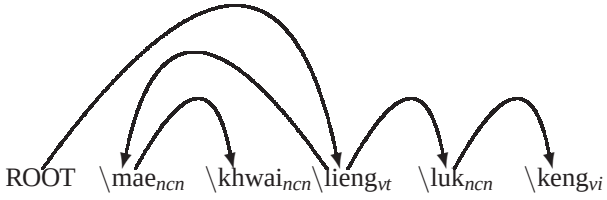
2. score=2.760695



3. score=-0.622163 (correct tree)



4. score=-0.724917



5. score=-0.840073

CURRICULUM VITAE

NAME : Mr. Sutee Sudprasert

BIRTH DATE : March 15, 1979

BIRTH PLACE : Bangkok, Thailand

EDUCATION :

YEAR	INSTITUTE	DEGREE/DIPLOMA
2001	Kasetsart Univ.	B.Eng (Computer Engineering)
2005	Kasetsart Univ.	M.Eng (Computer Engineering)

POSITION/TITLE : Lecturer

WORK PLACE : Faculty of Science, Kasetsart University

SCHOLARSHIP/AWARDS : Commission on Higher Education 2003-2008