NTT's Japanese-English Cross-Language Question Answering System

Hideki ISOZAKI, Katsuhito SUDOH, Hajime TSUKADA
NTT Communication Science Laboratories
Nippon Telegraph and Telephone Corporation
2-4 Hikaridai, Seikacho, Sorakugun, Kyoto, 619-0237, Japan
{isozaki,sudoh,tsukada}@cslab.kecl.ntt.co.jp

Abstract

This paper describes NTT's Japanese-English Cross-Language Question Answering System SAIQA-J/E. The system performed best among eight systems that participated in the Japanese-English subtask of the NTCIR Cross-Language Question Answering task. For cross-language document retrieval, we used a dictionary-based approach without word sense disambiguation. We used a synonym operator to represent translation alternatives. Our experiments show that the synonym operator improved the retrieval precision of our proximity-based document retrieval module. We also developed a web-based back-transliteration submodule for unknown katakana words.

Keywords: *cross-language information retrieval, question answering, transliteration*

1 Introduction

NTT's Japanese-English Cross-Language Question Answering System SAIQA-J/E is based on a Japanese Question Answering System SAIQA that performed best in NTCIR QAC-2 [1] and an English Question Answering System SAIQA-e. We reused SAIQA's question analysis module and SAIQA-e's answer extraction/evaluation modules. The Japanese question analysis module uses ALTJAWS morphological analyzer based on a Japanese lexicon "Nihongo Goi-Taikei" [2].

A new module, the *Japanese-English query translation module*, was introduced to translate the output of the Japanese question analysis module into English, and the translated query is sent to SAIQA-e. Figure 1 shows a rough sketch of the system.

In the past, we participated in TREC QA tracks, but the performance of our English QA system was mediocre: e.g., MRR = 0.228 for TREC-10 QA track [3]. There were several problems in our English QA system, but the most serious problem was the inefficiency of our SVM-based English NE recognizer.

Since our SVM-based Japanese Named Entity (NE) recognizer performed very well, we simply applied the same method to English NE recognition. We prepared a training corpus for English NE by manually annotating English news articles for five NE classes: PERSON, LOCATION, ORGANIZATION, FACILITY, and ARTIFACT. The Japanese NE recognizer used a five-word window function to classify a word, and we followed this method. In English NE recognition, however, we found that important clues such as appositive phrases were not covered by this window function. Therefore, we tried to incorporate such clues into our English NE recognizer, but it required heuristic rules to detect distant apposition relations.

However, SVM was too inefficient to find a better combination of such features. In order to make it faster, we built another NE system, in which NE candidates are detected by hand-crafted rules and they are classified by SVMs into three NE classes (PERSON, LOCATION, and ORGANIZATION) and one non-NE class (OTHER). The second NE system was much faster than the first, but it was still too slow. In addition, the second NE system can detect only three NE classes, and the rule-based NE detector degraded the accuracy. Because of this inefficiency, we had insufficient time to debug the entire QA system.

The second serious problem was the retrieval method. We used a passage retrieval engine based on a suffix array, but it was difficult to retrieve relevant passages.

We solved these problems after TREC-10. First, we proposed an efficient algorithm for SVM-based NE [4]. However, it does not cover dozens of answer types that we need for TREC QA. Therefore, we also developed a traditional rule-based NE recognizer. Second, we developed a proximity-based document retrieval method. This retrieval method performed better than Okapi BM25 tuned for Japanese QA [1]. Our experiment using TREC QA data also showed that our method outperformed tuned BM25.

The essential part of SAIQA-J/E is the query translation module, which we designed for our new document retrieval engine. Therefore, we first describe

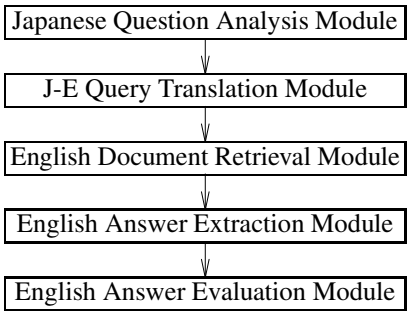

Figure 1. Rough sketch of SAIQA J/E system

our proximity-based document retrieval engine in Section 2. Then, we describe the query translation module in Section 3. This module has a web-based back-transliteraton submodule. We describe it in detail. Finally, we describe the English answer extraction/ evaluation module in Section 4.

