A Framework for Procedural Text Understanding

Hirokuni Maeta*
Cybozu, Inc.
2-7-1 Nihombashi, Chuo-ku,
Tokyo, Japan
hirokuni.maeta@gmail.com

Tetsuro Sasada
ACCMS, Kyoto University
Yoshida Honmachi, Sakyo-ku,
Kyoto, Japan
sasada@ar.media.kyoto-u.ac.jp

Shinsuke Mori
ACCMS, Kyoto University
Yoshida Honmachi, Sakyo-ku,
Kyoto, Japan
forest@i.kyoto-u.ac.jp

Abstract

In this paper we propose a framework for procedural text understanding. Procedural texts are relatively clear without modality nor dependence on viewpoints, etc. and have many potential applications in artificial intelligence. Thus they are suitable as the first target of natural language understanding. As our framework we extend parsing technologies to connect important concepts in a text. Our framework first tokenizes the input text, a sequence of sentences, then recognizes important concepts like named entity recognition, and finally connect them like a sentence parser but dealing all the concepts in the text at once. We tested our framework on cooking recipe texts annotated with a directed acyclic graph as their meaning. We present experimental results and evaluate our framework.

1 Introduction

Among many sorts of texts in natural languages, procedural texts are clear and related to the real world. Thus they are suitable for the first target of natural language understanding (NLU). A procedural text is a sequence of sentences describing instructions to create an object or to change an object into a certain state. If a computer understands procedural texts, there are potentially tremendous applications: an intelligent search engine for how-to texts (Wang et al., 2008), more intelligent computer vision (Ramanathan et al., 2013), a work help system teaching the operator what to do the next (Hashimoto et al., 2008), etc.

The general natural language processing (NLP) tries to solve the understanding problem by a long series of sub-problems: word identification, part-of-speech tagging, parsing, semantic analysis, and so on. Contrary to this design, in this paper, we propose a concise framework of NLU focusing on procedural texts. There have been a few attempts at procedural text understanding. Momouchi (1980) tried to convert various procedural texts into so-called PT-chart on the background of automatic programming. Hamada et al. (2000) proposed a method for interpreting cooking instruction texts (recipes) to schedule two or more recipes. Although their definition of understanding was not clear and their approach was based on domain specific heuristic rules, these pioneer works inspired us to tackle a major problem of NLP, text understanding.

As the meaning representation of a procedural text we adopt a flow graph. Its vertices are important concepts consisting of word sequences denoting materials, tools, actions, etc. And its arcs denote relationships among them. It has a special vertex, root, corresponding to the final product. The problem which we try to solve in this paper is to convert a procedural text into the appropriate flow graph. The input of our NLU system is the entire text, but not a single sentence.

Our framework first segments sentences into words (word segmentation; abbreviated to WS hereafter). This process is only needed for some languages without clear word boundary. Then we identify concepts in the texts and classify them into some categories (concept identification; abbreviated to CI hereafter). And finally we connect them with labeled arcs. For the first process, WS, we adapt an existing tool to the target domain and achieve an enough high accuracy. The second process, CI, can be solved by the named entity recognition (NER) technique given an annotated corpus (training data). The major difference is the definition of named entities (NE). Contrary to many other NERs we propose a method that does not

*This work was done when the first author was at Kyoto University.

Proceedings of the 14th International Conference on Parsing Technologies, pages 50–60,
Bilbao, Spain; July 22–24, 2015.

require part-of-speech (POS) tags. This makes our text understanding framework simple. For the final process we extend a graph-based parsing method to deal with the entire text, a sequence of sentences, at once. The difference from sentence parsing is that the vertices are concepts but not words and there are words not covered by any concept functioning as clues for the structure.

As a representative of procedural texts, we selected cooking recipes, because there are many available resources not only in the NLP area but in the computer vision (CV) area. For example, the TACoS dataset (Regneri et al., 2013), is a collection of short videos recording fundamental actions in cooking with descriptions written by Amazon Mechanical Turk. Another example, the KUSK dataset (Hashimoto et al., 2014), contains 40 videos recording entire executions (20 recipes by two persons). The recipes in the KUSK dataset are taken from the r-FG corpus (Mori et al., 2014), in which each recipe text is annotated with its "meaning."

We tested our framework on recipe texts manually annotated with word boundary information, concepts, and a flow graph. We compare a naive application of an MST dependency parser and our extension for flow graph estimation. We also measure the accuracy at each step with the gold input assuming the perfect preceding steps. Finally we evaluate the full automatic process of building a flow graph from a raw text. Our result can be a solid baseline for future improvement in the procedural text understanding problem.

2 Related Work

Some attempts at procedural text understanding were proposed in the early 80's (Momouchi, 1980). Then Hamada et al. (2000) proposed tree-based representation of cooking instruction texts (recipes) from the application point of view. These approaches used rule-based methods, but they, along with the current success of the machine learning approach, inspired us to conceive that the procedural text understanding can be a tractable problem for the current NLP.

