Technical Report of *Japanese Morphological Analyzer*

Jun Jiang, Vernkin Chen

October 16, 2009

Abstract

In Japanese language, word boundaries are not clear. As word-level information is useful for analysis of known words, while character-level information is useful for analysis of unknown words, we present a hybrid method for Japanese word segmentation, which utilizes both these two types of information in order to effectively handle known and unknown words. To resolve the word boundaries ambiguity, we employ a statistical model based on conditional random fields(CRFs) to the situations where word boundary ambiguity exists.

Changes

Date	Author	Notes
2009-08-28	Jun, Vernkin	Initial version
2009-08-31	Vernkin	Add Change Log Section
2009-08-31	Jun	Update project schedule section with the final refinement stage.
2009-09-01	Vernkin	Update the "Compile System Dictionary" and "Run the Demo" Section in User Guide Chapter and "User Dictionary" section in Dictionary Chapter.
2009-09-01	Jun	Add section 6.3 "Manual Comparison Result"
2009-09-11	Jun	Update section 5.2 "How to use the interface" on OPTION_TYPE_POS_FORMAT_ALPHABET in JMA API to configure POS result in alphabet format.
2009-09-18	Jun	Update section 6.1 "Dictionary Statistics" on extending Juman dictionary size by adding proper nouns from IPA dictionary.
2009-09-22	Jun	Update project schedule appendix B on getting iJMA pass the QC.
2009-09-23	Jun	Add Part-of-Speech Tagset appendix A.
2009-09-30	Jun	Update chapter 5 "User Guide" on how to build and use iJMA, and add chapter 7 "Conclusion".

Contents

Chapter 1

Introduction

Automatic morphological analysis is a widely-used technique for the development of NLP systems and linguistically-annotated corpora. Morphological analysis in Chinese and Japanese is an important and difficult task. In these languages, words are not separated by explicit delimiters, and word segmentation must be conducted first in most natural language processing applications. One of the problems which makes word segmentation more difficult is existence of unknown (out-of-vocabulary) words. Unknown words are defined as words that do not exist in a system's dictionary. The word segmentation system has no knowledge about these unknown words, and determining word boundaries for such words is difficult. Accuracy of word segmentation for unknown words is usually much lower than that for known words.

In this TR, we first propose a hybrid method for Chinese and Japanese word segmentation, which utilizes both word-level and character-level information. Word-level information is useful for analysis of known words, and character-level information is useful for analysis of unknown words. We use these two types of information at the same time to obtain high overall performance.

A great amount of studies have attempted to develop software for performing automatic morphological analysis with high accuracy, and several systems for Japanese morphological analysis have been freely available (Asahara and Matsumoto, 2000[1]; Kudo et al., 2004[2]).

In traditional morphological analysis on computer, the task is divided into two sub-tasks: i) segmentation of an input string into a sequence of units, and ii) assignment of a part of speech (POS) tag to each segmented unit. Linguistic analysis of corpora, however, requires more information than one provided by these sub-tasks (see e.g., Mizutani, 1983[4]). For instance,

Japanese words often have variation in orthography; verb arawasu (to express) is written either as "表わす","表す," or "あらわす," and noun sakura (a cherry blossom) either as "桜," "サクァ," or "さくら" In examining the frequency of words occurring in a text, linguists usually want to collapse these variants. Japanese also has lots of heteronyms[1], two or more words that have the identical writing form but different word forms. For instance, nouns namamono (raw food) and seibutu (a living thing) are both written as "生物." These issues are crucial for linguistic analysis of corpora, but are not handled by traditional Japanese morphological analyzers on computer. These tasks, though operating in opposite directions—one mapping two or more different writing forms onto a single word form and the other mapping one writing form onto more than one word form, can be regarded as the same problem; that is identification of lemmas, i.e., entry words in a dictionary. The task of identifying a lemma corresponding to each segmented unit in an input has been totally ignored in the study of automatic morphological analysis of Japanese.

In this paper, we use a hybrid approach for Japanese morphological analysis. We first propose an electronic dictionary, UniDic, which employs hierarchical definition of word indexes to represent orthographic variants as well as allomorphs. In this hierarchical structure, heteronyms are represented as different nodes with the same index but with different super-nodes. We then propose a statistical model for resolving word boundary ambiguity, basing on CRFs.

[1] An example of heteronym in English is "bow," which has two different meanings with different sounds, /bou/ and /bau/. In this paper, writing forms refer to representation in orthography, which corresponds to spelling in English. Word forms, on the other hand, are based on kana-reading and roughly correspond to sounds, although in a few cases, e.g., particles wa and e, there is dissociation between kana-reading and sound.

Chapter 2

Related Works

2.1 The Character Tagging Method

This method carries out word segmentation by tagging each character in a given sentence, and in this method, the tags indicate word-internal positions of the characters. We call such tags position-of-character (POC) tags (Xue, 2003[9]) in this paper. Several POC-tag sets have been studied (Sang and Veenstra, 1999[6]; Sekine et al., 1998[7]), and we use the 'B, I, E, S' tag set shown in Table 2.1[1].

Table 2.1: The 'B, I, E, S' Tag Set

B	The character is in the beginning of a word.
I	The character is in the middle of a word.
E	The character is in the end of a word.
S	The character is itself a word.

An example of POC-Tagging:

Sentence: 窓 の 近く に 大きな 木 が あり ます 。

POC Tag: 窓/B の/E 近/B く/E に/S 大/B き/I な/S 木/S が/S あ/B り/E ま/B す/E 。/S

The POC-tags can represent word boundaries for any sentences, and the word segmentation task can be reformulated as the POC-tagging task. The tagging task can be solved by using general machine learning techniques

[1] The 'B, I, E, S' tags are also called '$OP-CN$, $CN-CN$, $CNCL$, $OP-CL$' tags (Sekine et al., 1998[7]) or 'LL, MM, RR, LR' tags (Xue, 2003[9]).

such as maximum entropy (ME) models (Xue, 2003[9]) and support vector machines (Yoshida et al., 2003[8]).

This character tagging method can easily handle unknown words, because known words and unknown words are treated equally and no other exceptional processing is necessary. This approach is also used in base-NP chunking (Ramshaw and Marcus, 1995[5]) and named entity recognition (Sekine et al., 1998[7]) as well as word segmentation.

However, this character tagging method is not a standard method in 20-year history of corpus-based Japanese morphological analysis. This is because it cannot directly reflect lexicons which contain prior knowledge about word segmentation.

2.2 UniDic

UniDic[2] is with the aim of providing a proper tool for Japanese morphological analysis (Den et al., 2007[10])[3]. The dictionary has the following features:

1. The unit for identifying a word is based on the short unit word (Maekawa, in press), which provides word segmentation in uniform size, without being harmed by too long words.

2. The indexes for words are defined at several levels, including *lemma*, *form*, and *orthography*, which enables us to represent orthographic variants as well as allomorphs.

