Oki QA System for QAC-2

Masachika Fuchigami(*1) Hiroyuki Ohnuma(*1) Atsushi Ikeno(*2)
(*1) Human Interface Laboratory,
(*2) Ubiquitous System Laboratory,
Corporate R&D Center
Oki Electric Industry Co., Ltd.
(*1)550-5 Higashi-Asakawamachi Hachioji 193-0514, Japan
(*2)2-5-7 Hommachi, Chuo-ku, Osaka 541-0053, Japan
{fuchigami636,ohnuma838,ikeno546}@oki.com

Abstract

This note describes OKI QA system for QAC-2 of NTCIR4. Our system has three characteristics: dependency-matching based answer extraction, run-time tagging, and learning-based query type analyzer. The result of the experiment implies that dependency revision technique will occupy an important position.

Keywords: *Dependency based matching, QAC, Question Answering,, Information Extraction*

1. Introduction

Our target in QAC-2 is precise responses however we pursued the system that returns a swift response in QAC-1[1].

Before we discuss our new system, we reconsider the two problems in our QAC-1 system.

The former problem is that IR module and question analysis module adopts different word unit. This inconsistency brings a lot of IR failures because the IR module is not able to find articles for the keywords given by the question analysis module. We resolve the inconsistency problem by using ChaSen[3] for both module and transferring whole query sentence to article retriever subsystem.

The latter problem is poor scoring algorithm. The algorithm gives high score to the NEs that happen to exist close to keywords. We concentrated mainly on improvements of the answer extractor subsystem in our QAC-2 system.

We describe our QA System for QAC-2 in the rest of this paper.

Our system has the following characteristics, dependency-matching based answer extraction, run-time tagging and learning-based query type analyzer.

First, dependency-matching based answer extraction avoids choosing incorrect answers that happen to exist close to keywords.

Second, run-time tagging enables our system to apply open document set, e.g. the Internet web documents.

Finally, learning-based query type analyzer liberates us from endless matching pattern authoring.

Section 2 shows overview of our system, and its subsystems are detailed in section 3 through 6. We describe the results of experiments in section 7, and evaluate them in section 8. Section 9 is conclusion.

2. System overview

Figure 1 shows block diagram for our system. Our system consists of the following subsystems: query analyzer, article retriever, answer extractor, filter/formatter and NE tagger.

Query analyzer subsystem parses query sentences and decides answer NE types.

Article retriever subsystem retrieves appropriate articles. We utilize GETA[2] as article retriever.

Answer extractor subsystem extracts answer candidates with their scores.

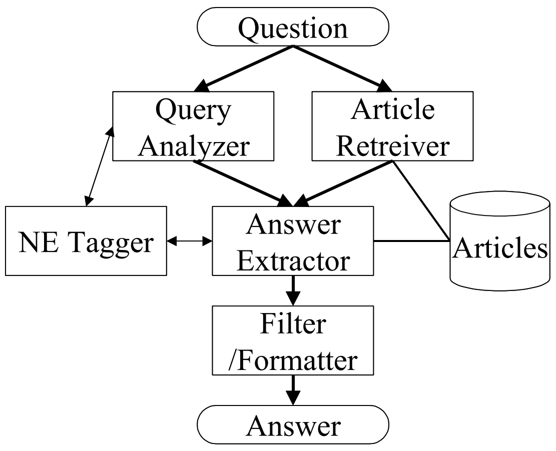

Figure 1: System diagram

Filter/formatter subsystem consists of two parts: answer filter and formatter. The former eliminates candidates of low score and chooses appropriate candidates on the borderline, and the latter formats the result in QAC-2 style.

3. Query analyzer subsystem

Our query analyzer subsystem adopts 1 nearest neighbour(1nn) algorithm in order to simplify the registration of query-types for sentences.

Figure 2 shows the structure of the query analyzer subsystem. The subsystem consists of two parts: register and searcher.

The former part constructs a DB in advance. The DB contains the five query factors described later and the query type. In QAC-2, we construct the DB with 1679 query and its query-type data set. The data set consists of 50 queries form QAC-1 dry run, 200 queries from QAC-1 formal run subtask1/2, 761 queries from QAC-1 additional subtasks, and original 718 queries.