2 Proximity-based Document Retrieval Engine

For English word indexing, we used TreeTagger (http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/) for part-of-speech tagging. We also used downcasing and a variant of the Porter stemmer (http://www.tartarus.org/~martin/PorterStemmer/). We did not remove any words from the index.

We described our proximity-based document retrieval engine in [5, 1]. With our method, document D's score is defined as the maximum value of the scores of all possible passages in D.

$$DS_{\mathrm{DIDF}(\beta)}(D) = \max_{p \sqsubseteq D} PS_{\mathrm{DIDF}(\beta)}(p).$$

Here, $p_1 \sqsubseteq p_2$ means that a passage p_1 is contained in another passage p_2. The document D is also regarded as a passage.

The score of a passage spanning the l-th word to the r-th word is given by the following definition.

$$PS_{\mathrm{DIDF}(\beta)}([l,r]) = \exp(-\beta(r-l)) \sum_{q \in Q([l,r])} \mathit{idf}[q]$$

where $\mathit{idf}[q] = \log(N/\mathit{df}[q])$ is Inverse Document Frequency (IDF) of query term q. $\mathit{df}[q]$ is the number of documents that contain q. N is the total number of documents. $Q([l,r])$ is the set of query terms that appear in the passage.

β is a non-negative *decay factor*. If β is large, the scoring function becomes nearsighted. A short passage that contains many important query terms is preferred. If β is small, the scoring function becomes farsighted. Even a long passage can obtain a good score if it contains many important query terms. We call this method Decayed IDF (DIDF).

score	Document ID	term(IDF),position list
26.14	20000224E1T-DY03B000010	drift(5.318),40, usuki(8.174),43 netherland(4.580),60,126,279,..
21.19	20000413E1T-DY03A000040	drift(5.318),255 usuki(8.174),262 netherland(4.580),245 ..
16.41	20010625E1T-DY18A000050	usuki(8.174) 6,68,129,205,214,317 *ship or2 vessel(3.330) 26,107,161* ..

Table 1. Output of the search engine

Note that this formula completely disregards the repetition of query terms. We think simple repetitions should not be regarded as evidence of an answer.

Our previous paper [1] shows that DIDF's precision was best around $\beta = 0.001$ for Japanese QA. BM25 was best around $k_1 = 0.1$ but DIDF was better than BM25. According to our experiment using TREC-11 QA data, DIDF gave the best retrieval precisions around $\beta = 0.001$. Okapi BM25 was best around $k_1 = 0.3$ but DIDF was again better than BM25.

Query terms for the proximity search are joined by the `or` operator. For instance, query '`A or B or C`' gives a set of query terms $Q = \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$.

However, one Japanese word often corresponds to two or more English expressions. We should consider these alternatives for document retrieval. For instance, `Nippon` is a synonym of `Japan`. If we use '`Japan or Nippon`,' the above scoring function prefers a passage that contains both `Japan` and `Nippon` to another passage that contains only `Japan` when the passages have the same length. However, synonyms should not be regarded as new evidence of an answer.

In order to avoid this problem, we incorporated a synonym operator `or2` in this engine. This operator treats query terms as if they are the same word. Then, we can use '`Japan or2 Nippon`' to represent the synonymity. IDF of '`Japan or2 Nippon`' is defined as the minimum of *idf*`[Japan]` and *idf*`[Nippon]`. Pirkola [6] also used such a synonym operator for cross-language information retrieval.

We also used `or2` for QAC-2, but we could not show its effectiveness because QAC-2 questions did not require paraphrasing to obtain answers. In CLQA, however, many Japanese words have two or more English translations. Therefore, we expect that `or2` will play an important rule in the CLQA system.

Table 1 shows the top 3 documents for CLQA1-JA-S0011-00 (1600 nen, Usuki ni hyouchaku shita Oranda no fune wa nan to iu?, What is the name of the Dutch ship that drifted ashore at Usuki in 1600?). The Japanese word 'fune' in the question is translated to '`watercraft or2 ship or2 boat or2 vessel or2 steamship`.' The output for the third document contains '`ship or2`

vessel.' This means that this document contains both ship and vessel, but they are treated as if they are the same word. In fact, ship appears at 26 and 107 and vessel appears at 161.

