Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

Clause Splitting with Conditional Random Fields

Vinh Van Nguyen†, Minh Le Nguyen† and Akira Shimazu†

In this paper, we present a Conditional Random Fields (CRFs) framework for the Clause Splitting problem. We adapt the CRFs model to this problem in order to use very large sets of arbitrary, overlapping and non-independent features. We also extend N-best list by using the Joint-CRFs (Shi and Wang 2007). In addition, we propose the use of rich linguistic information along with a new bottom-up dynamic algorithm for decoding to split a sentence into clauses. The experiments show that our results are competitive with the state-of-the art results.

Key Words: *Computational Linguistics, Partial Parsing, Clause Splitting*

1 Introduction

Clause Splitting (CS) is the task of splitting a complex sentence into several clauses. This task is important for various tasks such as machine translation, aligning parallel texts, text to speech systems and transformation by natural language sentences into logical forms and it became the shared-task problem in CoNLL 2001 (Tjong Kim Sang and Dejean 2001). CS is a deeper level of partial parsing, which is the task of recovering only a limited amount of syntactic information.

Machine learning techniques in the last decade have permeated most areas of natural language processing. The reason is that a vast number of machine learning algorithms have proved to be able to learn from natural language data given a relatively small correctly annotated corpus. Therefore, machine learning algorithms make it possible to within a short period of time develop language resources (data analyzed on various linguistic levels) that are necessary for numerous applications in natural language processing. For partial parsing such as CS, there is a lot of interest in the design of learning systems which perform only a partial analysis of a sentence (Abney 1991; Hammerton et al. 2002).

Recently, many machine learning methods have been successfully applied to partial parsing tasks. Most learning models used for partial parsing tasks are discriminative models. These approaches have reached the-state-of-the-art of partial parsing (Kudo and Matsumoto 2001; Carreras and Marquez 2005; Ando and Zhang 2005).

(Carreras and Marquez 2005) proposes the discriminative model, so-called FR-Perceptron (Collins 2002) for recognizing structures of clauses. They used a global online learning algorithm

† School of Information Science, Japan Advanced Institute of Science and Technology

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

to train a discriminative model for CS. However, FR-Perceptron updates the learning functions, depending on the predictions on that example and computation time for the training and coding process are time-consuming in large training sets as described below. This can be inefficient when we apply CS to other applications.

This paper builds on the previous works described by (Nguyen et al. 2007) with a novel and efficient method for CS as an alternative training method to FR-Perceptron. The CRFs model (Lafferty, McCallum, and Pereira 2001) defines a conditional distribution over labeling given an observation and it allows for the use of very large sets of arbitrary, overlapping and non-independent features and has efficient training and decoding processes which both find globally optimal solution. (Roark, Saraclar, Collins and Johnson 2004) claimed that "the CRFs method does a better job of parameter estimation for the same feature set, and is parallelizable, so that each pass over the training set can require just a fraction of the computation time of the perceptron method". Recently, there are many successful applications of CRFs including Shallow Parsing (Sha and Pereira 2003) and Named Entity Recognition (McCallum and Li 2003).

However, using CRFs for CS is not completely simple because there are three weaknesses for applying CRFs:

- The long distance dependence between a start position (S) and an end position (E) of a clause. It is difficult for the conventional CRFs to deal with the problem of long distance dependence between S and E. We see the example in Section 3, which shows that the distance from the start position to the end position of the clause is long.
- Balancing between a number of start word positions and those of end word positions of clauses in a sentence.
- The clauses can be embedded in the outer clauses.

To overcome the drawbacks mentioned above, we use N-best list by adapting the Joint-CRFs (Shi and Wang 2007), and simultaneously we use rich linguistic information and propose a new bottom-up dynamic algorithm for decoding. The experiments show that our results are competitive with the previous results. Especially, the precision of our results performs better than that of the previous methods. Additionally, with decoding process, our system is also more than approximately 50 times faster than that of (Carreras and Marquez 2005) written in Perl.

The rest of this paper is structured as follows. Section 2 reviews related work. Section 3 formulates the Clause splitting problem. Section 4 briefly introduces linear chain CRFs and Joint-CRFs and how to apply them to Clause Splitting. Section 5 describes and discusses the experimental results. Finally, conclusions are given in Section 6.

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

2 Related Work

Many supervised methods have been developed for Clause Splitting. (Carreras and Marquez 2005) used a discriminative model approach for it. They applied a global learning algorithm, FR-Perceptron (Collins 2002) to recognize structure of clauses. They divided the problem into two layers of local subproblems: a filtering layer, which reduces the search space by identifying plausible clause candidates; and a ranking layer, which builds the optimal clause structure by discriminating among competing clauses. A recognition-based feedback rule is presented, which reflects to each local function its committed error from a global point of view, and follow to train them together online as perceptrons. As a result, the learned function automatically behaves as a filter and ranker, rather than as a binary classifier. The FR-Perceptron method shows the best result for Clause Splitting now.

(Carreras, Marquez, Punyakanok, and Roth 2002) applied the Adaboost algorithm (Carreras and Marquez 2001). They improved Clause Identification by using global inference on the top of the outcome clauses hierarchically learned by local classifiers. Other approaches such as Maximum Entropy, and Winnow are applied for CS too (Hachey 2002).

A number of different methods for the supervised learning approach were used for the CoNLL-2001 shared task (Sang and Dejean 2001). These methods include boosting decision trees and decision graphs, neural networks, memory-based learning, statistical, and symbolic learning (Carreras and Marquez 2001; Hammerton 2001; Tjong Kim Sang 2001).