The latter part searches the most similar query sentence in the DB and returns its query-type. In order to retrieve the most similar sentence, similarity between sentences is required. Similarity is calculated from five factors (examples in Table 1): type of wh-NE, words prepended/appended to wh-NE, verbs, topic words/phrases and words depended by wh-NE. 'Wh-NE' means a NE whose type is WH, e.g. "何"(what),"どこ"(where), "何年"(how many years).

After the query-type is retrieved, the subsystem outputs the query-type, tagged NEs and dependencies.

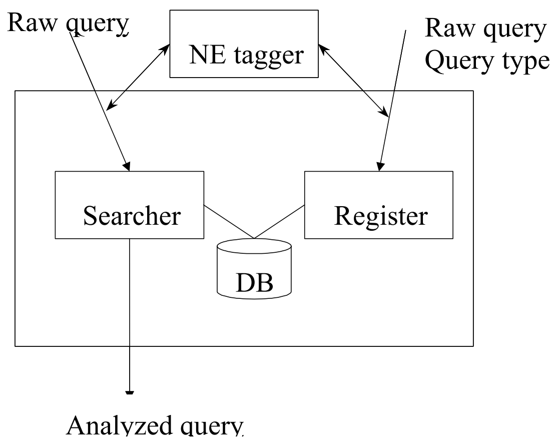

Figure 2: Query analyzer

(1)新幹線は時速何 km? (What km/h the shinkansen runs?)
(2) どこの会社が○○を開発したのですか (Which company developed **?)

	(1)	(2)
Wh-NE	何(what)	どこ(where)
prepend/append word(s)	時速(per hour), km	-
depending word(s)	-	会社 (company)
topic(s)	新幹線 (Shinkansen)	-
verb(s)	-	開発 (develop)

Table 1: Example of similarity factors

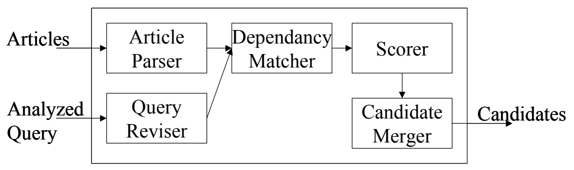

Figure 3: Answer extractor

4. Answer extractor subsystem

Figure 3 shows answer extractor subsystem.

It consists of following 5 parts: article parser, query reviser, dependency matcher, scorer, and candidate merger.

Details of all these parts are described in following subsections.

4.1 Article parser

Article parser parses the article supplied by the article retriever subsystem.

The parser extracts NEs and dependencies using NE tagger. After the extraction, this part revises the following sequence of NEs and adds corresponding dependencies in order to adapt to the variation of expressions.

- verb (or adjective) - noun
- noun-particle "の"(NO)-noun
- noun "ARTIFACT"

The first verb/adjective - noun pattern, e.g. "住んでいる人"(dwelling person), is revised to noun-[unspecified]-verb or noun-[が(GA)]-adjective, for example "人"(person)-[unspecified]-"住んでいる"(dwell).

In the second pattern, noun-[の(NO)]-verb, the particle "の"(NO) is considered to be equivalent to the particle "が"(GA) when the pattern appears in a subordinate clause.

For example, "私の言っ(たこと)"((what) I said) is equivalent to "私が言っ(たこと)". The parser therefore adds Noun-[の(NO)]-verb pattern.

The last pattern noun "ARTIFACT", e.g. "夏目漱石「草枕」"(Natsume Souseki "Kusamakura"), is revised to noun-[の]-ARTIFACT in order to match with noun-[の]- (noun with wh-flag:described in 4.2), e.g. "夏目漱石の名作(は何)"((what is) Natsume Souseki's masterpieces).

For example, the example shown above is revised to "夏目漱石"-[の]-"「草枕」"(Natsume Souseki's "Kusamakura").

All these revisions are applied to the original dependencies. In other words, the added dependencies are not revised again in this step.