In our framework the procedural text understanding problem is decomposed into three processes. The first process is the well-known WS. There have been many researches reporting high accuracies in various languages based on the corpus-based approach (Merialdo, 1994; Neubig et al., 2011, *inter alia*). The second one is CI, which can be solved in the same way of NER (Chinchor, 1998) with a different definition of named entities. The accuracy of the general NER is less than WS but is more than 90% when a large annotated corpus is available (Sang and Meulder, 2003, *inter alia*). So we can say that CI can also be solved given an annotated corpus. The only open question is how many examples are required to achieve a practically high accuracy. This paper gives a solution to this. The third one is our original text parsing, which outputs a flow graph taking a text and the concepts in it as the input. To solve this problem, we follow the idea of the graph-based dependency parsing (McDonald et al., 2006; McDonald et al., 2005). Dependency parsing attempts to connect *all the words* in an input sentence with labeled arcs to form a rooted tree. In our method, the units are concepts instead of words and the input is an entire text (a sequence of sentences), not a single sentence. The words not forming concepts (mainly function words), are only referred to as features to estimate the flow graph. We also add another module to form a directed acyclic graph (DAG).

From the NLP viewpoint, the major problems we are solving are 1) dependency parsing (Buchholz and Marsi, 2006) among concepts only, 2) predicate-argument structure analysis (Taira et al., 2010; Yoshino et al., 2013), 3) semantic parsing (Wong and Mooney, 2007; Zettlemoyer and Collins, 2005), and 4) coreference, anaphora, and ellipsis resolution (Nielsen, 2004; Fernández et al., 2004). For dependency parsing we resolve the target of modifiers such as quantities, durations, timing clauses. For predicate-argument structure analysis, we figure out which action is applied to what object with what tools, even if it is stated in passive form or just by a past participle. For semantic parsing we resolve the relationships between concepts. For coreference, anaphora, and ellipsis resolution, our DAG constructor links an action to another action that takes the result of the former action or an abstract expression to a concrete intermediate product. Our method solves these problems focusing on important notions at once.

The understanding of procedural texts may allow a more sophisticated combination of NLP an CV. Recently there have been some attempts at aligning videos and natural language descriptions

1. 両手 鍋$_{T}$ で 油$_{F}$ を 熱$_{Ac}$ する 。
(In a Dutch oven$_{T}$, heat$_{Ac}$ oil$_{F}$.)
セロリ$_{F}$ と 緑 玉ねぎ$_{F}$ と ニンニク$_{F}$ を 加え$_{Ac}$ る 。
(Add$_{Ac}$ celery$_{F}$, green onions$_{F}$, and garlic$_{F}$.)
1 分 ほど$_{D}$ 炒め$_{Ac}$ る 。
(Cook$_{Ac}$ for about 1 minute$_{D}$.)
2. ブイヨン$_{F}$ と 水$_{F}$ と マカロニ$_{F}$ と 胡椒$_{F}$ を 加え$_{Ac}$ て 、 パスタ$_{F}$ が 柔らか$_{Sf}$ く な$_{Af}$ る まで 煮$_{Ac}$ る 。
(Add$_{Ac}$ broth$_{F}$, the water$_{F}$, macaroni$_{F}$, and pepper$_{F}$, and simmer$_{Ac}$ until the pasta$_{F}$ is$_{Af}$ tender$_{Sf}$.)
3. 刻$_{Ac}$ ん だ セージ$_{F}$ を まぶ$_{Ac}$ す 。
(Sprinkle$_{Ac}$ the snipped$_{Ac}$ sage$_{F}$.)

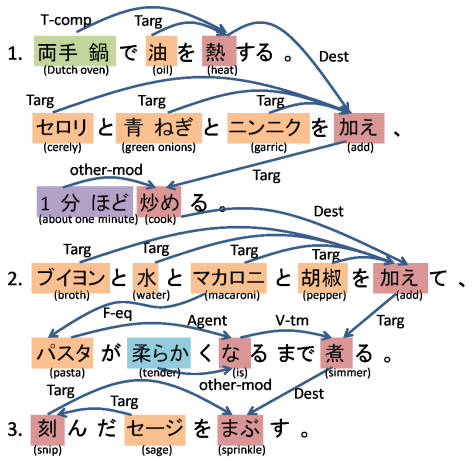

Figure 1: Examples of a procedural text and its flow graph.

(Naim et al., 2014; Rohrbach et al., 2013). In these researches, the NLP part is very naive. They just identify the nouns in the text and apply a sequence-based alignment tool. Now the machine translation community is shifting to the tree-based approach to capture structural differences in two languages. The flow graph representation enables grounding of tuples consisting of an action and its target objects, and also absorbs the difference in the execution order of a procedural text and the video recording its execution.

Although NLU is the major scientific problem of AI, procedural text understanding is important from the viewpoint of applications as well. For cooking recipes for example, on which we test our framework in this paper, we can realize a more intelligent search engine, summarization, or a help system (Wang et al., 2008; Yamakata et al., 2013; Hashimoto et al., 2008).