3. An extensive amount of phonological information, such as lexical accent and sandhi, is also described and can be utilized in speech research.

2.3 Word Segmentation Using Word-Level and Character-Level Information

For the Word-Level segmentation, taking Markov model as example (And we will use CRF). It is observed that the Markov model-based method has high

[2]Freely available at http://download.unidic.org/

[3]Throughout the paper, writing forms, i.e., indexes at the orthography level, are written in Japanese characters, and word forms, i.e., indexes at the form level, are written in Romaji. A lemma is expressed by a triple consisting of a standardized word form, a standardized writing form, and a meaning articulated in English.

overall accuracy, however, the accuracy drops for unknown words, and the character tagging method has high accuracy for unknown words but lower accuracy for known words (Yoshida et al., 2003[8]; Xue, 2003[9]). This seems natural because words are used as a processing unit in the Markov model-based method, and therefore much information about known words (e.g., POS or word bigram probability) can be used. However, unknown words cannot be handled directly by this method itself. On the other hand, characters are used as a unit in the character tagging method. In general, the number of characters is finite and far fewer than that of words which continuously increases. Thus the character tagging method may be robust for unknown words, but cannot use more detailed information than character-level information.

Then, we propose a hybrid method which combines the Word-Level method and the character tagging method to make the most of word-level and character-level information, in order to achieve high overall accuracy. The hybrid method employ the statistical model based on CRFs to the situations where word boundary ambituity exists.

2.4 Goshu Information

Japanese words often have variation in orthography and the vocabulary of Japanese consists of words of several different origins, it sometimes happens that more than one writing form corresponds to the same lemma and that a single writing form corresponds to two or more lemmas with different readings and/or meanings.

Hence, the mapping from a writing form onto a lemma is important in linguistic analysis of corpora. The current study focuses on disambiguation of heteronyms, words with the same writing form but with different word forms. Goshu is the classification of words based on their origin. So it could be employed as features in the heteronym disambiguation.

2.5 Comparison of ChaSen and MeCab

ChaSen and MeCab are popular in Japanese morphological analysis.

ChaSen (Asahara and Matsumoto, 2000[1]) run with the published version of UniDic (a dictionary with the aim of providing a proper tool for Japanese morphological analysis) , which employs (an extension of) a hidden Markov model (HMM) to determine the optimal segmentation and POS assignment but can only bring a poor modeling for lemma identification, i.e., the uni-

gram probability of lemmas given a writing form. Incorporating statistical information of goshu classes into an HMM-based analyzer, however, is problematic since in HMMs the only way to utilize goshu information is to introduce new tags that consist of combinations of POS tags and goshu classes and this will easily lead us to the data sparseness.

MeCab (Kudo et al., 2004[2]) is more recent and is based on a novel statistical method, conditional random fields (CRFs) (Lafferty et al., 2001[3]), which overcome several problems of HMMs including label bias, length bias, and difficulty in using features that can co-occur at the same position such as the POS tag and the goshu class. With the lexicon contained in ChaSen's standard dictionary and the RWCP Text Corpus as the training data, MeCab is shown to outperform ChaSen in the segmentation and POS assignment tasks.

2.6 Conclusion

From the survey, CRF are better choice than HMM and ME. Also we perform a small experiment on the CRF with character tagging approach, which precision is 0.964 (Corpus are generated from the MeCab). And we think it is good enough to develop our JMA based on the CRF. Also, we will use the hybird method to gain a higher precision than 0.964.

Chapter 3

Algorithm

3.1 Problem Description

In Japanese morphological analysis, we assume that a lexicon (or dictionary) is available, which lists a pair of a word and its corresponding part-of-speech. The lexicon gives a tractable way to build a lattice from an input sentence. A lattice represents all candidate paths or all candidate sequences of tokens, where each token denotes a word with its part-of-speech.

In the building of the lattice, if no matching word could be found in the lexicon, unknown word processing would be invoked. That is, the characters in the same type would be grouped together as a candidate token. The character types include *hiragana*, *katakana*, Chinese characters, alphabets, numbers and punctuations.

Formally, the task of Japanese morphological analysis can be defined as follows. Let $\mathbf{x}$ be an input, unsegmented sentence. Let $\mathbf{y}$ be a path consisting of a sequence of tokens where each token is a pair of word w_i and its part-of-speech t_i. Then their relation could be described below, where $\#\mathbf{y}$ is the number of tokens in the path $\mathbf{y}$:

$$\mathbf{y} = (\langle w_1, t_1 \rangle, \ldots, \langle w_{\#\mathbf{y}}, t_{\#\mathbf{y}} \rangle) \tag{3.1}$$

Let $\mathcal{Y}(\mathbf{x})$ be a set of candidate paths in a lattice built from the input sentence $\mathbf{x}$ and a lexicon. The goal is to select a correct path $\hat{\mathbf{y}}$ from all candidate paths in the $\mathcal{Y}(\mathbf{x})$.

The distinct property of Japanese morphological analysis is that the number of tokens $\mathbf{y}$ varies, since the set of labels and the set of states are not the same.

3.2 Conditional Random Fields

To achieve the high accuracy and performance, our system uses the statistical analysis approach based on Conditional Random Fields (CRF)(Lafferty et al., 2001[3]).

CRFs are discriminative models and can thus capture many correlated features of the inputs. This allows flexible feature designs for hierarchical tagsets. CRFs have a single exponential model for the joint probability of the entire paths given the input sentence, while MEMMs consist of a sequential combination of exponential models, each of which estimates a conditional probability of next tokens given the current state. This minimizes the influences of the label and length bias.

As explained in 3.1, we use the lattice to represent the word boundary ambiguity in Japanese. This implies that the set of labels and the set of states are different, and the number of tokens $\#\mathbf{y}$ varies according to a path.

In order to accomodate this, we define CRFs for Japanese morphological analysis as the conditional probability of an output path $\mathbf{y}$ given an input sequence $\mathbf{x}$:

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z_{\mathbf{x}}} \exp\Big(\sum_{i=1}^{\#\mathbf{y}} \sum_{k} \lambda_k f_k(\langle w_{i-1}, t_{i-1}\rangle, \langle w_i, t_i\rangle)\Big) \tag{3.2}$$

where $Z_{\mathbf{x}}$ is a normalization factor over all candidate paths, i.e.,

$$Z_{\mathbf{x}} = \sum_{\mathbf{y}' \in \mathcal{Y}(\mathbf{x})} \exp\Big(\sum_{i=1}^{\#\mathbf{y}'} \sum_{k} \lambda_k f_k(\langle w'_{i-1}, t'_{i-1}\rangle, \langle w'_i, t'_i\rangle)\Big) \tag{3.3}$$

$f_k(\langle w_{i-1}, t_{i-1}\rangle, \langle w_i, t_i\rangle)$ is an arbitrary feature function over i-th token $\langle w_i, t_i\rangle$ and its previous token $\langle w_{i-1}, t_{i-1}\rangle$. $\lambda_k (\in \mathbf{\Lambda} = \{\lambda_1, \ldots, \lambda_K\} \in \mathbb{R}^K)$ is a learned weight or parameter associated with feature function f_k.