4.2 Query reviser

Query reviser preprocesses the analyzed query for the dependency matcher part in order to decrease loss of dependency-matching coverage caused by variations of query sentences.

The preprocessing consists of two tasks: flagging wh-NEs and dependency revising.

In the former task, wh-flags are set on NEs whose NE-type is WH, e.g. "何", "どこ", etc. The wh-flag means query target NE, therefore NEs matched to wh-flagged NEs in dependency matcher part(see section 4.3) become answer candidates.

In the latter task, the subsystem adds extra dependencies by revising sequences of NEs containing WH-type NE, in order to deal with the variation of expressions. The revision is based on pattern-matching. The subsystem searches pattern-matched rules and adds corresponding pseudo-dependencies to the query dependency list.

Table 2 shows some of the revision rules. We explain revising task using the first rule in table 1. Assume the input sentence contains "五輪の開催地は何処ですか"(Where is the site of the Olympics?). The NEs extracted from the sentence matches the pattern in the first rule(A="五輪", B="開催[1]").

Then the subsystem fills in NEs to corresponding dependencies. The dependencies become "五輪"-[unspecified]->"開催する" and WH-[で]->"開催する" The subsystem also set wh-flags to appropriate NEs specified in the rule. In this example, the subsystem sets wh-flag to the anonymous NE X in revision rule X2. In this case, though X is ordinarily represented as “どこ” in natural Japanese language, we do not need surface form because the wh-flagged NE X works as a wildcard NE(see 4.3).

[1] We regard *sahen* nouns as a kind of NE here.

Finally, the subsystem adds the dependencies X1 and X2 to the query dependency list.

Thus the system is now able to retrieve candidates from sentences in other expressions, for example, "五輪を長野で開催する"(take place the Olympics in Nagano).

Table 2: Revision pattern example

Pattern	Revision
A の B(さ変名詞)地は何処	**X1:**A-[unspecified]->B する
	X2:X(wh-flagged)-[で]->B する
A の B は誰	A-[の]-WH

4.3 Dependency matcher

Dependency matcher part compares dependencies in the query and ones in an article.

4.3.1 Matching algorithm

The subsystem compares dependencies in the article by each sentence from beginning of the article to the end of the article. Dependency comparison between query and the sentence in article consists of two steps. In the first step, dependencies in the query sentence are compared to ones in the sentence, and then in the second step the subsystem searches preceding sentences in the article for dependencies that match remaining query dependencies in the first step, in order to resolve zero anaphora.

The subsystem classifies matchness into one of five results: fully-matched, particleless-matched, negative-matched, context-matched and not-matched.

fully-match: dependers and dependents matches and particles are matched and specified.

particleless-match: dependers and dependents matches, but one or both particles are unspecified.

For example, "6月"-[unspecified]-"発表する" particleless-matches to "6月"-[に]-"発表する".

negative-match: dependers differs, and particles are matched and specified.

context-match: fully-match or particleless-match in previous sentences in the article. This matchness substitutes zero anaphora analyzing. See an example described later.

not-match: otherwise.

Note that wh-flagged NEs matches any NEs e.g. "誰"(Who:wh-flagged)-[が]-"書く"(write) fully matches "シェイクスピア"(Shakespeare)-[が]-"書く".

4.3.2 Matching example

Examples of matching are shown in figure 4. Assume query sentence is `SQ`, sentences in the article are S1 and S2. They are parsed as the dependencies `DQ1` - `DQ3`, `D11` - `D22`.

First, S1 and SQ are compared (the upper half of the figure). Its result in the first step is [fully-match: DQ1 and D12], and no result in the second step because S1 has no preceding sentence.

Then S2 and SQ are compared (the lower half of the figure). In the first step the system finds [fully-match: DQ2 and D21] and [particleless-match: DQ3 and D22](shown as solid connectors) and DQ1 remains unmatched. In the second step, the subsystem searches the preceding sentences (i.e. only S1 for S2), for the dependencies that fully- or particleless-match with DQ1. D12 fully-matches DQ1, therefore the result in the second step becomes [context-match: DQ1 and D12](shown as dotted connector).