The romanization of Japanese words is ambiguous. For instance, 'Chatan-cho' is the name of a town. It is also written as 'Chatancho,' 'Chatan cho,' 'Chatan chou,' and so on. These variations can be automatically generated, and we used or2 to cover these variations.

3 Query Translation Module

SAIQA-J/E's query translation module depends on two submodules: 1) a dictionary-based word translation submodule and 2) a web-based back-transliteration submodule. The back-transliteration submodule is called when a query term is written in katakana and its translation is unknown.

3.1 Dictionary-based word translation

For each Japanese query term, the *query translation module* consults publicly available Japanese-English dictionaries (EDICT 110,428 entries, ENAMDICT 483,691 entries, http://www.csse.monash.edu.au/~jwb/j_edict.html) and an in-house translation dictionary (660,778 entries).

Since inter-word spaces are not used in Japanese language, different dictionaries lead to different segmentations of a sentence. Therefore, these dictionaries were also incorporated into a Japanese morphological analyzer ALTJAWS used in SAIQA.

Suppose these dictionaries give one or more English phrases $E_1, \ldots, E_m$ for a Japanese term. As we mentioned before, the use of 'or' gives too much scores to redundant occurrences of these phrases, so we use "E_1 or2 $\cdots$ or2 E_m" instead. Then, these English queries for different query terms are joined by the or operator, which works in the same way as adding IDF scores.

For instance, CLQA1-JA-S0031-0 ("Shoukaiseki ga shibou shita no wa itsu?" = When did Chiang Kai-shek die?) has two Japanese query terms: 'shoukaiseki' and 'shibou.' Therefore, the original Japanese query is 'shoukaiseki or shibou'. 'Shoukaiseki' has three alternatives in the above translation dictionaries: 'Shou Kaiseki' (Japanese pronunciation), 'Jiang Jieshi' (Chinese Pinyin), and 'Chiang Kai-Shek' (traditional English spelling). There are two translations for 'shibou': 'die' and 'mortal.' Therefore, the translated English query will be '("Shou Kaiseki" or2 "Jiang Jieshi" or2 "Chiang Kai Shek") or ("die" or2 "mortal").'

Japanese	Downcased & stemmed English query
北京	((pekin) or2 (beij))
義務	((duti) or2 (oblig) or2 (respons))
金閣寺	((templ or kinkaku) or2 (kinkakuji) or2 (pavilion or templ or golden))

Table 2. Translation Table

However, phrase matching is sometimes too strict even if the phrases are names. For instance, "William Clinton" will not match "Bill Clinton" or "President Clinton". Therefore, we replaced the above phrases by disjunctions. Hence, the query becomes '(((Shou) or (Kaiseki)) or2 ((Jiang) or (Jieshi)) or2 ((Chiang) or (Kai) or (Shek)) or ((die) or2 (mortal)).'

By downcasing and stemming such queries, we prepared a translation table as shown in Table 2. This translation table contains 847,155 Japanese words.

However, when we implemented or2, we assumed that or2 joins only single words and phrases. For instance, in 'w1 or2 w2 or2 (w3 or w4),' or2 occurs twice, but the current system recognizes only the synonymity of w1 and w2. We tried to extend the definition of or2 to such cases, but we were unsuccessful until the formal run of CLQA-1. Therefore, we decided to use the current implementation for CLQA-1.

By using the development dataset provided by the CLQA organizer, we found that some one kanji character entries in the above dictionaries degraded the system performance. For instance, the kanji character 'ichi' that corresponds to the English word 'one' has 18 entries (possible translations for different word senses) in ENAMDICT. Another kanji character 'nen' that corresponds to the English word 'year' has five entries. Most of these entries are unusual (perhaps wrong). Since these words appears in many questions, they will greatly degrade the system performance. Therefore, we extracted one character entries from the dictionaries, and asked a Japanese to remove inappropriate entries from the list.