3 Clause Splitting Problem

At a deeper level of partial parsing is clause splitting. A clause is a sequence of words in a sentence and is a grammatical unit that includes, at minimum, a predicate and an explicit or implied subject, and expresses a proposition. For example: given an input sentence:

```
Coach them in handling complaints so that they can resolve problems
immediately
```

The problem is to split a sentence into clauses as follows:

```
(Coach them in (handling complaints) (so that (they can resolve problems
immediately)))
```

The problem is more difficult than simply detecting non-recursive phrases in sentences. Clause Splitting is divided into three parts: identifying clause starts, identifying clause ends, and finding complete clauses (Sang and Dejean 2001).

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

Formulation

Let X be a sentence space, and Y be a clause space. We can consider a model for finding clauses as a function $R : X \mapsto Y$ which, given a sentence x, identifies the set of clauses $y \subset Y$ of $x \in X$. First, we assume a filter function F which, given a sentence x consisting of a sequence of n words $(x_1, x_2, \ldots, x_n)$, identifies a set of candidate clauses, $F(x) \subseteq \mathcal{P}$ where $\mathcal{P}$ is the set of all possible clauses. A candidate clause is represented as (s, e) for the sentence x where (s, e) is the sequence of consecutive words from word x_s to word x_e, Second, we assume a *score* function which, given a clause, produces a real-value prediction of the clause. We identify a set of clauses for a sentence according to the following optimality criterion:

$$C(x) = \text{argmax}_{y \subseteq F(x)} \sum_{(s,e)_k \in y} score((s, e)_k, x, y) \quad (1)$$

in which $C(x)$ is a set of clauses for a sentence x, and $(s, e)_k$ is a *k-th* clause in y.

We will identify the clause starts (Task 1) and the clause ends (Task 2) to predict a set of candidate clauses for finding complete clauses (Task 3).

4 Applying CRFs to Clause Splitting

In this section, we show how to overcome the drawbacks of applying CRFs and Joint-CRFs to CS as mentioned in the introduction. First, we present an overview of the CRFs and Joint-CRFs models, we then propose a decoding algorithm as well as exploiting rich linguistic information to deal with the problem when applying CRFs and Joint-CRFs to CS.

4.1 Conditional Random Fields

Conditional Random Fields (CRFs) (Lafferty et al. 2001) are undirected graphical models used to calculate the conditional probability of values on designated output nodes, given values assigned to other designated input nodes for data sequences. CRFs make a first-order Markov independence assumption among output nodes, and thus correspond to finite state machine (FSMs).

Let $\boldsymbol{o} = (o_1, o_2, \ldots, o_T)$ be some observed input data sequence, such as a sequence of words in a text (values on T input nodes of the graphical model). Let $\boldsymbol{S}$ be a finite set of FSM states, each is associated with a label l such as a clause start position. Let $\boldsymbol{s} = (s_1, s_2, \ldots, s_T)$ be some sequences of states (values on T output nodes). CRFs define the conditional probability of a state sequence given an input sequence to be

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

$$P_\Lambda(s|o) = \frac{1}{Z_o} \exp\left(\sum_{t=1}^{T} F(s,o,t)\right) \tag{2}$$

where $Z_o = \sum_s \exp\left(\sum_{t=1}^{T} F(s,o,t)\right)$ is a normalization factor over all state sequences. We denote δ to be the Kronecker-δ. Let $F(s,o,t)$ be the sum of CRFs features at time position t:

$$\sum_i \lambda_i f_i(s_{t-1}, s_t, t) + \sum_j \lambda_j g_j(o, s_t, t) \tag{3}$$

where $f_i(s_{t-1}, s_t, t) = \delta(s_{t-1}, l^{'})\delta(s_t, l)$ is a *transition* feature function which represents sequential dependencies by combining the label $l^{'}$ of the previous state s_{t-1} and the label l of the current state s_t, such as the previous label $l^{'}$ = AV (adverb) and the current label l = JJ (adjective). $g_j(o, s_t, t) = \delta(s_t, l)x_k(o, t)$ is a *per-state* feature function which combines the label l of current state s_t and a context predicate, i.e., the binary function $x_k(o, t)$ that captures a particular property of the observation sequence o at time position t. For instance, the current label is JJ and the current word is *"conditional"*.

Training CRFs

Let $\Lambda = \{\lambda_i, \lambda_j\}$ be the set of weights in a CRFs model. Λ is set to maximize the conditional log-likelihood of state sequences in some training set, $D = \{\langle o, s\rangle^{(1)}, \ldots, \langle o, s\rangle^{(N)}\}$:

$$L_\Lambda = \sum_{j=1}^{N} log\left(p_\Lambda(s^{(j)}|o^{(j)})\right) - \sum_k \frac{\lambda_k^2}{2\sigma^2} \tag{4}$$

where the second sum is a Gaussian prior over parameters (with variance σ^2) which provides smoothing to avoid overfitting in the training data.

When the training labels make the state sequence unambiguous, the likelihood function in exponential models such as CRFs is convex, and finding the global optimum is guaranteed. Parameter estimation of a CRFs model requires an iterative procedure. Currently, various methods can be used to optimize L_Λ, including Iterative Scaling algorithms such as GIS and IIS (Lafferty et al. 2001), and quasi-Newton methods such as L-BFGS (Sha and Pereira 2003). Among these methods, L-BFGS is the most efficient (Malouf 2002; Sha and Pereira 2003).