Note that the formulation 3.2 is different from the widely-used formulation 3.4 below. The previous applications of CRFs assign a conditional probability for a label sequence $\mathbf{y} = y_1, \ldots, y_T$ given an input sequence $\mathbf{x} = x_1, \ldots, x_T$ as:

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z_{\mathbf{x}}} \exp\Big(\sum_{i=1}^{T} \sum_{k} \lambda_k f_k(y_{i-1}, y_i, \mathbf{x})\Big) \tag{3.4}$$

In formulation 3.2, CRFs deal with word boundary ambiguity. Thus, the size of output sequence T is not fixed through all candidates $\mathbf{y} \in \mathcal{Y}(\mathbf{x})$. The index i is not tied with the input $\mathbf{x}$ as in the original CRFs, but unique to the output $\mathbf{y} \in \mathcal{Y}(\mathbf{x})$.

Here, we introduce the *global feature vector* $\mathbf{F}(\mathbf{y}, \mathbf{x})$ below:

$$\mathbf{F}(\mathbf{y}, \mathbf{x}) = \{F_1(\mathbf{y}, \mathbf{x}), \ldots, F_K(\mathbf{y}, \mathbf{x}))\} \tag{3.5}$$

$$F_k(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{\#\mathbf{y}} f_k(\langle w_{i-1}, t_{i-1}\rangle, \langle w_i, t_i\rangle) \tag{3.6}$$

Using the above global feature vector, $P(\mathbf{y}|\mathbf{x})$ in formulation 3.2 can also be represented as:

$$P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z_{\mathbf{x}}} \exp\left(\mathbf{\Lambda} \cdot \mathbf{F}(\mathbf{y}, \mathbf{x})\right) \tag{3.7}$$

The most probable path $\hat{\mathbf{y}}$ for the input sentence $\mathbf{x}$ is then given by

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{x})}{\operatorname{argmax}} P(\mathbf{y}|\mathbf{x}) = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{x})}{\operatorname{argmax}} \mathbf{\Lambda} \cdot \mathbf{F}(\mathbf{y}, \mathbf{x}) \tag{3.8}$$

which can be found with the Viterbi algorithm.

Chapter 4

Dictionary

As explained in 3.1, we need a lexicon (or dictionary) to build the lattice in Japanese morphological analysis. So dictionary plays an important role in our JMA. That's because the character tagging method (in 2.1) cannot directly reflect lexicons which contain prior knowledge about word segmentation. We cannot ignore a lexicon since over 90% accuracy can be achieved even using the longest prefix matching with the lexicon.

Basing on the approach used in MeCab, we introduce the dictionary usage in our JMA.

4.1 Source Files of System Dictionary

The source files include *dicrc*, **.csv*, *char.def*, *unk.def*, *matrix.def*, *rewrite.def*, *left-id.def*, *right-id.def*, *pos-id.def*, which are described below respectively.

- dicrc

  The MeCab configuration file. Table 4.1 is an example.

Table 4.1: Example of dicrc

config-charset = EUC-JP
dictionary-charset = EUC-JP
cost-factor = 800
bos-feature = BOS/EOS,*,*,*,*,*,*,*,*
eval-size = 8
unk-eval-size = 4

The meaning of each entry in Table 4.1 is described below.

config-charset character encoding of files *dicrc*, *char.def*, *unk.def*, *rewrite.def*, *left-id.def*, *right-id.def* and *pos-id.def*.

dictionary-charset character encoding of CSV files, used in compiling CSV files into binary file.

cost-factor scaling factor used in cost calculation.

bos-feature feature definition for sentence beginning and ending.

eval-size the number of features used in training evaluation on known words.

unk-eval-size the number of features used in training evaluation on unknown words.

- *.csv

The CSV file contains the dictionary words in text format. Multiple CSV files could be defined if only its file extension is *csv*. Each line in the file is an entry of one word. Table 4.2 is an example of CSV file from IPA dictionary.

Table 4.2: Example of CSV

行く,994,994,8852,動詞,非自立,*,*,五段　カ行促音便,基本形,行く,イク,イク
行か,1002,1002,7754,動詞,非自立,*,*,五段　カ行促音便,未然形,行く,イカ,イカ
行こ,998,998,8417,動詞,非自立,*,*,五段　カ行促音便,未然ウ接続,行く,イコ,イコ
行き,1014,1014,8036,動詞,非自立,*,*,五段　カ行促音便,連用形,行く,イキ,イキ
行け,986,986,7878,動詞,非自立,*,*,五段　カ行促音便,仮定形,行く,イケ,イケ

The meaning of each column in Table 4.2 is described below in Table 4.3.

Table 4.3: Format of CSV

column id	content example	meaning in Japanese	meaning in English
1	行く	表層形	string of the word entry
2	994	左連接状態番号	index as a left token in context state
3	994	右連接状態番号	index as a right token in context state
4	8852	コスト	word cost used in run time analysis
5	動詞	品詞	POS category in 1st level
6	非自立	品詞細分類1	POS category in 2nd level
7	*	品詞細分類2	POS category in 3rd level
8	*	品詞細分類3	POS category in 4th level
9	五段　カ行促音便	活用型	conjugation type
10	基本形	活用形	conjugation form
11	行く	基本形	base form
12	イク	読み	reading form
13	イク	発音	pronounciation form

In Table 4.3, the first four columns are mandatory for analysis. The succeeding columns are called *feature*, including those Japanese language characteristics such as POS, conjugation, etc.

The value in column 2 and 3 is the index in a context state, which index value is selected from *left-id.def* and *right-id.def* respectively. This value could also be set to -1, so that in dictionary compiling, MeCab would automatically select an appropriate index basing on those features in succeeding columns. Column 4 is a cost value of word. The smaller this value, the higher frequency this word is assumed.

- char.def

  The definition file for unkown word processing, which groups characters into unkown words if they are in the same category. The characters are categorized into symbol, numeric, Kanji, Hiragana, Katakana, etc.

- unk.def

  The feature definition file for the character categories defined in *char.def*. The format of this file is similar with that of CSV file.

- matrix.def

  The definition file for the connection matrix. Each value in the matrix represents a cost value for connecting adjacent tokens, which cost value would be used in CRF analysis.

- rewrite.def

  The definition file for feature selection. In this definition file, the features are selected or rewritten for the CRF analysis. For example, columns from 5 to 12 in Table 4.3 could be selected as features in the definition. Besides, as Japanese words often have variation in orthography, such as verb *kuru* (to come) is written either as “来る” or “くる”, by means of this definition file, the unified form “来る” could be selected as the base form in the features.

- left-id.def & right-id.def

  The definition file for feature index, which index value is used as column 2 and 3 in Table 4.3.

- pos-id.def

  The definition file for POS index. In this file, index number is defined for each POS string. If this file is not defined, the POS index of each word would be set to 1 defaultly.