3.2 Web-based Back-transliteration

In Japanese, foreign words are written in katakana, a set of phonetic characters. When a katakana word in a question does not have an entry in the translation table, the word is sent to a *back-transliteration submodule*. Figure 2 shows a web-based back-transliteration submodule. Here, transliteration is a process that converts an English word into a katakana sequence that is phonetically similar to the English word. Back-transliteration is a process that finds the original English word from the katakana word. It is possible to

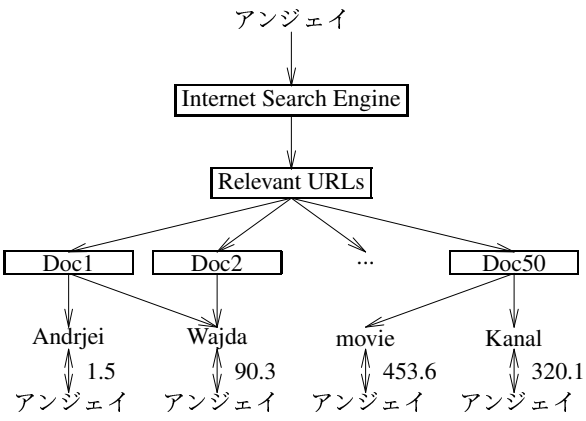

Figure 2. Web-based back-transliteration

build such a back-transliteration system from a set of training examples.

Tsuji et al. [7] proposed a web-based English-to-katakana transliteration system that send dozens of phonetically generated katakana candidates to a search engine for validation. It is possible to apply their method in the reverse direction. However, if the English spelling is unusual, we cannot expect this method to generate the correct spelling.

Here, we assume that there must be a web document that contains both the unknown katakana word and its spelling in English. In fact, we often find such documents in online encyclopedias and jargon lists. Therefore, we send only one query (i.e., the katakana word itself) to a search engine. We can obtain unusual correct spellings such as 'Andrjei' for katakana アンジェイ (*anjei*) by scoring candidates in relevant documents.

The submodule works as follows.

1. The submodule sends the katakana word to an internet search engine as it is. Then, the search engine returns URLs of relevant Japanese documents.

2. The submodule retrieves the 50 top documents from the web, and removes their HTML tags.

3. The submodule extracts strings of Roman letters from the documents. From the strings, word n-grams are extracted and enumerated as candidates.

4. A transliteration scorer ranks these candidates with respect to the given katakana word, and the submodule returns the highest scoring candidate. In Fig. 2, 'Andrjei' obtains the lowest penalty of 1.5, while 'movie' obtains the highest penalty of 453.6. Hence, 'Andrjei' is returned.

The transliteration scorer is trained as follows.

Given a katakana sequence ($\boldsymbol{k}$ = "$k_1 \cdots k_{l(k)}$") and a set of Roman letter sequences $\boldsymbol{R} = \{\boldsymbol{r}^1, \ldots, \boldsymbol{r}^n\}$. Here, $l(k)$ is the length of the string k. We find the most likely back-transliteration $\boldsymbol{r}^*$ that maximizes the transliteration likelihood as follows:

$$\begin{aligned} \boldsymbol{r}^* &= \operatorname*{argmax}_{\boldsymbol{r}\in\boldsymbol{R}} p(\boldsymbol{r} \mid \boldsymbol{k}) = \operatorname*{argmax}_{\boldsymbol{r}\in\boldsymbol{R}} p(\boldsymbol{k}, \boldsymbol{r})/p(\boldsymbol{k}) \\ &= \operatorname*{argmax}_{\boldsymbol{r}\in\boldsymbol{R}} p(\boldsymbol{k}, \boldsymbol{r}). \end{aligned} \quad (1)$$

For the calculation of the joint probability $p(\boldsymbol{k}, \boldsymbol{r})$, we consider a paired character sequence $\boldsymbol{c} = \langle c_1, ..., c_{l(c)} \rangle$ where each c_j is a pair $\langle c_j.k, c_j.r \rangle$. $c_j.k$ is a katakana character in $\boldsymbol{k}$ or a NULL character $\langle \epsilon \rangle$. $c_j.r$ is a Roman letter in $\boldsymbol{r}$ or $\langle \epsilon \rangle$. We assume the alignment is monotone. Based on this alignment, we approximate $p(\boldsymbol{k}, \boldsymbol{r})$ by considering only the most likely alignment that maximizes the transliteration likelihood among all possible paired character sequences $\boldsymbol{C}(\boldsymbol{k}, \boldsymbol{r})$.