L-BFGS requires only that one provides the first-derivative of the function to be optimized. Let $s^{(j)}$ denote the state path of training sequence j, and then the first-derivative of the log-likelihood is

$$\frac{\delta L_\Lambda}{\delta \lambda_k} = \left(\sum_{j=1}^{N} C_k(s^{(j)}, o^{(j)})\right) - \left(\sum_{j=1}^{N} \sum_s p_\Lambda(s|o^{(j)}) C_k(s, o^{(j)})\right) - \frac{\lambda_k}{\sigma^2} \tag{5}$$

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

in which $C_k(s, o)$ is the count of feature f_k, given s and o. The first two terms correspond to the difference between the empirical and the model expected values of feature f_k. The last term is the first-derivative of the Gaussian prior.

Inference in CRFs

Given the conditional probability of the state sequence defined in (2) and set of the parameters $\Lambda = \{\lambda, \dots\}$, inference in CRFs is to find the most likely state sequence s^* subject to:

$$\boldsymbol{s}^* = \text{argmax}_s p_\Lambda(s|o) = \text{argmax}_s \text{exp}\left(\sum_{t=1}^{T} F(s, o, t)\right)$$

We can efficiently calculate $\boldsymbol{s}^*$ with the Viterbi algorithm (Rabiner 1989). For Viterbi algorithm, we use the table for storing the probability of the most likely path up to time t, which accounts for the first t observations and ends in state s_i. We define this probability to be $\varphi_t(s_i)$ $(0 \leq t \leq T-1)$, $(s_i \in \mathbf{S})$, where $\varphi_0(s_i)$ is the probability of starting in each state s_i. We are given a recursive formulation as follows:

$$\varphi_{t+1}(s_i) = max_{s_j}\{\varphi_t(s_j)\text{exp}(F(s, o, t))\} \tag{6}$$

where $s_j \in \boldsymbol{S}$. The formula (6) terminates in the most likely state ${s_i}^*$ where ${s_i}^* = \text{argmax}_{s_i}[\varphi_T(s_i)]$. From ${s_i}^*$, we can backtrack through the dynamic programming table to recover $\boldsymbol{s}^*$.

4.2 Joint Conditional Random Fields

There is the limitation of applying CRFs to three sub problems: If we process Task 1, Task 2, and Task 3 separately then errors in processing nearly always cascade through chain, causing errors in the final output. To tackle this limitation, we introduce the use of Joint-CRFs of Task 1, Task 2, and Task 3. Our Joint-CRFs models is based on the Dual-layer Conditional Random Fields developed by (Shi and Wang 2007) for segmentation and tagger. We combine three subproblems: Task 1, Task 2, and Task 3 using the joint probability model with Joint-CRFs.

Let $W = \{W_1, W_2, \dots, W_n\}$ denote the observed sentence where W_i is the *i-th* word in the sentence, $S = \{S_1, S_2, \dots, S_k\}$ denotes a label of Task 1 where $S_i \in$\{a start position word (S), or a word which is not a start position word (*)\}, $E = \{E_1, E_2, \dots, E_m\}$ denotes a label of Task 2 where $E_i \in$\{an end position word (E), or a word which is not an end position word (*)\}, $C = \{C_1, C_2, \dots, C_m\}$ denote a label of a clause where $C_i \in$ \{the named clause labels for Task 3\} (we can see the example in Section 5 in more detail). Our goal is to identify a start word of clause, an end word of clause and a boundary label of a clause that maximize the joint probability $P(S, E, C|W)$. We can formulate the joint problem as follows:

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

$$\begin{aligned}
\langle (S,E)^*, C^* \rangle &= \arg\max_{S,E,C} P(S,E,C|W) \\
&= \arg\max_{S,E,C} P(C|(S,E),W)P(S,E|W) \qquad (7) \\
&\approx \arg\max_{S,E,C} P(C|S_1,S_2,\ldots,S_n,E_1,E_2,\ldots,E_m)P(S,E|W) \\
&= \arg\max_{S,E,C} P(C|Identify(S,E,W))P(S,E|W) \qquad (8) \\
&\approx \arg\max_{S,E,C} P(C|Identify(S,E,W))P(S|W)P(E|W) \qquad (9)
\end{aligned}$$

where $(S,E)^*$ and C^* is the most likely (boundary label at the start word, boundary label at the end word) and a boundary label of a clause, respectively, $\mathit{Identify}(S,E,W) = \{S_1, S_2, \ldots, S_n, E_1, E_2, \ldots, E_m\}$ is a set of the result of Task 1 and Task 2.

Applying Bayes's theorem, the above joint probability $P(S,E,C|W)$ is factorized into two terms, $P(C|(S,E),W)$ and $P(S,E|W)$. The first term represents the conditional probability of Task 3, given the result of Task 1 and Task 2 ($\mathit{Identify}(S,E,W)$), the second term represents the conditional probability of Task 1 and Task 2 given W. Note $P(S,E|W) \approx P(S|W)P(E|W)$, assuming that identifying a start word of clause (S) and identifying an end of word (E) of clause is independent together, in which $P(S|W)$ and $P(E|W)$ are the conditional probability of Task 1 given W and the conditional probability of Task 2 given W, respectively.