4.2 Binary Files of System Dictionary

Also it is called the Compiled System Dictionary. After compiling, some of the dictionary source files are converted to binary type, which are used for run time analysis. These binary files are listed below.

- sys.dic

  The binary file compiled from those CSV files **.csv*.

- char.bin

  The binary file compiled from *char.def* and *unk.def*.

- unk.dic

  The binary file compiled from *unk.def*.

- matrix.bin

  The binary file compiled from *matrix.def*.

- pos-feature.def

  The Simple POS maps to feature. This file are with " simplePOS = feature" format. The Simple POS used in the User Dictionary are only allowed appeared in this file. If give wrong Simple POS, the Program only print out warning and reset the Simple POS to default POS.

4.3 User Dictionary File

User dictionary are CSV files or TXT (text) files.

4.3.1 CSV User Dictionary

If user dictionary are CSV files, its format is the same as that in Table 4.3. Unlike the CSV files in system dictionary, which need to be compiled into a binary file *sys.dic* for run time using, multiple user CSV files could be loaded in run time directly. Table 4.4 is an example of user CSV file.

In Table 4.4, we could see that column 2 and 3 are set to -1. In this case, please confirm that the features in succeeding columns has a corresponding index value in file *left-id.def* and *right-id.def*, otherwise MeCab would not be able to select an appropriate index value for this entry. User could also adjust the cost value in column 4 with a small integer for those high frequency words.

Table 4.4: Example of User Dictionary

眞人,-1,-1,0,名詞,固有名詞,人名,名,*,*,眞人,マサト,マサト
藤樹,-1,-1,0,名詞,固有名詞,人名,名,*,*,藤樹,トウジュ,トージュ
香与子,-1,-1,0,名詞,固有名詞,人名,名,*,*,香与子,カヨコ,カヨコ
千紘,-1,-1,0,名詞,固有名詞,人名,名,*,*,千紘,チヒロ,チヒロ

4.3.2 TXT User Dictionary

The TXT User Dictionary are more flexible than CSV User Dictionary. Each entry is in one line. The simple POS see pos-feature.def in each Compiled System Dictionary (see 4.2).

Each entry can be:

- the word itself. Like "眞人". The word would be given the default Simple POS, which is set by attribute "*default-pos*" in the file dicrc under Compiled System Dictionary.
- the word and the one or more Simple POSs, these tokens are separated by spaces, like "眞人 N V".
- The same as entry in the CSV file, like "眞人,-1,-1,0,名詞,固有名詞,人名,名,*,*,眞人,マサト,マサト".
- the word and the completed feature, like " 眞人 -1,-1,0,名詞,固有名詞,人名,名,*,*,眞人,マサト,マサト" (The first comma becomes space).

4.4 Dictionary Usage

The CSV files in the source directory of system dictionary are editable so that user could revise words and add new words as well. When CSV files are modified or new CSV file is added, the source files of system dictionary need to be compiled into binary type for run time loading. The CSV files as user dictionary could be loaded at run time directly without compiling. The details of these files have been described in previous sections.

JMA supports different character encodings such as *EUC-JP*, *SHIFT-JIS* and *UTF-8*. The run time encoding type should be the same with that of system dictionary in binary type. A specific encoding type could be configured and used like below steps.

1. Set character encoding of system CSV files.

That is, set the entry **dictionary-charset** with value *EUC-JP*, *SHIFT-JIS* or *UTF-8* in *dicrc* file, which file is under source directory of system dictionary. The example of *dicrc* file is shown in Table 4.1, which file in IPA dictionary could be available as */jma/db/ipadic/src/dicrc*.

2. Compile system dictionary source files to binary type in run time encoding.

   Below is an example of compiling system dictionary files into run time encoding of *EUC-JP*.

   ```
   $ cd bin
   $ mkdir ../db/ipadic/bin_eucjp
   $ ./jma_encode_sysdict --encode eucjp ../db/ipadic/src ../db/ipadic/bin_eucjp
   ```

   In the above example, the source files of system dictionary are in directory *../db/ipadic/src*. They are compiled to binary files in directory *../db/ipadic/bin_eucjp*.

   Before executing program *jma_encode_sysdict*, please ensure that the directory for binary files *../db/ipadic/bin_eucjp* already exists. The value of command option *--encode* in this program could be *eucjp*, *sjis* or *utf8*, which would be the character encoding type on JMA run time.

3. Set character encoding of user CSV file.

   This step is similar with step 1, except with a different *dicrc* file.

   That is, set the entry **dictionary-charset** with value *EUC-JP*, *SHIFT-JIS* or *UTF-8* in *dicrc* file, which file is under the directory of system dictionary in binary type, instead of the directory in source type. That is because at JMA run time, only the binary directory of system dicitonary and user CSV file would be loaded.

   The directory of IPA dictioanry in binary type could be available as */jma/db/ipadic/bin_eucjp*, and the example file of user CSV file could be */jma/db/userdic/ipa_eucjp.csv*. Both of them are in *EUC-JP* encoding type.

4. Load the dictionaries at JMA run time analysis.

   At run time, before loading dictionary files, call JMA library interface *Knowledge::setEncodeType()* to set the character encoding type. The parameter[1] of this interface should be the same with the encoding type used in step 2.

   Then call interfaces *Knowledge::setSystemDict()* and *Knowledge::addUserDict()* to set dictionary paths, and call interface *Knowledge::loadDict()* to load those dictionary files. The interface calling example could be found in section 5.2.

[1] ENCODE_TYPE_EUCJP, ENCODE_TYPE_SJIS, or ENCODE_TYPE_UTF8

Chapter 5

User Guide

5.1 Interface description

The C++ API of the library is described below:

Class JMA_Factory creates instances for JMA, its methods below create instances of JMA_Factory, Analyzer and Knowledge, which are used in the morphological analysis.

```
static JMA_Factory* instance();
virtual Analyzer* createAnalyzer() = 0;
virtual Knowledge* createKnowledge() = 0;
```

Class Knowledge manages the linguistic information, its methods below load files of system dictionary, user dictionary, stop-word dictionary and configuration files.

```
virtual int setSystemDict(const char* dirPath) = 0;
virtual int addUserDict(const char* fileName) = 0;
virtual int loadDict() = 0;
virtual int loadStopWordDict(const char* fileName) = 0;
virtual int loadSentenceSeparatorConfig(const char* fileName) = 0;

void setEncodeType(EncodeType type);
virtual int encodeSystemDict(const char* txtFileName, const char* binFileName) = 0;
```

By means of Knowledge::setEncodeType(), we could set the character encode as EUC-JP, SHIFT-JIS or UTF-8. If this function is not called, the default encode is EUC-JP. Knowledge::encodeSystemDict() is the interface of encoding system dictionary from text type to the binary one.