3 Recipe Flow Graph Corpus

As a test bed of the text parsing problem, we adopt the recipe flow graph corpus (r-FG corpus) (Mori et al., 2014). To our best knowledge, this is the only corpus annotated with flow graphs that matches with our requirements. In addition cooking recipes are representative procedural texts describing very familiar activities of our daily life, and its meaning representation has various applications. Our framework is, however, not limited to this corpus.

3.1 r-FG Corpus

The r-FG corpus contains randomly crawled recipes in Japanese from a famous Internet recipe site.[1] The specification of the corpus is shown in Table 1. The text part of a recipe consists of a sequence of steps and the steps have some sentences. All the concepts (entities and actions) appearing in the sentences are identified and annotated with a concept tag.[2] The text part is annotated with a rooted DAG representing its meaning as shown in Figure 1.

#recipes	#sentences	#NEs	#words
200	1,303	7,268	25,446

Table 1: Corpus statistics.

3.2 Vertices

Each vertex of a flow graph corresponds to a concept represented by a word sequence in the text and a concept type such as food, tool, action. Table 2 lists the concept types along with the average number of occurrences per recipe. There is one special vertex, root, corresponding to the final dish. In the Figure 1 example, the node of "splinkle" is the root.

3.3 Arcs

An arc between two vertices indicates that they have a certain relationship. An arc has a label denoting its relationship type. Table 3 lists the arc types with their average frequencies per recipe. The most interesting relationships may be coreferences and null-instantiated arguments. In Fig-

[1] http://cookpad.com (accessed on 2015 May 19)

[2] In the original r-FG paper (Mori et al., 2014), they call the concepts "recipe named entities." In this paper we use the term "concept" to refer to them, because the recipe named entities contain verb phrases.

Concept tag	Meaning	Freq.
F	Food	11.87
T	Tool	3.83
D	Duration	0.67
Q	Quantity	0.79
Ac	Action by the chef	13.83
Af	Action by foods	2.04
Sf	State of foods	3.02
St	State of tools	0.30
Total	–	36.34

Table 2: Concept tags with frequencies per recipe.

Arc label	Meaning	Freq.
Agent	Action agent	2.15
Targ	Action target	15.67
Dest	Action destination	7.22
F-comp	Food complement	0.65
T-comp	Tool complement	1.32
F-eq	Food equality	3.15
F-part-of	Food part-of	2.37
F-set	Food set	0.15
T-eq	Tool equality	0.44
T-part-of	Tool part-of	0.39
A-eq	Action equality	0.53
V-tm	Head of a clause for timing	1.06
other-mod	Other relationships	3.54
Total	–	38.62

Table 3: Arc labels with frequencies per recipe.

ure 1 for example, “macaroni” is equal to “pasta.” According to the world knowledge, macaroni is a sort of pasta, but in this recipe they are identical. An example of a null-instantiated argument is the relationship between “heat” and “add.” Celery etc. should be added not to the initial cold Dutch oven without oil but to the hot Dutch oven with oil, which is the implicit result of the action “heat.”

4 Overview of Procedural Text Understanding

Our framework of procedural text understanding consists of the following three processes combined in the cascaded manner.

1. Word segmentation (WS)
2. Concept identification (CI)
3. Flow graph estimation

The input of WS is a raw sentence and the output is a word sequence. For example the WS takes the first sentence in Figure 1 without any tag as the input as follows:

両手鍋で油を熱する。

Then WS outputs the following word sequence separated by whitespace as the output.

両手 鍋 で 油 を 熱 する 。

The input of CI is the word sequence, the output of WS, and it identifies concepts, which are spans of words without overlap annotated with its type sequences. For the above example, the CI outputs three concepts as follows:

両手 鍋$_{F}$ で 油$_{F}$ を 熱$_{Ac}$ する 。

This part is similar to NER. Contrary to a normal NER, however, our method does not require POS tag for the words in the input. Thus we do not need to adapt a POS tagger to the target domain. For English or other languages with obvious word boundary, we can start from CI.

Now we have a text consisting of some sentences with concepts identified. An example is the left hand side of Figure 1. This is the input of the flow graph estimation step and the output is a flow graph as show on the right hand side of Figure 1 for example.

In the traditional NLP approach, many subproblems proceed after NER. Syntactic parsing clarifies the intra-sentential relationships among NEs, then anaphora/coreference resolution figures out their inter-sentential relationships. Contrary, we process the entire text at once. In the subsequent section, we describe the above three process in detail.

5 Word Segmentation

Some languages such as Japanese or Chinese, have no obvious word boundary like whitespace in English. The first step of our framework is WS. For many European languages this process is almost obvious and instead of WS we only need to decompose some special words like “isn’t” to “is” + “not” in English or “du” to “de” + “le” in French.