In training, the probability $P(S,E,C|W)$ can be rewritten (according to formula 2) as:

$$\begin{aligned}
P(S,E,C|W) &\approx P(C|Identify(S,E,W))P(S|W)P(E|W) \\
&= \frac{1}{Z_o(C)}\frac{1}{Z_o(S)}\frac{1}{Z_o(E)} \\
&\quad \exp\left(\sum_{t=1}^{T} F_1(s_1,o_1,t)\right)\exp\left(\sum_{t=1}^{T} F_2(s_2,o_2,t)\right)\exp\left(\sum_{t=1}^{T} F_3(s_3,o_3,t)\right) \qquad (10)
\end{aligned}$$

where F_1, F_2 and F_3 are the sum of CRFs features of Task 1, Task 2 and Task 3, respectively and $Z_o(C)$, $Z_o(S)$ and $Z_o(E)$ are the normalizing term of the probability $P(C|Identify(S,E,W))$, $P(S|W)$ and $P(E|W)$, respectively. Their properties and functions are the same as common CRFs described in 4.1.

We can consider the learning process into two steps: one for learning the first layer of Task 1 (S) and Task 2 (E), and one for learning the second layer of Task 3.

N-best List Approximation for Decoding

Adopting (Shi and Wang 2007), we also use a N-best list approximation method. We limit our reranking targets to the N-best list $\Psi = \{S_1, E_1, S_2, E_2, \ldots, S_N, E_N\}$, in which $\Psi =$

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

$\{S_1, E_1, S_2, E_2, \ldots, S_N, E_N\}$ is ranked by the probability $P(S|W)$ and $P(E|W)$. Therefore, maximum of the joint probability $P(S, E, C|W)$ can be defined approximately:

$$\begin{aligned}\langle (S,E)^*, C^* \rangle &= \arg\max_{S,E,C} P(S,E,C|W) \\ &\approx \arg\max_{(S,E)\in\Psi,C} P(S,E,C|W) \qquad (11) \\ &\approx \arg\max_{(S,E)\in\Psi,C} P(C|Identify(S,E,W))P(S|W)P(E|W) \qquad (12)\end{aligned}$$

We obtain the N-best list of Task 1 (S) and Task 2 (E) and their corresponding probabilities $P(S|W)$ and $P(E|W)$ $(S, E \in \Psi)$ by using a combination of forward Viterbi and backward A* search. Given a particular S and E, the most clause boundaries and its probability $P(C|identify(S, E, W))$ can be calculated by the Viterbi algorithm in section 4.1.

4.3 Features

The set of features we use is the same as that of features reported in (Carreras and Marquez 2005). The set of features includes features at word level and features at sentence level.

Features at word level

The features are used with a window representation of size 2. For a window centered at the the word x_t, we use the following features extracted from $(x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2})$. Where x can be:

- Word form (w) and POS tag (p).
- Chunking tag (c).
- Count: the number of a particular linguistic element which appear in a sentence fragment. We consider two fragments of a sentence, with separate features for each: from the beginning of the sentence to w_i (CountBegin), and from w_i to the end (CountEnd). The linguistic elements are enumerated as follows:
  - Relative pronouns (e.g “that”, “where”, “who”, “which”, “whom”, “whose”)
  - Punctuation marks (. , ; :)
  - Quotes
  - Verb phrase chunks
  - Relative phrase chunks

The feature templates at word level is described in Table 1.

Features at sentence level

These features are used for capturing long-distance dependencies and identifying the clause

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

boundaries of a clause candidate (s, e):

- Top-most structure: A pattern representing the relevant elements of the top-most structure forming the candidate from s to e. The following elements are used to form the pattern:
  - Punctuation marks
  - Coordinate conjunctions (e.g., "and", "or")
  - The word "that"
  - Relative pronouns (e.g., "that", "which", "who", "whom", "whose")
  - Verb phrase chunks
  - The top clause within the $[x_s, \ldots, x_e]$

  where the pattern only considers the top-most structure.[1] We will ignore a clause which appears in the pattern. For example, the pattern for the clause "((to raise)VP rates on containers (carrying U.S. exports to Asia)S about 10%)" is VP-%-S-%.
- The number of clauses found inside the candidate $[x_s, \ldots, x_e]$.

4.4 Decoder for Clause Splitting

As mention in section 1 with three weakness, we do not apply CRFs to Task 3 directly. In this section, we will describe an algorithm for decoding Clause Splitting in a segment of a sentence from l to r. It is a dynamic algorithm presented in Figure 1 as a recursive function. We use results of Task 1 and Task 2 as input of Task 3. Array $mstart[] = [s_1, s_2, \ldots, s_h]$ and $mend[] = [e_1, e_2, \ldots, e_m]$ store results of Task 1 (s_i where $i \in 1, \ldots, h$ is *i-th* start word position of a clause) and Task 2 (e_j where $j \in 1, \ldots, m$ is *j-th* end word position of clause) respectively

Table 1 Feature templates at word level

Transition feature templates	
Current state: s_t	Previous state: s_{t-1}
l	l'
Per-state feature templates	
Current state: s_t	Context predicate: $x(o, t)$
l	$w_{t-2}; w_{t-1}; w_t; w_{t+1}; w_{t+2}$
	$p_{t-2}; p_{t-1}; p_t; p_{t+1}; p_{t+2}$
	$c_{t-2}; c_{t-1}; c_t; c_{t+1}; c_{t+2}$
	CountBegin(w_n)
	CountEnd(w_n)
	where $n \in \{t-2, t-1, t, t+1, t+2\}$