Class Analyzer executes the morphological analysis, its methods below execute the morphological analysis based on a sentence, a paragraph or a file

separately. Analyzer::splitSentence() splits a paragraph into sentences.

```
virtual void setKnowledge(Knowledge* pKnowledge) = 0;
void setOption(OptionType nOption, double nValue);

virtual int runWithSentence(Sentence& sentence) = 0;
virtual const char* runWithString(const char* inStr) = 0;
virtual int runWithStream(const char* inFileName, const char* outFileName) = 0;

virtual void splitSentence(const char* paragraph, std::vector<Sentence>& sentences) = 0;
```

Class Sentence saves the analysis results of Analyzer::runWithSentence(), so that the n-best or one-best results could be accessed.

```
void setString(const char* pString);
const char* getString(void) const;
int getListSize(void) const;
int getCount(int nPos) const;
const char* getLexicon(int nPos, int nIdx) const;
const char* getBaseForm() const;
int getPOS(int nPos, int nIdx) const;
const char* getStrPOS(int nPos, int nIdx) const;
double getScore(int nPos) const;
int getOneBestIndex(void) const;
```

5.2 How to use the interface

The interface could be used like the following steps:

1. Include the header files in directory include.

```
#include "jma_factory.h"
#include "analyzer.h"
#include "knowledge.h"
#include "sentence.h"
```

2. Use the library name space.

```
using namespace jma;
```

3. Call the interface and handle the result.

In the example below, the return value of some functions are not handled for simplicity. In your using, please properly handle those return values in case of failure. The interface details could be available either in document docs/html/annotated.html, or the comments in the header files of directory include.

```
// create instances
JMA_Factory* factory = JMA_Factory::instance();
Analyzer* analyzer = factory->createAnalyzer();
Knowledge* knowledge = factory->createKnowledge();

// (optional) set character encode type, the default encode type is EUCJP,
// you could set it as SJIS or UTF8
knowledge->setEncodeType(Knowledge::ENCODE_TYPE_SJIS);

// set the system dictionary path
knowledge->setSystemDict("db/jumandic/bin_eucjp");
// (optional) add the paths of multiple user dictionaries
knowledge->addUserDict("db/userdic/juman_eucjp.csv");
knowledge->addUserDict("db/userdic/juman_eucjp.txt");
// load system and user dictionary files
knowledge->loadDict();

// (optional) load stop word dictionary
knowledge->loadStopWordDict("db/stopworddic/test-eucjp.txt");

// (optioinal) if Analyzer::splitSentence() would be called,
// load the sentence separator configuration file in specific encode type
knowledge->loadSentenceSeparatorConfig("db/config/sen-eucjp.config");

// (optional) if POS tagging is not necessary,
// disable the output of POS result when Analyzer::runWith***() is called
analyzer->setOption(Analyzer::OPTION_TYPE_POS_TAGGING, 0);

// (optional) output POS result in alphabet format such like "NP-S", instead of in Japanese format
analyzer->setOption(Analyzer::OPTION_TYPE_POS_FORMAT_ALPHABET, 1);

// (optional) output POS result in full category of Japanese format,
// some POS results might include asterisk symbol
analyzer->setOption(Analyzer::OPTION_TYPE_POS_FULL_CATEGORY, 1);

// (optional) get 5-best results when Analyzer::runWithSentence() is called
analyzer->setOption(Analyzer::OPTION_TYPE_NBEST, 5);

// set knowledge
analyzer->setKnowledge(knowledge);

// 0. split paragraphs into sentences
string line;
vector<Sentence> sentVec;
while(getline(cin, line)) // get paragraph string from standard input
{
    sentVec.clear(); // remove previous sentences
    analyzer->splitSentence(line.c_str(), sentVec);
    for(size_t i=0; i<sentVec.size(); ++i)
    {
        analyzer->runWithSentence(sentVec[i]); // analyze each sentence
        ...
    }
}

// 1. analyze a sentence
Sentence s;
s.setString("...");
analyzer->runWithSentence(s);

// get one-best result
int i= s.getOneBestIndex();
```

```
...

// get n-best results
for(int i=0; i<s.getListSize(); ++i)
{
    for(int j=0; j<s.getCount(i); ++j)
    {
        const char* pLexicon = s.getLexicon(i, j);
        const char* strPOS = s.getStrPOS(i, j);
        // get the base form of the specific Japanese word
        const char* baseForm = s.getBaseForm(i, j);
        ...
    }
    double score = s.getScore(i);
    ...
}

// 2. analyze a paragraph
const char* result = analyzer->runWithString("...");
...

// 3. analyze a file
analyzer->runWithStream("...", "...");

// destroy instances
delete knowledge;
delete analyzer;
```

5.3 How to build and link with the library

CMake[1] is used as the build system.

5.3.1 Under Linux platform

Use script build.sh in directory build like below.

```
$ cd build
$ ./build.sh
```

After the project is built, the library targets libjma.a and libmecab.so are created in directory lib, and the executables in directory bin are created for demo and test.

To link with the library, the user application needs to include header files in directory include, and link the library files libjma.a, libmecab.so in directory lib.

An example of compiling user application test.cpp looks like:

```
$ export JMA_PATH=path_of_jma_project
```

[1] http://www.cmake.org

```
$ g++ -I$JMA_PATH/include -o test test.cpp $JMA_PATH/lib/libjma.a \
  $JMA_PATH/lib/libmecab.so -Wl,-rpath,$JMA_PATH/lib
```

5.3.2 Under Win32 platform

Please check that whether Visual Studio 9 2008 exists on your Win32 platform. If it does not exist, please install it beforehand. You could also use other Visual Studio version, in which case you need to revise the script build/build.sh to set the value of flag WIN32_BUILD_SYSTEM to your own Visual Studio version.

Then use script build.sh in directory build like below.

```
$ cd build
$ ./build.sh release win32
```

Then the MSVC project file build/temp/JMA.sln is generated, please open and build it inside MSVC IDE. After the project is built, the library targets jma.lib, mecab.lib and mecab.dll are created in directory lib/Release, and the executables in directory bin/Release are created for demo and test.

To link with the library, the user application needs to include header files in directory include, and link the library files jma.lib, mecab.lib in directory lib.

To run the user application, it is also necessary to put mecab.dll in the search path used by Windows to locate it.

5.4 How to run the demo

To run the demo bin/jma_run, please make sure the project is built (see 5.3), and also the compiled system dictionary exists on your disk, such like db/jumandic/bin_eucjp.

Below is the demo usage:

```
$ cd bin
$ ./jma_run --sentence N-best [--dict DICT_PATH]
$ ./jma_run --string [--dict DICT_PATH]
$ ./jma_run --stream INPUT OUTPUT [--dict DICT_PATH]
```

The DICT_PATH is the path of compiled system dictionary, such like db/jumandic/bin_eucjp, which is the default system dictionary used in demo.

Demo of analyzing 5-best results of a sentence string from standard input.

```
$ ./jma_run --sentence 5
```

Demo of analyzing a paragraph string from standard input.

```
$ ./jma_run --string
```

To exit the loop in the above demos, please press CTRL-C.