For WS we adopt the pointwise method (Neubig et al., 2011) because of its flexibility for language resource addition.[3] This characteristics is suitable especially for domain adaptation. Below we explain pointwise WS briefly and our method to improve its accuracy for user generated recipes.

[3] An implementation and the default model for the general domain are available from `http://www.phontron.com/kytea/` (accessed on 2015 May 19).

Type	Feature setting
Character n-gram	x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2}, x_{i+3}, $x_{i-2}x_{i-1}$, $x_{i-1}x_i$, x_ix_{i+1}, $x_{i+1}x_{i+2}$, $x_{i+2}x_{i+3}$, $x_{i-2}x_{i-1}x_i$, $x_{i-1}x_ix_{i+1}$, $x_ix_{i+1}x_{i+2}$, $x_{i+1}x_{i+2}x_{i+3}$
Character type n-gram	$c(x_{i-2})$, $c(x_{i-1})$, $c(x_i)$, $c(x_{i+1})$, $c(x_{i+2})$, $c(x_{i+3})$, $c(x_{i-2})c(x_{i-1})$, $c(x_{i-1})c(x_i)$, $c(x_i)c(x_{i+1})$, $c(x_{i+1})c(x_{i+2})$, $c(x_{i+2})c(x_{i+3})$, $c(x_{i-2})c(x_{i-1})c(x_i)$, $c(x_{i-1})c(x_i)c(x_{i+1})$ $c(x_i)c(x_{i+1})c(x_{i+2})$, $c(x_{i+1})c(x_{i+2})c(x_{i+3})$
Dictionary	$d(x_{i-2}x_{i-1}x_ix_{i+1})$, $d(x_{i-1}x_ix_{i+1}x_{i+2})$, $d(x_ix_{i+1}x_{i+2}x_{i+3})$ $L(\cdots x_{i-2}x_{i-1}x_i)$, $R(x_{i+1}x_{i+2}x_{i+3}\cdots)$

Table 4: Features for word segmentation. The fuction $c(\cdot)$ maps a character into one of six character types: symbol, alphabet, arabic, number *hiragana*, *katakana*, and *kanji*. The fuction $d(\cdot)$ returns whether the string is in the dictionary or not. And the functions $L(\cdot)$ and $R(\cdot)$ return whether substrings of any length on the left hand side or right hand side match with a dictionary entry.

5.1 Pointwise Method

The pointwise method formulate WS as a binary classification problem, estimating boundary tags $\boldsymbol{b}_1^{I-1}$. Tag $b_i = 1$ indicates that a word boundary exists between characters x_i and x_{i+1}, while $b_i = 0$ indicates that a word boundary does not exist. This classification problem can be solved by tools in the standard machine learning toolbox such as support vector machines (SVMs).

The features are character n-grams surrounding the decision point i, which are substrings of $x_{i-2}x_{i-1}x_ix_{i+1}x_{i+2}x_{i+3}$, character type n-grams, and whether character n-grams matches an entry in the dictionary or not. Table 4 lists the features.

As we can see, the pointwise WS does not refer to the other decisions, thus we can train it from a partially segmented sentences, in which only some points between characters are annotated with word boundary information.

5.2 Domain Adaptation

As the WS adaptation to recipes, we convert the r-FG corpus into partially segmented sentences following (Mori and Neubig, 2014). In the corpus only r-NEs are segmented into words. That is to say, only both edges of the r-NEs and the inside of the r-NEs are annotated with word boundary information. If the r-NE in focus is ホットドッグ composed of two words, then the partially segmented sentences are

ex.) 各 | ホ – ッ – ト | ド – ッ – グ | に␣チ␣リ␣、 …,

ex.) | ホ – ッ – ト | ド – ッ – グ | を␣ア␣ル␣ミ…,

where the symbols "|," "–," and "␣" mean word boundary, no word boundary, and no information, respectively. Then we use the partially annotated sentences which we obtained in this way as an additional language resource to train the model.

Type	Feature setting
Word n-gram	w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}, $w_{i-2}w_{i-1}$, $w_{i-1}w_i$, w_iw_{i+1}, $w_{i+1}w_{i+2}$, $w_{i-2}w_{i-1}w_i$, $w_{i-1}w_iw_{i+1}$, $w_iw_{i+1}w_{i+2}$

Table 5: Features for concept identification.

6 Concept Identification

The second step is the concept identification. The concept in the text parsing problem is a span of words without overlap annotated with its type. Thus the concept identification (CI) can be solved in the same manner as the named entity recognition (NER). NER is a sequence labeling problem and many solutions have been proposed so far (Borthwick, 1999; Sang and Meulder, 2003, *inter alia*).

The standard NER method is based on linear chain conditional random fields (CRFs). In this paper we use an NER which allows a partially annotated corpus as a training data as well as a normal fully annotated corpus (Mori et al., 2012).[4] In the training step this NER estimates the parameters of a classifier based on logistic regression (Fan et al., 2008) from sentences fully (or partially) annotated with NEs (concepts). The features are word n-grams surrounding the word in the focus w_i, Table 5 lists the features.