[1] we use the term "top-most structure" according to (Carreras and Marquez 2005)

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

Algorithm 1 FindClause(int l,int r,int i,int j)

```
1: if l > r then
2:    return 0
3: endif
4: if j ≥ 1 then
5:    FindClause(l, mend[j − 1], i, j − 1)
6: endif
7: if i < mstart.size() − 1 then
8:    FindClause(mstart[i + 1], r, i + 1, j)
9: endif
10: k* = argumax_{k∈Γ} score(l,k) + score(k + 1,r)
11: BestClause[l, r]=BestClause[l, k*]∪BestClause[k*+1, r]
12 if score(l,r) ≥ score(l,k*) + score(k* + 1,r) then
13 BestClause[l, r]=BestClause[l, r]∪{(l,r)}
14: endif
```

Fig. 1 Dynamic Algorithm for Clause Splitting

in a segment $(0, n)$; $mstart.size() = h$ and $mend.size() = m$ are the number of mstart array elements and mend array elements respectively. Set $\Gamma = \{s_1, s_2, \ldots, s_h, e_1, e_2, \ldots, e_{m-1}\}$. Bidimensional array BestClause$[l, r]$ stores clauses with an optimal split found in (l, r). Bidimensional array $score[l, r]$ stores the score of the clause candidate (l, r). Recursive function FindClause in Figure 1 includes 4 parameters l, r, i, j, in which i and j are indexes of mstart[] and mend[] respectively. FindClause(l, r, i, j) finds an optimal clause split for the segment (l, r) and stores it in BestClause$[l, r]$. The call to the function FindClause$(0, n, 0, mend.size())$ scans the whole sentence and the optimal clause split for the sentence is stored in BestClause$[0, n]$, in which n is the length of the sentence. What the algorithm does is roughly interpreted as follows:

Let l be a start position of a candidate clause and r be an end position of it. Such positions are based on Task 1 and Task 2. The algorithm picks up all candidate clauses and find the optimal split position k between l and r. The optimal split position is calculated using the score function described below. All candidate clauses (l, r) are checked using the recursive function FindClause(l, r, i, j), where i represents information about possible right positions to l and j possible left position to r. The algorithm starts from the longest segment (an input sentence) (0, n), and narrow down the segment using the parameter i and j.

In the Figure 1, beginning from line 4 to line 9 of the function uses two recursive calls on the sentence segment to enumerate all clause candidates (s_i, e_j) ($s_i \in$ mstart[], $e_j \in$ mend[]) of segment (l, r). Line 10 of the function finds the optimal split k^* for the current sentence segment. The line 11 will assign the union of two disjoint splits BestClause[l, k^*] and BestClause[k^*+1, r] which covers the segment (l, r) to BestClause[l,r] of (l, r) segment. The line 12 and 13 treat the

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

case that a clause (l, r) is added to BestClause$[l, r]$.

A sentence requires a function call for each clause candidate and there is a quadratic number of clause candidates over a number of start words and end words in the sentence. The function consumes a linear time for selecting the optimal split plus the cost of the scoring function. Consequently, computation time of identifying a clause split in a sentence is $O(n^2(n + cost(score)))$ where n is a number of start words and end words in a sentence. Because n is so small, computation time of CS is consumed essentially by computation time of Viterbi algorithm calculating $cost(score)$.

Scoring

It is essential that we identify the score of a candidate clause. We use the Viterbi algorithm in the decoding process for Task 3. Denote Ω as a set of boundary labels of clauses in the outputs which Viterbi algorithm produces to predict labels of clauses in (l, r) segment. The score of a candidate clause (l, r) is defined as follows:

$$\text{score}(l, r) = \sum_{s_k \in \Omega} \varphi_T(s_k) \tag{13}$$

in which $\varphi_T(s_k)$ is that of (6).

We can smooth the score(l, r) of a candidate clause (l, r) using some linguistic elements of clause candidate (l, r):

- verb phrase chunks: n_1
- Punctuation marks: n_2
- Coordinate conjunctions (e.g "or", "and"): n_3
- Relative pronouns (e.g "that", "which", "whose", "who", "whom"): n_4

Finally, we define score(l, r) below:

$$\text{score}(l, r) = \sum_{s_k \in \Omega} \varphi_T(s_k) + \sum_{i=1}^{4} \text{count}(n_i) \tag{14}$$

in which count(n_i) is the number of n_i in clause candidate (l, r).

5 Experiments

We conducted the experiments and evaluated the results with our CRFs framework. We used the Penn Treebank which is used in the CoNLL 2001 shared task[2] (Sang and Dejean 2001)

[2] Data sets are available at http://www.cnts.ua.ac.be/conll2001/clauses/

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

as data for training and testing the clause splitting. WSJ sections from 15 to 18 were used as training data (8,936 sentences), section 20 as development data (2,012 sentences), and section 21 as test data (1,671 sentences). The data of the CoNLL 2001 shared task includes sentences with words, the clause split solutions, POS labels and chunks labels. The data files contain four column separated by a blank space. Each token (a word or a punctuation mark) is put on a separate line and there is an empty line after each sentence. The first item on each line is a token, the second is the part-of-speech tag of the token, the third is a chunking tag of the token, and the fourth is the named clause label. For Task 1, the label of each token defines whether the token is not a start position word of a clause (*), or a start position word of a clause (S). For Task 2, the label of each token defines whether the token is not an end position word of a clause (*), or an end position word of a clause (E). For Task 3, the label of each token defines whether the token is not a boundary label of a clause (*), or a boundary label of a clause {(S*, *S), *S)S)S), ...}. The clause labels of the example in section 3 is described in Table 2. In our system, we used CRF++ (V0.44)[3] to implement the CRFs framework.