Demo of analyzing file from INPUT to OUTPUT. The example of INPUT file could be available as ../db/test/asahi_test_raw_eucjp.txt.

```
$ ./jma_run --stream INPUT OUTPUT
```

5.5 How to use the dictionary

5.5.1 System dictionary

The CSV files (such like db/jumandic/src/Noun.koyuu.csv) are source files of system dictionary. Whenever the words are revised or added in source files, you need to compile them into binary files like below.

```
$ cd bin
$ mkdir ../db/jumandic/bin_eucjp
$ ./jma_encode_sysdict --encode eucjp ../db/jumandic/src ../db/jumandic/bin_eucjp
```

In the above example, the character encoding of binary files is set as EUC-JP. For encoding SHIFT-JIS and UTF-8, please use sjis and utf8 instead of eucjp.

5.5.2 User dictionary

The user dictionary (such like db/userdic/juman_eucjp.csv or db/userdic/juman_eucjp.txt) could be loaded at run time without compilation beforehand.

Please note that the character encoding of user dictionary could be configured by the entry dictionary-charset in dicrc file, which is under the directory of compiled system dictionary (such like db/jumandic/bin_eucjp/dicrc).

5.5.3 Run time loading

In the example below, call the APIs to load dictionary files at run time.

```
// set character encode type, for which EUCJP, SJIS or UTF8 are available
knowledge->setEncodeType(Knowledge::ENCODE_TYPE_EUCJP);
```

```
// set the system dictionary path
knowledge->setSystemDict("db/jumandic/bin_eucjp");

// add the paths of multiple user dictionaries
knowledge->addUserDict("db/userdic/juman_eucjp.csv");
knowledge->addUserDict("db/userdic/juman_eucjp.txt");

// load system and user dictionary files
if(knowledge->loadDict() == 0)
{
    cerr << "fail to load dictionary files" << endl;
    exit(1);
}
```

Chapter 6

Experiment

We compared the result of our JMA with two other JMAs. One is Basis[1], the other is ChaSen[2]. We also used three dictionaries separately in our JMA, which are introduced below firstly.

6.1 Dictionary Staticstics

Two of the dictionaries are called IPA and Juman, which are in the source package of MeCab[3]. The other dictionary is called UniDic[4].

These dictionaries are built from different corpora, where each corpus has a different POS tagset and word segmentation standard. For example, the Juman dictionary is built from Kyoto University Corpus[5]. The staticstics of these dictionaries are listed in Table 6.1.

Table 6.1: Dictionary Statistics

Dictionary	#words	POS structure	#POS categories
IPA	392,033	4-levels	69
Juman	674,967	2-levels	43
UniDic	559,239	4-levels	53

By comparing these dictionaries manually, we found the nouns coverage in

[1]http://www.basistech.com
[2]http://chasen.naist.jp
[3]http://mecab.sourceforge.net
[4]http://www.tokuteicorpus.jp/dist
[5]http://www-lab25.kuee.kyoto-u.ac.jp/nl-resource/corpus.html

Juman dictionary is the lowest, so we extracted 158,964 nouns from IPA into Juman dictionary, which extends Juman dictionary word count from 516,003 to 674,967.

The column of POS structure in Table 6.1 means the number of POS levels. For example, IPA dictionary contains four levels in POS category, which could also be seen in Table 4.3. And the last column in Table 6.1 gives the number of POS categories at the bottom level.

Regarding the dictionary used in Basis, from its online document[6], it is said that there are about 600,000 words in its Japanese dictionary. And the POS structure of Basis is 1-level containing 29 POS categories.

By comparing the POS structure, we could see that the POS category number in our dictionaries is much more than that of Basis. So that the morpheme results in our JMA is more rigorous than Basis on Japanese grammar, which comparison result could also be seen in the section below.

6.2 Automatic Comparison Result

In the automatic comparison, we used Basis and ChaSen as the base JMA to compare with, which means that the result of the base JMA would be assumed as the standard answer.

Then we use precision, recall and balanced F-score as the comparison measure. Precision (P) is defined as the number of correctly segmented words divided by the total number of words in our segmentation result. Recall (R) is defined as the number of correctly segmented words divided by the total number of words in the standard answer. F-score (F) is defined as below:

$$F = \frac{P \times R \times 2}{P + R} \tag{6.1}$$

The test data is the raw text extracted from the news website[7], which data size is 806K bytes. By using this test data, we got the F-score result in Table 6.2, in which table the numbers are those F-scores between the base JMA and our JMA using those three dictionaries respectively.

From the comparison result, the highest F-score is 0.948, which is between ChaSen and our JMA with IPA dictionary. This is because ChaSen also uses the same source from IPA dictionary. As each dictionary gives its own definition on what should be a word, we could see that the comparison result

[6] http://www.basistech.com/data-sheets/basis-asian-text-analysis.pdf
[7] http://www.asahi.com

Table 6.2: Comparison F-score

Base JMA	IPA	Juman	UniDic
Basis	0.830	0.858	0.806
ChaSen	0.948	0.811	0.896

relies on the dictionary largely.

Compared with Basis, the F-score is above 0.8, although Basis dictionary is very different from ours. By comparing the analysis result manually, we found that our result is more rigorous than Basis on Japanese grammar.

For example, from the raw input text "もらえます", the analysis result of Basis is one word "もらえます/Verb 動詞", while our result on IPA dictionary contains two separated words "もらえ/Verb 動詞,自立" and "ます/Particle 助動詞", which result is more rigorous on Japanese grammar, and giving more details in word boundary and POS information.

By comparing the total number of segmented words in the analysis results, the number of Basis result is about ten pecentage less than ours. So we could see that our JMA could give more details on morphological information.

6.3 Manual Comparison Result

We also compared the precisions of our JMA (iJMA) with Basis by manual check. That is, the incorrect analysis results are picked out by users who could read Japanese.

We used IPA dictionary on iJMA in this manual check. The test data is the raw text extracted from web sites[8] by users randomly. The comparison result is summarized in Table 6.3.

Table 6.3: Precision Comparison by Manual Check

	Token Number	Same Percentage	Precision
Basis	14,970	84.85%	**95.99%**
iJMA	16,498	76.99%	**94.18%**

In above table, the 1st column is the total number of tokens in each JMA's

[8] www.zakzak.co.jp, www.yahoo.co.jp, www.goo.ne.jp, etc

output. We could see that the token number of iJMA is ten percentage more than that of Basis.

The 2nd column means how many tokens in one JMA's output are the same with that of the other JMA. By using the formulation 6.1, the comparison F-score between Basis and iJMA is 0.807.

The last column gives the precision value, meaning how many tokens are correct in user's manaul check. In the review of manual check results, we found that those incorrect results could be mainly categorized into two situations.

One situation is caused by lacking of the word in dictionary. For example, as the word "有元" (used as Japanese surname) does not exist in IPA dictinary, it is segmented into two words "有" and "元" incorrectly.