[4]CRFs are also trainable from a partially annotated corpus (Tsuboi et al., 2008). Recently Sasada et al. (2015) have proposed a hybrid method and reported a higher accuracy than CRFs. We may use it for further improvement.

At run-time, given a word sequence, the classifier enumerates all possible BIO2 tags t_i for each word w_i with their probabilities as follows:

$$P_{LR}(t_i|\boldsymbol{w}^-, w_i, \boldsymbol{w}^+),$$

where $\boldsymbol{w}^-$ and $\boldsymbol{w}^+$ are the word sequences preceding it and following it, respectively. Then this NER searches for the tag sequence of the highest probability satisfying the tag sequence constraints.[5]

7 Parsing an Entire Text

The final step is to build a flow graph. The input is a text whose sentences are segmented into words and all the concepts are identified. We call this part a text parsing. As we mentioned in Section 1, text parsing deals with various language phenomena at once, such as dependency, predicate-argument structure, and anaphora/coreference.

For text parsing we extend an MST parser (McDonald et al., 2005). Since the flow graph is a labeled DAG, we add some labeled arcs to the MST. Below we explain the processes one by one.

7.1 Spanning Tree Estimation

We first build a labeled spanning tree covering all the concepts (vertices) of the input text. Let V be a set of vertices and $\mathcal{G}$ be a set of possible spanning trees on V. We assume that there exists a score function $s(u, v, l)$ which represents the likelihood of making a labeled arc from u to v with label l. Then the maximum spanning tree (MST) can be found as follows:

$$\hat{G} = \mathop{\mathbf{argmax}}_{G \in \mathcal{G}} \sum_{(u,v,l) \in G} s(u, v, l).$$

We solve this problem using the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). We define the score function $s(u, v, l)$ as a probability[6]:

$$s(u, v, l) = \frac{\exp\{\boldsymbol{\Theta} \cdot \boldsymbol{f}(u, v, l)\}}{\sum_{(x,r) \in (V \setminus \{u\}) \times L} \exp\{\boldsymbol{\Theta} \cdot \boldsymbol{f}(u, x, r)\}}.$$

Here L is the arc label set (See Table 3), $\boldsymbol{\Theta}$ is a vector of weight parameters and $\boldsymbol{f}(u, v, l)$ is a function that maps a labeled arc into a feature vector. The score function $s(u, v, l)$ computes the probability of making a labeled arc from u to v with label l referring to their word sequences, concept tags, surrounding words in the original recipe text, and label l. A detailed description is given in Subsection 7.3.

We use a log-linear model (Berger et al., 1996) in order to train the weight parameters $\boldsymbol{\Theta}$. Let $\{(V_t, u_t, v_t, l_t)\}_{t=1}^{T}$ denote a set of T training instances, where V_t is a set of the vertices and (u_t, v_t, l_t) is a gold standard arc with label l in the t-th training instance. Given these training examples, weight parameters are estimated so that they maximize the following likelihood:

$$\sum_{t=1}^{T} \log \frac{\exp(\boldsymbol{\Theta} \cdot \boldsymbol{f}(u_t, v_t, l_t)}{\sum_{(x,r) \in (V_t \setminus \{u_t\}) \times L} \exp(\boldsymbol{\Theta} \cdot \boldsymbol{f}(u_t, x, r))} - \frac{1}{2}\|\boldsymbol{\Theta}\|^2.$$

Because our flow graph is not a tree but a DAG, there can be more than one arcs outgoing from a single vertex. In other words it may contain two arcs, (u, v, l) and (u, v', l'), which share the same start vertex u. In these cases, we add both (V, u, v, l) and (V, u, v', l') to the training data.

7.2 Arc Addition

Given the labeled MST G, we add some labeled arcs to form a flow graph of a labeled DAG with a root. Our algorithm for adding arcs is described in Figure 2.

The most important point is to choose the best arcs one by one among those which do not create cycles and add them to the MST. In Figure 2

1: $G \leftarrow$ Maximum spanning tree of V.
2: $A \leftarrow$ Sequence of arcs that can be added to G without violating the acyclic condition.
3: Sort A in the descending order of the value of the score function s.
4: $n \leftarrow 1$
5: **for** $(u, v) \in A$ **do**
6: **if** $G \cup \{(u, v)\}$ is acyclic and $p(n) < s(u, v)$ **then**
7: $G \leftarrow G \cup \{(u, v)\}$
8: $n \leftarrow n + 1$
9: **end if**
10: **end for**
11: return G

Figure 2: Algorithm of DAG estimation

[5] For example, the tag sequence F-I T-I is invalid.

[6] This is the probability of a directed arc with label l from a fixed vertex u, but not a probability over all the directed arcs. We have tried the latter scoring function but the result was worse than the former scoring function which we report in this paper.