$$\begin{aligned} p(\boldsymbol{k}, \boldsymbol{r}) &= \sum_{\boldsymbol{c}\in\boldsymbol{C}(\boldsymbol{k},\boldsymbol{r})} p(\boldsymbol{c}) \\ &\approx \max_{\boldsymbol{c}\in\boldsymbol{C}(\boldsymbol{k},\boldsymbol{r})} p(\boldsymbol{c}) \end{aligned} \quad (2)$$

We approximate $p(\boldsymbol{c})$ as a second-order Markov model as follows:

$$\begin{aligned} p(\boldsymbol{c}) &= \prod_{j=1}^{l(c)} p(c_j \mid c_1, \ldots, c_{j-1}) \\ &\approx \prod_{j=1}^{l(c)} p(c_j \mid c_{j-2}, c_{j-1}). \end{aligned} \quad (3)$$

We call this model the *transliteration model*. The calculation of $p(\boldsymbol{k}, \boldsymbol{r})$ can be implemented with two weighted finite state transducers (WFSTs),T and O. T is the transliteration WFST which reads a katakana character sequence and outputs the Roman letter sequences. The negative logarithms of the probabilities in 3 are used as weights in T. O is the output WFST which reads $\boldsymbol{r}$ and outputs nothing. That is, O is a finite state acceptor. Although we can obtain $\sum_{\boldsymbol{c}\in\boldsymbol{C}} p(\boldsymbol{c})$ with the fully-composed and determinized WFST $T \circ O$, we compose T and O partially by using a beam search because of computational cost.

In the training, we use only the most likely paired character sequence $\boldsymbol{c}_F^*$ obtained through a monotone alignment between $\boldsymbol{k}$ and $\boldsymbol{r}$ among all possible paired character sequences $\boldsymbol{c} \in \boldsymbol{C}(\boldsymbol{k}, \boldsymbol{r})$.

We determine $\boldsymbol{c}_F^*$ as the one that maximizes the character alignment score F as follows:

$$\boldsymbol{c}_F^*(\boldsymbol{k}, \boldsymbol{r}) = \operatorname*{argmax}_{\boldsymbol{c}\in\boldsymbol{C}(\boldsymbol{k},\boldsymbol{r})} F(\boldsymbol{c}) \quad (4)$$

$$F(\boldsymbol{c}) = \prod_{j=1}^{l(c)} \phi^2(c_j.k, c_j.r) \quad (5)$$

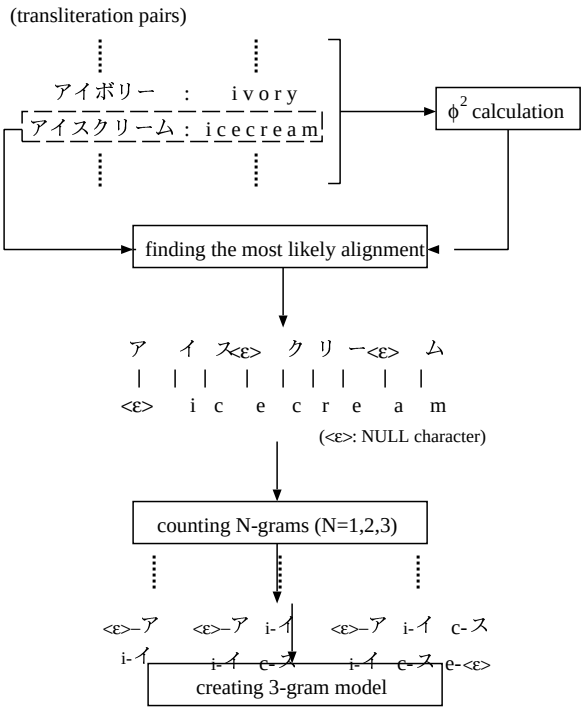

Figure 3. Training procedure of a transliteration model.

ϕ^2 is a correspondence measure between the two symbols [8]:

$$\phi^2(\sigma_k, \sigma_r) = \frac{(AD - BC)^2}{(A+B)(A+C)(B+D)(C+D)} \quad (6)$$

where

$$\begin{aligned} A &= \text{freq}(\sigma_k, \sigma_r) \\ B &= \text{freq}(\sigma_k) - \text{freq}(\sigma_k, \sigma_r) \\ C &= \text{freq}(\sigma_r) - \text{freq}(\sigma_k, \sigma_r) \\ D &= N - A - B - C. \end{aligned} \quad (7)$$

$\text{freq}(\sigma_k)$ is the number of katakana words including the katakana character σ_k in the training set. $\text{freq}(\sigma_r)$ is the number of Roman letter words including the Roman letter σ_r in the training set. $\text{freq}(\sigma_k, \sigma_r)$ is the number of transliteration pairs including both σ_k and σ_r in the training set. N is the total number of transliteration pairs in the training set. The training procedure is illustrated by Figure 3.