The other situation is caused by different segmentation standards between IPA dictionary and user's manual check. For example, the evaluator assumes "高さ" (meaning highness in English) is a whole word. While in iJMA result, it is separated into two words "高" (meaning high) and "さ" (meaning ness).

So we could see that both results could be correct on its own segmentation standard. Regarding iJMA with IPA dictionary, its segmentation standard is more rigorous than that of Basis. By comparing the three dictionaries of iJMA in table 6.2, we could see that the segmentation standard of Juman dictionary is most closest to that of Basis.

Both of these situations could be improved by adding those words into iJMA dictionary. For example, after words "有元" and "高さ" are added into a user dictionary loaded in iJMA, they are output as whole words.

So we could see that iJMA could be improved by expanding or adjusting the dictionary. The dictionary usage in iJMA could be found in section 4.4.

6.4 Analysis Speed Evaluation

The test environment for analysis speed evaluation is described in Table 6.4.

Table 6.4: Test environment

Platform	Ubuntu x86_32 Kernel Linux 2.6.28-14
Memory	3.7GB
CPU	Intel Core 2 Duo 2.33GHz

In case of 1 Megabyte raw input file, the execution time of word segmentation is evaluated on Basis, ChaSen, and our JMA with those three dictionaries

respectively. The statistics is listed in Table 6.5.

Table 6.5: Time statistics (seconds) on 1 Megabyte file

Basis	ChaSen	IPA	Juman	UniDic
6.83	0.68	**0.76**	**1.43**	**1.2**

In the time statistics result, ChaSen is the fastest one. The analysis speed of our JMA with IPA dictionary is very close to that of ChaSen, which uses the same dictionary source. The speed of our JMA relies on the dictionary size and unkown word processing rules.

From the above statistics, we could see that our JMA is faster than Basis about 5 to 9 times.

Chapter 7

Conclusion

7.1 Quality and Performance

We evaluated iJMA quality with two approaches.

One is comparing iJMA result with Basis. The result shows that the comparison F-score with Basis is above 0.8, which means that in the results of iJMA and Basis, more than 80% percentage are the same.

We also evaluated iJMA and Basis results by manual check. The result shows that iJMA with IPA dictionary is more rigorous than Basis on word segmentation standard, and their precisions are very close. So we choose to use Juman as our system dictionary, which segmentation standard is more close to Basis.

Regarding the performance, in analyzing the raw text in UTF-8 encoding for example, the speed of iJMA is more than 1 Megabytes per second, which is 5 times faster than Basis.

The evalutaion result shows that iJMA is capable in realistic usage of Japanese morphological analysis.

7.2 What Affects the Quality

From the experiment result, we could see that the quality is influenced by several factors below.

7.2.1 Dictionary Size

The dictionary coverage could largely affect the quality. To improve the dictionary, we extended the nouns in Juman dictionary, which word count reaches to 674,967 (Basis contains about 600,000 words in its Japanese dictionary).

iJMA also supports user dicitonary, so that user could easily expand the dictionary coverage on their own requirements.

7.2.2 Word Segmentation Standard

As the segmentation standard between iJMA dictionary and Basis are not the same, it could be acceptable that some of their results are both correct on its own standard. So it would be better to evaluate the quality basing on their respective segmentation standard.

To make the standard of iJMA as closely as possible with Basis, we chose Juman as our system dictionary.

7.3 Future Work

Based on the morphological analysis supplied in iJMA, more Japanese natural language processing task could be performed, such as entity extraction, syntax analysis, etc.

Appendix A

Appendix - Summary of the Part-of-Speech Tagset

Table A.1: IPA Dictionary tagset

Tag in Alphabet	Tag in Japanese	Tag in Alphabet	Tag in Japanese
O	その他,間投	D-P	副詞,助詞類接続
F	フィラー	N-VS	名詞,サ変接続
I	感動詞	N-NE	名詞,ナイ形容詞語幹
S-A	記号,アルファベット	NC-G	名詞,一般
S-G	記号,一般	N-R	名詞,引用文字列
PUNCT-L	記号,括弧開	N-AJV	名詞,形容動詞語幹
PUNCT-R	記号,括弧閉	NP-G	名詞,固有名詞,一般
EOS	記号,句点	NP-H	名詞,固有名詞,人名,一般
S-S	記号,空白	NP-S	名詞,固有名詞,人名,姓
PUNCT-C	記号,読点	NP-GN	名詞,固有名詞,人名,名
AJ-I	形容詞,自立	NP-O	名詞,固有名詞,組織
AJ-S	形容詞,接尾	NP-P	名詞,固有名詞,地域,一般
AJ-D	形容詞,非自立	NP-C	名詞,固有名詞,地域,国
PL-G	助詞,格助詞,一般	NN	名詞,数
PL-R	助詞,格助詞,引用	NJ	名詞,接続詞的
PL-C	助詞,格助詞,連語	NS-SN	名詞,接尾,サ変接続
PL-A	助詞,係助詞	NS-G	名詞,接尾,一般
PL-E	助詞,終助詞	NS-AJV	名詞,接尾,形容動詞語幹
PL-J	助詞,接続助詞	NU	名詞,接尾,助数詞
PL-O	助詞,特殊	NS-AUV	名詞,接尾,助動詞語幹
PL-DS	助詞,副詞化	NS-H	名詞,接尾,人名
PL-D	助詞,副助詞	NS-P	名詞,接尾,地域
PL-K	助詞,副助詞 / 並立助詞 / 終助詞	NS-S	名詞,接尾,特殊
PL-P	助詞,並立助詞	NS-D	名詞,接尾,副詞可能
PL-N	助詞,連体化	NR-G	名詞,代名詞,一般
AUV	助動詞	NR-A	名詞,代名詞,縮約
J	接続詞	N-VD	名詞,動詞非自立的
P-A	接頭詞,形容詞接続	N-AUV	名詞,特殊,助動詞語幹
P-NN	接頭詞,数接続	ND-G	名詞,非自立,一般
P-V	接頭詞,動詞接続	AN	名詞,非自立,形容動詞語幹
P-N	接頭詞,名詞接続	ND-AUV	名詞,非自立,助動詞語幹
V-I	動詞,自立	ND-D	名詞,非自立,副詞可能
V-S	動詞,接尾	N-D	名詞,副詞可能
V-D	動詞,非自立	AA	連体詞
D-G	副詞,一般		