$s(v, w, l)$ is the same score function used in the MST estimation and $p(n)$ is a function that gives a penalty when the n-th additional arc is added to G. Thus the arc of the highest s is added if the s value is larger than the penalty $p(n)$.

We adopt an exponentially increasing function as the penalty function $p(n)$ with parameters λ and ξ as follows[7]:

$$p(n) = \frac{\xi}{\lambda e^{-\lambda n}}.$$

The denominator on the right-hand side is an exponential distribution with parameter λ. The numerator ξ is a scale parameter. At the training step we first estimate λ which minimizes the square error between the training-data distribution of the number of arcs added to the tree and the exponential distribution. Then we choose ξ which maximizes the F-measure on the held-out data, a small portion of training data, as we do in the deleted interpolation method (Jelinek, 1985).

7.3 Features

The feature function outputs a high dimensional feature vector that represents a characteristic of a labeled arc (u, v, l).

We extract features from labeled arcs (u, v, l) by two processes: first we extract features from the arc and input recipe text and then we concatenate the label l to each feature we extract. First the following features are extracted from the input arc (u, v) and the recipe text:

F1: The number of concepts existing between u and v with sign, which is +1 if u is left to v and -1 otherwise,

F2: Whether u and v are in the same sentence,

F3: Whether u and v are in the same step,

F4: Word sequences, pronunciation and concept tags of each u and v;[8]

F5: Three preceding words and three following words for both u and v,

F6: concept tag of $u \wedge$ concept tag of v,

F7: concept tag of $u \wedge$ concept tag of $v \wedge$ whether u and v are in the same sentence,

F8: concept tag of $u \wedge$ concept tag of $v \wedge$ whether Ac exists between u and $v \wedge$ whether a function word exists between u and v,

F9: concept tag of $u \wedge$ concept tag of $v \wedge$ function words between u and v.

Here the symbol $\wedge$ indicates the combination of individual features.

Next we simply concatenate the label l with each feature to construct feature vectors of labeled arcs. For example, we extracts a feature "concept tag of $u \wedge$ concept tag of v." Then, this type of feature becomes "$l \wedge$ concept tag of $u \wedge$ concept tag of v." by concatenating the label l. The same concatenation of a label is done on the other features.

So-called higher order features which refer to the neighboring vertices in the DAG are common in work on dependency parsing (McDonald et al., 2006; Koo and Collins, 2010), but we do not use these kinds of features because we only have 200 recipes annotated with DAGs[9]. This number is much smaller than roughly 40,000 sentences of the Wall Street Journal which are commonly used to train dependency parsers (Marcus et al., 1994).

Method	Precision	Recall	F-measure
Baseline	65.1	61.5	63.2
Proposed	73.5	69.1	71.2

Table 6: Accuracies of the baseline and proposed methods.

8 Evaluation

We evaluated our framework on the r-FG corpus described in Table 1. We executed 10-fold cross validation for more reliable results.

DAG estimation accuracy is measured by the F-measure of labeled arcs, which is the harmonic mean of precision and recall. Let N_{sys}, N_{ref}, and N_{int} be the number of the estimated arcs, the gold standard arcs, and their intersection, respectively. Then precision = N_{int}/N_{sys}, recall = N_{int}/N_{ref}, and F-measure = $2N_{int}/(N_{ref} + N_{sys})$.

[7] The reason is that, in small training data, the relationship between the number of additional arcs and the number of the flow graph matched with an exponentially decreasing function well.

[8] The pronunciations are automatically estimated based on the method described in (Mori and Neubig, 2011).

[9] Mori et al. (2014) state that it took about one our to annotate one recipe with word boundaries, concept tags, and a flow graph. It is much more coslty than syntactic annotation.

Task	Input	F-meas.
WS	Raw texts	98.6
CI	Gold WS results	90.7
Flow graph estimaiton	Gold WS/CI results	72.1
WS + CI + flow graph estimaiton	Raw texts	51.6

Table 7: F-measure of each task and the overall task.

8.1 Flow Graph Estimation

As the first evaluation, we compared the simple application of the MST parser and our extension. We assumed the gold WS and CI results.

8.1.1 Settings of Flow Graph Estimation

We compared two methods in the following way.

Baseline A naive method for the text parsing is a simple application of MST parser (McDonald et al., 2005) to a concept sequence input. An MST parser takes a sequence of words annotated with POSs and outputs a labeled tree connecting all the words. Thus our baseline for flow graph estimation takes a sequence of concepts (pairs of a word sequence and a concept type) as the input. The output of an MST parser is a tree, but not a DAG. So we add our arc addition module for a fair comparison. As the implementation, we modified a Japanese dependency parser (Flannery et al., 2012) that uses the logistic regression as the scoring function.

Proposed This combines the spanning tree estimation (Subsection 7.1) and the arc addition (Subsection 7.2) in the cascaded manner. Different from the **Baseline**, this method referred to words not included in concepts such as function words as the features .