We evaluated clause splitting based on the standard measures which are widely used in Information Retrieve (Rijsbergen 1986): precision (p) - the proportion of correctly recognized clauses in output, recall (r) - the proportion of correctly recognized clauses in correct clauses and their harmonic mean F_1. Let $|\ .\ |$ be the number of elements in a set. The computation of the evaluation in a test set including k elements $\{(x^i, y^i)\}_1^k$ can be formulated below:

Table 2 Labels of Task 1, Task 2 and Task 3. The first and second columns show labels of Task 1 and labels of Task 2, respectively. The third columns shows labels of Task 3

Word	Start(Task 1)	End(Task 2)	Task 3
Coach	S	*	(S*
them	*	*	*
in	*	*	*
handling	S	*	(S*
complaints	*	E	*S)
so	S	*	(S*
that	*	*	*
they	S	*	(S*
can	*	*	*
resolve	*	*	*
problem	*	*	*
immediately	*	E	*S)S)S)

[3] CRF++ is available at http://chasen.org/~taku/software/CRF++/

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

$$\mathrm{p} = \frac{\sum_{i=1}^{k} |y^i \cap R(x^i)|}{\sum_{i=1}^{k} |R(x^i)|} \quad \mathrm{r} = \frac{\sum_{i=1}^{k} |y^i \cap R(x^i)|}{\sum_{i=1}^{k} |y^i|} \quad \mathrm{F}_1 = \frac{2pr}{p+r}$$

where $R(x^i)$ is a set of clauses that are identified for a sentence x^i.

For Task 1 and Task 2, we used the framework CRFs with the set of features in section 4.3 as unigram feature templates. We also used some constraints for Viterbi algorithm in the formula (6) as follows:

- Start position of a clause must be the boundary of a chunk.
- End position of a clause must be the boundary of a chunk.

We combined outputs of Task 1 and Task 2 with chunking tag respectively to enrich the dependence of its linguistic information. This combining is described in Table 3. The results of Task 1 and Task 2 are shown in Table 4.

We experimented on Task 1 and Task 2 with a set of features as bigram feature templates. The results of Task 1 and Task 2 are shown in Table 5. They show that F1 value of Task 1 for the test set improves 0.61% and F1 value of Task 2 for the test set improves 0.94%.

We also experimented Task 1, Task 2 and Task 3 using Joint-CRFs with a set of features as

Table 3 Integrating Output tag of Task 1 and Task 2 with chunking tag. The first and second columns show words and POS tags, respectively. The Chunking tag are shows in the third column, in BIO notation. The forth and fifth columns show the outputs of Task 1 and Task 2, respectively. The sixth and seventh columns annotate combining outputs of Task 1 and Task 2 with chunking tag, respectively

Word	Tagger	Chunking tag	Start (Task 1)	End (Task 2)	Start (Task 1)-Chunk	End (Task 2)-Chunk
Interactive	JJ	B-NP	S	*	S-B-NP	*-B-NP
Telephone	NN	IN-P	*	*	*-I-NP	*-I-NP
Technology	NN	I-NP	*	*	*-I-NP	*-I-NP
...	...	...	...	...	...	...
Possibilities	NNS	I-NP	*	E	*-I-NP	E-I-NP
.	.	O	*	E	*-O	E-O

Table 4 Task 1 and Task 2 results (unigram)

Category	Precision	Recall	F1
Testa1 (Dev)	96.14%	92.63%	94.35%
Testb1	94.87%	91.56%	93.19%
Testa2 (Dev)	91.31%	88.09%	89.67%
Testb2	90.29%	87.93%	89.09%

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

bigram feature templates. We chose N = 10 for using in the N-best list. Our results are shown on Table 5. The Joint-CRFs method shows 0.22 %and 0.21 % improvement in F1 of Task 1 and Task 2, respectively.

5.1 Using CRFs and Joint-CRFs to predict score

We used CRFs and Joint-CRFs (joint with Task 1, and Task 2) with the bigram feature templates for Task 3 presented in Section 4.3. Then we used the formula (6) for Viterbi algorithm to count score(l, r) of clause candidate (l, r) segment using the score function (13). The result of Task 3 (identifying the clauses) is shown in Table 6. The F1 performance of Task 3 using Joint-CRFs improves by 0.31% compared with that of Task 3 using CRFs.