Table A.2: Juman Dictionary tagset

Tag in Alphabet	Tag in Japanese	Tag in Alphabet	Tag in Japanese
I	感動詞	S-NS	接尾辞,名詞性名詞接尾辞
AJ	形容詞	V	動詞
DM-D	指示詞,副詞形態指示詞	PUNCT-L	特殊,括弧始
DM-N	指示詞,名詞形態指示詞	PUNCT-R	特殊,括弧終
DM-A	指示詞,連体詞形態指示詞	S-G	特殊,記号
PL-G	助詞,格助詞	EOS	特殊,句点
PL-E	助詞,終助詞	S-S	特殊,空白
PL-J	助詞,接続助詞	PUNCT-C	特殊,読点
PL-D	助詞,副助詞	AS	判定詞
AUV	助動詞	D	副詞
J	接続詞	N-VS	名詞,サ変名詞
P-AE	接頭辞,イ形容詞接頭辞	N-F	名詞,形式名詞
P-AN	接頭辞,ナ形容詞接頭辞	NP-G	名詞,固有名詞
P-V	接頭辞,動詞接頭辞	N-T	名詞,時相名詞
P-N	接頭辞,名詞接頭辞	NP-H	名詞,人名
S-AP	接尾辞,形容詞性述語接尾辞	NN	名詞,数詞
S-AN	接尾辞,形容詞性名詞接尾辞	NP-O	名詞,組織名
S-V	接尾辞,動詞性接尾辞	NP-P	名詞,地名
S-NP	接尾辞,名詞性述語接尾辞	NC	名詞,普通名詞
S-NO	接尾辞,名詞性特殊接尾辞	N-D	名詞,副詞的名詞
S-NN	接尾辞,名詞性名詞助数辞	AA	連体詞

Table A.3: UniDic Dictionary tagset

Tag in Alphabet	Tag in Japanese	Tag in Alphabet	Tag in Japanese
NR	代名詞	AJV-T	形状詞,タリ
D	副詞	AJV-G	形状詞,一般
AUV	助動詞	AJV-AV	形状詞,助動詞語幹
PL-A	助詞,係助詞	I-F	感動詞,フィラー
PL-D	助詞,副助詞	I-G	感動詞,一般
PL-J	助詞,接続助詞	S-V	接尾辞,動詞的
PL-G	助詞,格助詞	S-NS	接尾辞,名詞的,サ変可能
PL-N	助詞,準体助詞	S-NG	接尾辞,名詞的,一般
PL-E	助詞,終助詞	S-ND	接尾辞,名詞的,副詞可能
V-G	動詞,一般	S-NN	接尾辞,名詞的,助数詞
V-D	動詞,非自立可能	S-NAV	接尾辞,名詞的,形状詞可能
N-AV	名詞,助動詞語幹	S-AJ	接尾辞,形容詞的
NP-G	名詞,固有名詞,一般	S-AV	接尾辞,形状詞的
NP-H	名詞,固有名詞,人名,一般	J	接続詞
NP-GN	名詞,固有名詞,人名,名	P	接頭辞
NP-S	名詞,固有名詞,人名,姓	S-S	空白
NP-P	名詞,固有名詞,地名,一般	S-G	補助記号,一般
NP-C	名詞,固有名詞,地名,国	EOS	補助記号,句点
NP-O	名詞,固有名詞,組織名	PUNCT-R	補助記号,括弧閉
NN	名詞,数詞	PUNCT-L	補助記号,括弧開
N-VS	名詞,普通名詞,サ変可能	PUNCT-C	補助記号,読点
N-SAV	名詞,普通名詞,サ変形状詞可能	S-AG	補助記号,ＡＡ,一般
NC-G	名詞,普通名詞,一般	S-AE	補助記号,ＡＡ,顔文字
N-D	名詞,普通名詞,副詞可能	S-CG	記号,一般
N-AV	名詞,普通名詞,形状詞可能	S-CL	記号,文字
AJ-G	形容詞,一般	AA	連体詞
AJ-D	形容詞,非自立可能		

Appendix B

Appendix - Project schedules and milestones

	Milestone	Start	Finish	In Charge	Description	Status
1	Survey on the project	2008-05-04	2008-05-15	Vernkin	1. Have a general overview of current Japanese Segmentation techniques and their differences. 2. Select a suitable one and do more research on it.	Finished
2	Experiment using CMA approach	2009-05-11	2009-05-15	Jun	Experiment CMA approach using only character-level information, which result reached 0.96 on ASAHI text. To improve the result further for JMA, we would adopt the hybrid approach, which utilizes both word-level and character-level information.	Finish
2	Analyze requirement specification, and study the potential solutions	2009-05-18	2009-05-22	Vernkin	Analyze requirements, and study the hybrid approach.	Finish
3	Implement the Japanese morphological analyzer	2009-05-25	2009-06-26	Jun	Implement an initial version of this project.	Finish
4	Experiment and debugging	2009-06-29	2009-07-31	Vernkin	Experiment the system on training data and test data, and debugging the system.	Finish
5	Compare iJMA with basis	2009-08-03	2009-08-31	Vernkin	Build a web platform for JMAs comparison, so that user could compare and manual check the results.	Finish
6	Improve iJMA to pass the QC.	2009-09-01	2009-09-25	Jun	1. Basing on the feedback of the manual comparison by Wisenut users, extract those words corrected by user into iJMA user dictionary. And in the three system dictionaries of iJMA, select the one which segmentation standard is most closest to user requirement. And improve the system dictionary as far as possible. 2. Enhance iJMA features, such as improve the usability of user dictionary, support POS alphabet output, etc. 3. To meet the requirement of QC, such as multithread environment support, test iJMA thoroughly and fix any bug found. 4. Wrap up the project, including TR, doxygen documents, etc.	Finish

Bibliography

[1] Masayuki Asahara and Yuji Matsumoto. Extended models and tools for high-performance part-of-speech tagger. *Proceedings of the 18th International Conference on Computational Linguistics*, pages 21–27, 2000.

[2] Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. Applying conditional random fields to japanese morphological analysis. *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 230–237, 2004.

[3] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[4] Shizuo Mizutani. Vocabulary (in japanese). *Asakura New Series on Japanese*, 2, 1983.

[5] Lance Ramshaw and Mitch Marcus. Text chunking using transformation-based learning. *Proceedings of the 3rdWorkwhop on Very Large Corpora*, pages 88–94, 1995.

[6] Erik F. Tjong Kim Sang and Jorn Veenstra. Representing text chunks. *Proceedings of 9th Conference of the European Chapter of the Association forComputational Linguistics*, pages 173–179, 1999.

[7] Ralph Grishman Satoshi Sekine and Hiroyuki Shinnou. A decision tree method for finding and classifying names in japanese texts. *Proceedings of the 6th Workshop on Very Large Corpora*, pages 171–177, 1998.

[8] Kiyonori Ohtake Tatsumi Yoshida and Kazuhide Yamamoto. Performance evaluation of chinese analyzers with support vector machines. *Natural Language Processing*, 10(1):109–131, 2003.

[9] Nianwen Xue. Chinese word segmentation as character tagging. *International Journal of Computational Linguistics and Chinese*, 8(1):29–48, 2003.

[10] Hideki Ogura Atsushi Yamada Nobuaki Minematsu Kiyotaka Uchimoto Yasuharu Den, Toshinobu Ogiso and Hanae Koiso. The development of an electronic dictionary for morphological analysis and its application to japanese corpus linguistics (in japanese). *Japanese Linguistics*, 22:101–123, 2007.

Index