The transliteration model in the current transliteration scorer was trained by using 19,856 transliteration pairs.

Table 3 shows the output of the back-transliteration submodule for the development dataset and the formal run dataset. In addition to the development dataset provided by the CLQA organizer, we also used one hundred in-house CLQA questions. Question numbers starting with 'I' indicate the in-house questions. The back-transliteration submodule was called for 11 unknown katakana words for the development dataset. Correct translations were returned for seven of these katakana words. Since the internet search engine returned no documents for オダネウ and マイモニデズ, the submodule also returned no English words for them. It returned an incorrect translation for ピュンダー. For this word, the internet search engine returned no documents that contained the correct spelling 'Punder.' Therefore, a phonetically inappropriate candidate 'index' was selected.

Since our transliteration method does not generate candidates but simply collects Roman letter sequences from retrieved web documents, phonetically inappropriate candidates are sometimes returned. It must be possible to reject the best candidate when its penalty is too large.

For the formal run, the submodule was called for six katakana words. The submodule returned correct translations for three words (パテック, デ, and スーパーカミオカンデ), and returned only one incorrect translation (ネッタイイエカ). Since ネッタイイエカ is not a transliteration of a foreign word, we cannot expect the back-transliteration submodule to return the correct answer 'tropical culex mosquitoes.' Although the submodule returned no translations for ノムラ and スカイビズ for the formal run, it returned

Q	katakana	output	English	OK?
S0126	ドッカー	Docker	Docker	OK
S0196	サモア	SAMOA	Samoa	OK
S0207	ピュンダー	index	Punder	NG
S0218	オダネウ	—	Odunewu	NG
S0222	ムケシュ	Mukesh	Mukesh	OK
S0222	アフジャ	Ahuja	Ahuja	OK
S0238	マイモニデズ	—	Maimonides	NG
S0244	ヌーナー	Nooner	Nooner	OK
I0013	ソニーコンピュータエンタテインメント	Sony Computer Entertainment	Sony Computer Entertainment	OK
I0069	バイオトープ	biotope	biotope	OK
I0092	バファローズ	Buffaloes	Buffaloes	OK
T0091	パテック	Patek	Patek	OK
T0132	デ	de	de	OK
T0157	ネッタイイエカ	nutria	tropical culex mosquitoes	NG
T0180	スカイビズ	skybiz*	skybiz	OK*
T0194	スーパーカミオカンデ	Super Kamiokande	Super-Kamiokande	OK
T0199	ノムラ	Nomura*	Nomura	OK*

Table 3. Output of the back-transliteration submodule (*In the formal run, the system failed to obtain the output of the internet search engine because of the network status.)

correct translations when the first author was writing this working note. According to the execution log of the formal run, the system failed to obtain the output of the internet search engine. Perhaps, the system would have succeeded if it had retried. Since this submodule depends on the web, its output depends on the network status and changes from time to time.

4 Answer Extraction/Evaluation Module

4.1 Extraction

As we mentioned earlier, SVM-based English NE can cover only a small number of answer types. In order to cover the dozens of answer types we need for QA, we developed a hand-crafted rule-based NE recognizer. Here, we used only the hand-crafted recognizer for CLQA-1. The hand-crafted recognizer has 669 rules for 143 answer types.

Figure 4 shows two NE rules. The first rule says "if a sequence of proper nouns (NPs) are preceded by '`wife of`,' the sequence should be tagged with `<MALE>`." It matches a word sequence such as "`wife of Bill Clinton`" and outputs "`wife of <MALE>Bill Clinton`." The second rule matches a word sequence such as "`Texas Governor George Bush`" and outputs "`<STATE>Texas <PTITLE>Governor <PERSON>George Bush`." Here, `stategov` is a predefined macro for WordNet's synset offset

```
wife of <MALE>{_,NP,_}+ -> priority(4).

<STATE>_CAP_,NP,[stategov] <PTITLE>Governor
   <PERSON>{_,NP,_}+ -> priority(9).
```

Figure 4. Examples of hand-crafted NE rules

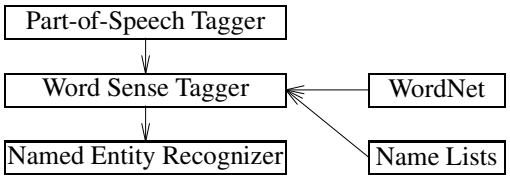

Figure 5. Answer extraction module

07130102 that means 'state or province' [9]. `_CAP_` stands for a capitalized word.