5.2 Combining linguistic information

We improved F1 value of Task 3 by using linguistic information for smoothing the score function presented in Section 4.4 and the score function is defined as the formula (14). The result is showed in Table 7. The F1 performances are 84.09% and 84.66%, which are improved by 1.25% and 1.51% compared with the case of using the formula (13), respectively. The Table

Table 5 Task 1 and Task 2 results (unigram + bigram)

Category	CRFs			Joint-CRFs		
	Precision	Recall	F1	Precision	Recall	F1
Testa1 (Dev)	96.67%	92.85%	94.72%	96.83%	92.97%	94.86%
Testb1	95.23%	92.42%	93.80%	95.51 %	92.58%	94.02%
Testa2 (Dev)	92.54%	88.95%	90.71%	92.80%	89.16 %	90.94%
Testb2	91.55%	88.56%	90.03%	91.69%	88.02%	90.23%

Table 6 Task 3 results (unigram + bigram)

Category	CRFs			Joint-CRFs		
	Precision	Recall	F1	Precision	Recall	F1
Testa3 (Dev)	87.62%	80.04%	83.66%	88.28%	80.16%	84.03%
Testb3	87.97%	78.45%	82.84%	88.35%	78.53%	83.15%

Table 7 Task 3 results (bigram + adding linguistic information)

Category	CRFs			Joint-CRFs		
	Precision	Recall	F1	Precision	Recall	F1
Testa3 (Dev)	89.07%	80.30%	84.46%	90.01%	80.47%	84.97%
Testb3	90.01%	78.98%	84.09%	91.03%	79.13%	84.66%

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

7 also shows that the F1 performance using Joint-CRFs outperforms 0.57 % higher than using CRFs.

Table 8 shows a comparison of our methods with the previous works on the same training and testing data. The results also show that our method is comparable to that of (Carreras and Marquez 2005), which is the state-of-the art result, and outperformed other methods. We see that the result of (Carreras and Marquez 2005) outperforms slightly our result because they combine the three tasks of CS together with end to end while we combine the three task with an intermediary role. With the error-driven method, they feedback errors on training process. However, our method shows precision improves that of other methods. This is very useful when we apply CS for other applications such as machine translation because the clauses need to be identified correctly.

We carried out statistical significance tests using the t-test. Pairwise t-test showed that the precision of our results is significantly better than that of the result of (Carreras and Marquez 2005) under the significant level 4.98×10^{-6}(p-value).

5.3 Evaluating decoding speed

Table 9 shows the average decoding time of a test set and that per a sentence by two methods our method and Carreras et al. 05 on Intel(R) Xeon(TM) CPU 3.06GHz, 4G RAM machine with Fedora Core 5. The result shows that the computation time of our system improves that of Carreras et al. 05, by the approximate factor of 50, though the experiment of our method was implemented in C++, and that of Carreras et al. 05 in Perl[4].

Table 8 Comparison of our result and previous results

Reference	Technique	Precision	Recall	F1
Carreras et al. 05	FR-Perceptron	88.17%	**82.10%**	**85.03%**
Our method	**Joint-CRFs**	**91.03%**	79.13%	84.66%
Carreras et al. 02	AdaBoost class	90.18%	78.11%	83.71%
Carreras et al. 01	AdaBoost class	84.82%	78.85%	81.73%
Monila and Pla 01	HMM	70.85%	70.51%	70.68%

[4] We also measured the simple task such as QuickSort algorithm for C++ and Perl with 10 random test sets of 1 millions real numbers. The result showed that the computation time of QuickSort algorithm in C++ is faster than that in Perl, by the average factor of 25.

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

Table 9 Comparison of the decoding time

Reference	Decoding time of the test set	Decoding time per sentence
Our system	336 sec	0.20 sec
Carreras et al. 05	16503 sec	9.8 sec

Table 10 Task 3 results (with gold standard result Task 1 & Task 2)

Category	Precision	Recall	F1
Testa3 (Dev) (bigram)	90.34%	80.52%	85.15%
Testa3 (Dev) (bigram + adding linguistic information)	92.96%	80.97%	86.55%
Testb3 (bigram)	90.37%	78.97%	84.28%
Testb3 (bigram + adding linguistic information)	92.19%	80.58%	85.99%

5.4 Relation between performance of Task 3 and results of Task 1 and Task 2

In order to test how the results of Task 1 and Task 2 effect on the performance of Task 3, we conducted an experiment by performing Task 3 using the gold standard data of Task 1 and Task 2. Table 10 shows that F_1 values of the Task 3 are 84.28% and 85.99% with adding linguistic information, respectively. We see that the results of using the gold standard data of Task 1 and Task 2 (85.99%) could improve our results (84.09%). This explains that the performance of Task 1 and Task 2 are important to the result of splitting clause. However, the main errors of the Task 3 are the miss-corresponding of the starting point (the result of Task 1) and the ending point (the result of Task 2). These errors are caused by the inappropriate scores in decoding algorithm. Our future work is focused on how to find a better scoring method for the decoding algorithm.

6 Conclusion

In this paper, we have presented the CRFs-based framework approach for clause splitting. We have proposed a new bottom-up dynamic algorithm for decoding and some effective linguistic information for clause splitting. We compared the results of exploiting our framework to the previous works in the CONLL 2001 shared task. The experiments show that our result is competitive with the state-of-the-art results of clause splitting.

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

Acknowledgment

This study was supported by Japan Advanced Institute of Science and Technology, the 21^{st} Century COE Program: "Verifiable and Evolvable e-Society".

Reference

Abney, Steven(1991). "Parsing by chunks." *In In Robert Berwick, Steven Abney, and Caroll Tenney, editors, Principle-Based Parsing*, pp. 257–278.

Ando, Rie, and Zhang, Tong. (2005). "A highperformance semi-supervised learning method for text chunking." *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1–9.

Carreras, X., and Màrquez, L. (2001). "Boosting trees for clause splitting." *In Proceedings of CoNLL-2001*, pp. 73–75. Toulouse, France.

Carreras X. and , Màrquez, L. (2003). "Phrase recognition by filtering and ranking with perceptrons." *Proceedings of RANLP-2003*, pp. 205–216. Borovets, Bulgaria.