Table 6 shows the accuracies of the baseline method and our MST extension. We can see that there is a significant difference in accuracy between **Baseline** and **Proposed**. The major difference between these two methods is whether or not they refer to the words not covered by the concepts in the original texts, such as in **F5** for example. These words are mainly function words. Therefore we can say that the function words are, as we know intuitively, important for the relationships among the concepts.

8.2 Text Parsing on a Raw Text

We also executed the text parsing taking a raw text as the input. For this problem, we combine WS, CI, and flow graph estimation (**Proposed**) in the cascaded manner.

The performance measurement is F-measure. Different from the first experiment, the unit is a triplet $(\langle \boldsymbol{w}_s, c_s \rangle, \langle \boldsymbol{w}_e, c_e \rangle, l)$. Here, $\boldsymbol{w}_s$ and c_s are the word sequence of the out-going vertex of the arc and its concept type, respectively. $\boldsymbol{w}_e$ and c_e are those of its in-coming vertex. And l is its label. A triplet is correct if and only if all these elements match with those of an arc in the manually annotated data.

8.2.1 Settings of Word Segmentation and Concept Identification

We built an automatic word segmenter and an automatic concept identifier in the following way.

WS: The word segmenter (see Section 5) is trained on the following corpora.

1. Balanced Corpus of Contemporary Written Japanese (Maekawa, 2008) containing fully segmented 53,899 sentences from newspaper articles, books, magazines, whitepapers, Web logs, and Web QAs.
2. The partially segmented sentences derived from 208 recipes in the r-FG corpus and additional 208 recipes annotated with concept types. In the experiment, we excluded the test part in 10-fold cross validation. Thus we built 10 models in total.
3. Partially annotated 1,651 sentences crawled from another recipe Web site[10].

CI: The concept identifier (see Section 6) is trained on the corpus 2. used in the WS training in the same way. Thus we built 10 models in total for 10-fold cross validation[11].

[10] `http://park.ajinomoto.co.jp/` accessed on 2014/May/21.

[11] We made this concept identifier with the model trained on 416 recipes publicly available from `http://plata.ar.media.kyoto-u.ac.jp/data/recipe/home-e.html`.

8.2.2 Results

Table 7 shows the accuracies of WS, CI, and flow graph estimation on the gold results of the preceding task and that of the combination of three tasks starting from a raw text to a flow graph.

As we see in the table, the flow graph estimation task is the most difficult and has a large room for improvement. The accuracy of WS without adaptation was about 95% and our adaptation technique raised it to about 99% which is as high as in the general domain case. The accuracy of CI, trained on less than 3,000 sentences, is as high as the general NER whose accuracy is about 90% by a model trained on about 10,000 sentences. This is still lower than WS accuracy, so the concept identifier is also a target of improvement.

From Table 7, the accuracy of the cascade combination of three tasks (WS + CI + flow graph estimation) is 51.6. This value is lower than the simple multiplication result that assumes the independence among the tasks $57.2 = (0.986^{1.27} \times 0.907)^2 \times 0.721 \times 100$, where 1.27 is the average word length of the concepts. This indicates that it is worth trying to investigate joint methods for WS, CI, and flow graph estimation.

9 Conclusion

In this paper, we proposed a framework of procedural text understanding consisting of the three processes. The first process is the well-known word identification. The second one is concept identification, which can be solved in the same way of named entity recognition with different definition of named entities. The third one is our original text parsing, which estimates a flow graph taking a text and the concepts in it as the input.

We tested our framework on recipe texts manually annotated with a flow graph. The results showed that our method outperforms a naive application of an MST dependency parser. Thus we can say that the simple application of dependency parsing to flow graph estimation does not work well, and that it is important to focus on not only concepts but also words surrounding them. Finally we evaluated the full automatic process of building a flow graph from a raw text. Our result can be a solid baseline for future improvement in the procedural text understanding problem.

Our method is applicable to various procedural texts allowing us to realize more intelligent search engine for how-to texts, more sophisticated symbol grounding by combining NLP and CV, etc.

Acknowledgments

This work was supported by JSPS Grants-in-Aid for Scientific Research Grant Numbers 26280084, 26280039, and 24240030.

References

Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1).

Andrew Borthwick. 1999. *A Maximum Entropy Approach to Named Entity Recognition*. Ph.D. thesis, New York University.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164.

Nancy A. Chinchor. 1998. Overview of MUC-7/MET-2. In *Proceedings of the Seventh Message Understanding Conference*.

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

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

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Raquel Fernández, Jonathan Ginzburg, and Shalom Lappin. 2004. Classifying ellipsis in dialogue: A machine learning approach. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 240–246.

Daniel Flannery, Yusuke Miyao, Graham Neubig, and Shinsuke Mori. 2012. A pointwise approach to training dependency parsers from partially annotated corpora. *Journal of Natural Language Processing*, 19(3).

Reiko Hamada, Ichiro Ide, Shuichi Sakai, and Hidehiko Tanaka. 2000. Structural analysis of cooking preparation steps in Japanese. In *Proceedings of the fifth International Workshop on Information Retrieval with Asian Languages*, pages 157–164.