These rules are compiled and applied by a left-to-right longest match method. The priorities given in the rules are used when two or more rules have the same span. The pattern matching engine is an extension of our old rule-based Japanese NE system [10].

These NE rules depend heavily on word senses. In addition, we sometimes want to treat a phrase that is composed of two or more words as a single entity. For instance, the second rule should be applicable to a governor of North Carolina. Therefore, a *word sense tagger* is applied before the NE recognizer (Figure 5). The word sense tagger chunks words and adds word senses to each noun phrase by using WordNet and additional name lists (76,555 phrases) obtained from the web. Since the Daily Yomiuri newspaper has many romanized Japanese names, we added romanized Japanese name lists to the tagger. Figure 6 shows an example output of the word sense tagger. This document contains `New` as the 1,598th word and `Zealand` as the 1,599th word. They appear in the 54th sentence. Since they form a single country name, they are chunked as `New_Zealand`, and its WordNet synset offsets (`00001742` etc.) are given. The word sense tagger was implemented by using a simplified version of the aforementioned left-to-right longest pattern matcher.

4.2 Evaluation

We followed SAIQA's answer evaluation method [1]. Candidate c in a document D is evaluated by a weighted Hanning window function defined as follows:

$$\text{score}(c, D) = \sum_{q \in Q} idf[q] H(d(c, q)),$$

```
54-1589-1589 Jane NP 99999998
54-1590-1590 Hunter NP 00001742/00002664/000
54-1591 ,       ,       ,
54-1592-1592 owner NN 08514464/08116255/0811
54-1593 of      IN      of
54-1594-1594 Hunter NP 00001742/00002664/000
54-1595 's      POS     's
54-1596-1596 Winery NN 04002319/02797084/034
54-1597 in      IN      in
54-1598-1599 New_Zealand NP 00001742/0001306
```

Figure 6. Output of word sense tagger

$d(c, q)$ is the distance between c and the nearest position of a query term q, and

$$H(d) = \begin{cases} \frac{1}{2}(\cos(\pi\, d/W) + 1) & \text{if } 0 \le d \le W, \\ 0 & \text{otherwise.} \end{cases}$$

In SAIQA, we multiplied the candidate score by document D's score $DS(D)$, but we removed this multiplication here because $DS(D)$ did not improve SAIQA's performance at all and it degraded SAIQA-J/E's performance for the development dataset. We used $W = 60$ following SAIQA.

The evaluation module uses the output of the retrieval engine shown in Table 1 to calculate the score, and takes synonyms into consideration.

5 Results

Table 4 compares the performance of SAIQA-J/E (CLQA) and SAIQA-e (EQA). According to this table, CLQA's performance is comparable to that of EQA. In the strict evaluation, CLQA obtained better MRR than EQA. This is not a surprising result, because we did not spend much time on debugging the English question analysis module before the NTCIR CLQA-1 formal run. The Japanese question analysis module returned correct answer types for 95% of the given questions. On the other hand, the English question analysis module succeeded for only 73% of the questions. Therefore, we debugged only the English question analysis module. EQA2 in Table 4 shows the improved EQA system. Although the English question analyzer is still worse than the Japanese question analyzer, EQA2's performance is better than CLQA's.

Table 5 shows the performance of the document retrieval module. p@r is a standard measure, namely 'precision' or the average ratio of 'relevant' documents within top r documents. Here, a 'relevant' document is a document that contains a correct answer with or without sufficient evidence. a@r is the ratio of 'answerable' questions that has at least one relevant document in the top r documents. According to the table, p@r is very low for larger r values, but this simply means that only a small number of documents contain correct answers. A high p@r for a small r implies the effectiveness of our search engine.