Carreras, X., Màrquez, L., Punyakanok, V. and Roth, D. (2002). "Learning and inference for clause identification." *Proceedings of the 14th European Conference on Machine Learning (ECML 2002)*, pp. 35–47. Finland.

Carreras, X., and Màrquez, L. (2005). "Filtering-ranking perceptron learning for partial parsing." *Machine Learning*, *60* (1), pp. 41–71.

Collins, M. (2002). "Discriminative training methods for hidden markov models: Theory and experiments perceptron algorithms." *Proceeding EMNLP-02*, pp. 1–8. Philadelphia, PA, USA.

Hachey, Benjamin C. (2002). "Recognising clauses using symbolic and machine learning approaches." *Masters thesis*. University of Edinburgh.

Hammerton, James. (2001). "Clause identification with Long Short-Term Memory." *In Proceedings of the 5th Conference on Natural Language Learning, CoNLL-2001*, pp. 61–63. Toulouse, France.

Hammerton, J., Osborne, M., Armstrong, S. and Daelemans, W. (2002). "Introduction to the special issue on machine learning approaches to shallow parsing." *Journal of Machine Learning Research*, (2), pp. 551–558.

Kudo, T. and Matsumoto, Y. (2001). "Chunking with support vector machines." *Proceedings of HLT-NAACL01*, pp. 192–199. Pittsburgh, USA.

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

Lafferty, J., McCallum, A., and Pereira, F. (2001). "Conditional random fields: Probabilistic models for segmenting and labeling sequence data." *Proc. 18th International Conference on Machine Learning*, pp. 282–289. Morgan Kaufmann, San Francisco, USA.

Malouf, R. (2002). "A comparison of algorithms for maximum entropy parameter estimation." *Proceedings of CoNLL'02*, pp. 49–55. Taipei, Taiwan.

McCallum, A., and Li, W. (2003). "Early results for named entity recognition with conditional random fields, feature induction and webenhanced lexicons." *Proceedings of CoNLL- 2003*, pp. 188–191. Edmonton, Canada.

Nguyen, V. V., Nguyen, L. M., and Shimazu, A. (2007). "Using Conditional Random Fields for Clause Splitting." *In Proceedings of Pacling-07*, pp. 58–65.

Rabiner, L. (1989). "A tutorial on hidden markov models and selected applications in speech recognition." *Proc. of IEEE, volume 77*, pp. 257–286.

Rijsbergen, V. (1986). "Information Retrieval." *Text Book*.

Roark, B., Saraclar, M., Collins, M., and Johnson, M. (2004). "Discriminative language modeling with conditional random fields and the perceptron algorithm." *Proceedings of ACL 04*, pp. 47–54. USA.

Tjong Kim Sang, Erik F. (2001). "Memory-Based Clause Identification." *In Proceedings of the 5th Conference on Natural Language Learning, CoNLL-2001*, pp. 67–69. Toulouse, France.

Tjong Kim Sang, Erik F., and Déjean, H. (2001). "Introduction to the conll-2001 shared task: Clause identification." *In Proceedings of the 5th Conference on Natural Language Learning, CoNLL-2001*, pp. 53–57. Toulouse, France.

Sha, F., and Pereira, F. (2003). "Shallow parsing with conditional random fields." *Proceedings of HLT-NAACL03*, pp. 213–220.

Shi, Y., and Wang, M. (2007). "A Dual-layer CRFs Based Joint Decoding Method for Cascaded Segmentation and Labeling Tasks." *In Proceedings of IJCAI-07*, pp. 1707–1712.

Vinh Van Nguyen: received the Bachelor degree in Faculty of Information Technology, Vietnam National University of Hanoi (VNUH) in 1998, and Master degree at Hanoi College of Technology (COLTECH) in 2004. From September 1998 to 2005, he worked as a researcher and developer at R & D Department of Lac Viet Computing Corporation, Vietnam. Since April 2006, he has been a PhD student in Natural Language Processing Laboratory, School of Information Science, Japan Advanced Institute of Science and Technology (JAIST).

Information and Media Technologies 4(1) : 57–75 (2009)
reprinted from: Journal of Natural Language Processing 16(1) : 47–65 (2009)

Minh Le Nguyen: received the Bachelor in Information Technology from Hanoi University of Science, and Master degrees in Information Technology from VNUH, in 1998 and 2001, respectively. He received Doctoral degree in School of Information Science, JAIST in 2004. From 2005 to 2007, he was a Post-doctoral Fellow at Natural Language Processing Laboratory, School of Information Science, JAIST. Currently, he is an assistant professor at Natural Language Processing Laboratory, School of Information Science, JAIST. His research interests include Text Summarization, Natural Language Understanding, Machine Translation, and Information Retrieval.

Akira Shimazu: received the Bachelor and Master degrees in mathematics from Kyushu University in 1971 and 1973, respectively, and a Doctoral degree in Natural Language Processing from Kyushu University in 1991. From 1973 to 1997, he worked at Musashino Electrical Communication Laboratories of Nippon Telegram and Telephone Public Corporation, and at Basic Research Laboratories of Nippon Telegraph and Telephone Corporation. From 2002 to 2004, he was the president of the Association for Natural Language Processing. He has been a professor in the Graduate school of Information Science, JAIST since 1997.

(Received February 19, 2008)
(Revised June 26, 2008)
(Accepted August 4, 2008)