Atsushi Hashimoto, Naoyuki Mori, Takuya Funatomi, Yoko Yamakata, Koh Kakusho, and Michihiko Minoh. 2008. Smart kitchen: A user centric cooking support system. In *Proceedings of the 12th Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 848–854.

Atsushi Hashimoto, Tetsuro Sasada, Yoko Yamakata, Shinsuke Mori, and Michihiko Minoh. 2014. Kusk dataset: Toward a direct understanding of recipe text and human cooking activity. In *Proceedings of the SixthInternational Workshop on Cooking and Eating Activities*.

Frederick Jelinek. 1985. Self-organized language modeling for speech recognition. Technical report, IBM T. J. Watson Research Center.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11.

Kikuo Maekawa. 2008. Balanced corpus of contemporary written Japanese. In *Proceedings of the 6th Workshop on Asian Language Resources*, pages 101–102.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: Annotating predicate argument structure. In *Proceedings of the ARPA Workshop on Human Language Technology*, pages 114–119.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 216–220.

Bernard Merialdo. 1994. Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2):155–171.

Yoshio Momouchi. 1980. Control structures for actions in procedural texts and PT-chart. In *Proceedings of the Eighth International Conference on Computational Linguistics*, pages 108–114.

Shinsuke Mori and Graham Neubig. 2011. A pointwise approach to pronunciation estimation for a TTS front-end. In *Proceedings of the InterSpeech2011*, pages 2181–2184, Florence, Italy.

Shinsuke Mori and Graham Neubig. 2014. Language resource addition: Dictionary or corpus? In *Proceedings of the Nineth International Conference on Language Resources and Evaluation*, pages 1631–1636.

Shinsuke Mori, Tetsuro Sasada, Yoko Yamakata, and Koichiro Yoshino. 2012. A machine learning approach to recipe text processing. In *Proceedings of the 1st Cooking with Computer Workshop*, pages 29–34.

Shinsuke Mori, Hirokuni Maeta, Yoko Yamakata, and Tetsuro Sasada. 2014. Flow graph corpus from recipe texts. In *Proceedings of the Nineth International Conference on Language Resources and Evaluation*, pages 2370–2377.

Iftekhar Naim, Young Chol Song, Qiguang Liu, Henry Kautz, Jiebo Luo, and Daniel Gildea. 2014. Unsupervised alignment of natural language instructions with video segments. In *Proceedings of the 28th National Conference on Artificial Intelligence*.

Graham Neubig, Yosuke Nakata, and Shinsuke Mori. 2011. Pointwise prediction for robust, adaptable Japanese morphological analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 529–533.

Leif Arda Nielsen. 2004. Verb phrase ellipsis detection using automatically parsed text. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 1093–1099.

Vignesh Ramanathan, Percy Liangy, and Li Fei-Fei. 2013. Video event understanding using natural language descriptions. In *Proceedings of the 14th International Conference on Computer Vision*.

Michaela Regneri, Marcus Rohrbach, Dominikus Wetzel, Stefan Thater, Bernt Schiele, and Manfred Pinkal. 2013. Grounding action descriptions in videos. *Transactions of the Association for Computational Linguistics*, 1:25–36.

Marcus Rohrbach, Wei Qiu, Ivan Titov, Stefan Thater, Manfred Pinkal, and Bernt Schiele. 2013. Translating video content to natural language descriptions. In *Proceedings of the 14th International Conference on Computer Vision*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Computational Natural Language Learning*, pages 142–147.

Tetsuro Sasada, Shinsuke Mori, Tatsuya Kawahara, and Yoko Yamakata. 2015. Named entity recognizer trainable from partially annotated data. In *Proceedings of the Eleventh International Conference Pacific Association for Computational Linguistics*.

Hirotoshi Taira, Sanae Fujita, and Masaaki Nagata. 2010. Predicate argument structure analysis using transformation-based learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*.

Yuta Tsuboi, Hisashi Kashima, Shinsuke Mori, Hiroki Oda, and Yuji Matsumoto. 2008. Training conditional random fields using incomplete annotations. In *Proceedings of the 22nd International Conference on Computational Linguistics*.

Liping Wang, Qing Li, Na Li, Guozhu Dong, and Yu Yang. 2008. Substructure similarity measurement in Chinese recipes. In *Proceedings of the 17th International Conference on World Wide Web*, pages 978–988.

Yuk Wah Wong and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 960–967.

Yoko Yamakata, Shinji Imahori, Yuichi Sugiyama, Shinsuke Mori, and Katsumi Tanaka. 2013. Feature extraction and summarization of recipes using flow graph. In *Proceedings of the 5th International Conference on Social Informatics*, LNCS 8238, pages 241–254.

Koichiro Yoshino, Shinsuke Mori, and Tatsuya Kawahara. 2013. Predicate argument structure analysis using partially annotated corpora. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of Uncertainty in Artificial Intelligence*.