		CLQA	EQA	EQA2
Development set				
	Top1	0.377	0.473	0.497
lenient	MRR	0.454	0.536	0.562
	Top5	0.580	0.617	0.650
Official run				
	ans type	0.950	0.730	0.875
	Top1	0.300	0.300	0.370
strict	MRR	0.376	0.359	0.440
	Top5	0.490	0.445	0.540
	Top1	0.315	0.350	0.425
lenient	MRR	0.420	0.432	0.518
	Top5	0.585	0.550	0.650

Table 4. Performance of CLQA and EQA (automatic evaluation)

	r	CLQA	without or2	EQA2
	1	0.520	0.450	0.805
	3	0.312	0.282	0.437
p@r	10	0.174	0.152	0.224
	20	0.122	0.105	0.152
	50	0.082	0.070	0.097
	1	0.520	0.450	0.805
	3	0.660	0.590	0.890
a@r	10	0.765	0.690	0.910
	20	0.805	0.745	0.915
	50	0.835	0.800	0.915

Table 5. Performance of the document retrieval module for the official run (automatic evaluation)

	CLQA	without or2
Top1	0.377	0.317
MRR	0.454	0.377
Top5	0.580	0.567

Table 6. Effectiveness of `or2` for the development set (automatic evaluation)

EQA(2)'s retrieval module performed much better than CLQA's. Our proximity-based document retrieval engine ranked relevent documents as the best for 80.5% of the official run questions. This implies that the official run questions are relatively easy as monolingual QA. On the other hand, the same engine returned relevant documents as the best for only 52.0% of the Japanese questions. When we used `or` instead of `or2`, the rate dropped to 45.0%. This result shows the usefulness of the synonym operator.

Table 6 compares the performance of the entire CLQA system with that of the system without `or2`. According to the table, the use of '`or`' instead of '`or2`' greatly degrades the system's MRR. This result also shows the effectiveness of `or2`. However, Top5 did not degrade very much.

English question analysis appears to be more difficult than Japanese question analysis because WordNet gives unusual word senses. For example, many English words such as 'tiger' and 'planet' are used to represent people. English question analyzers have to disambiguate such words. On the other hand, we rarely encounter such ambiguous cases in Japanese question analysis.

6 Conclusion

We developed a Japanese-English Cross-Language Question Answering system that performed best among eight systems in an NTCIR CLQA JE task. According to our experiments, the introduction of a synonym operator significantly improved the retrieval performance and the overall QA performance. Our system employed a web-based back-transliteration system for unknown words.

References

[1] H. Isozaki, "An analysis of a high performance Japanese question answering system (to appear)," *ACM Transaction on Asian Language Processing*, 2005.

[2] S. Ikehara, M. Miyazaki, S. Shirai, A. Yokoo, H. Nakaiwa, K. Ogura, Y. Ooyama, and Y. Hayashi, *Goi-Taikei — A Japanese Lexicon (in Japanese)*. Iwanami Shoten, 1997.

[3] H. Kazawa, H. Isozaki, and E. Maeda, "NTT's question answering system in TREC 2001," in *Notebook of The Tenth Text Retrieval Conference*, pp. 415–422, 2001.

[4] H. Isozaki and H. Kazawa, "Efficient support vector classifiers for named entity recognition," in *Proceedings of COLING-2002*, pp. 390–396, 2002.

[5] H. Isozaki, "NTT's question answering system for NTCIR QAC2," in *Working Notes of the Fourth NTCIR Workshop Meeting*, pp. 326–332, 2004.

[6] A. Pirkola, "The effects of query structure and dictionary setups in dictionary-based cross-language information retrieval," in *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 55–63, 1998.

[7] K. Tsuji, S. Sato, and K. Kageura, "Evaluation of the effectiveness of transliteration and a search engine for person names (in Japanese)," in *Proceedings of the Eleventh Annual Meeting of the Association for Natural Language Processing*, pp. 352–355, 2005.

[8] W. A. Gale and K. W. Church, "Identifying word correspondences in parallel texts," in *Proceedings of 4th DARPA Workshop on Speech and Natural Language*, pp. 152–157, 1991.

[9] C. Fellbaum, ed., *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[10] H. Isozaki, "Japanese named entity recognition based on a simple rule generator and decision tree learning," in *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pp. 306–313, 2001.