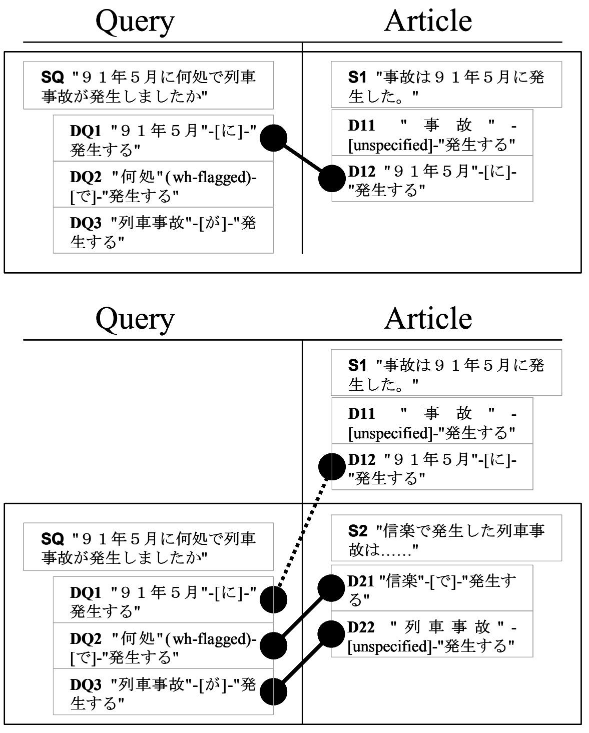
Query
Article
SQ "９１年５月に何処で列車事故が発生しましたか"
DQ1 "９１年５月"-[に]-"発生する"
DQ2 "何処"(wh-flagged)-[で]-"発生する"
DQ3 "列車事故"-[が]-"発生する"
S1 "事故は９１年５月に発生した。"
D11 "事故"-[unspecified]-"発生する"
D12 "９１年５月"-[に]-"発生する"
Query
Article
S1 "事故は９１年５月に発生した。"
D11 "事故"-[unspecified]-"発生する"
D12 "９１年５月"-[に]-"発生する"
SQ "９１年５月に何処で列車事故が発生しましたか"
DQ1 "９１年５月"-[に]-"発生する"
DQ2 "何処"(wh-flagged)-[で]-"発生する"
DQ3 "列車事故"-[が]-"発生する"
S2 "信楽で発生した列車事故は……"
D21 "信楽"-[で]-"発生する"
D22 "列車事故"-[unspecified]-"発生する"

Figure 4: Matching example

4.4 Scorer

Scorer part scores each dependency retrieved in dependency matcher part and sums up the scores.

The scoring consists of 4 steps: dependency scoring, NE scoring, inner-article scoring, inter-article scoring.

The first scoring measures the dependencies' matchness. The subsystem scores dependencies according to Table 3.

The second scoring measures the NE's score. The score is:

$$neScore = \max(0, \sum_{n \in sentence} depScore(n)) \times typeMatch(T_{ne}, T_{query})$$

(if the NE matches wh-flagged NE)

$neScore = 0$ (otherwise)

T_{query}: query type calculated in query analyzer subsystem.

T_{ne}: NE-type of the NE

depScore: dependency score

typeMatch is lookup-table function, whose range is between 0.0 and 1.0 each ends inclusive. Table 4 shows examples of the table used in *typeMatch*.

The system adds the NE to answer candidates list if the NE's score is more than zero.

The third scoring, i.e. inner-article scoring, merges the scores of the same surface form NEs in the article. The inner-article score for a candidate is $\max(neScore(n))$ where n is a member of the same surface form NE set.

The last scoring, i.e. inter-article scoring is simply the sum of the inner-article score for each candidate.

Table 3: Dependency score

-2.0	negative-match
+1.1	fully-match
+0.5	particleless-match
+0.4	context-match

Table 4: Type matchness table

T_{query}	T_{ne}	value
PS(*1)	PS	1.0
PS	PS_LASTNAME	0.9
PS	PS_FIRSTNAME	0.91
PS	PS_WORD(*2)	0.2
PS	ARTIFACT	0.01
	:	
ANIMAL	INSECT	0.2
ANIMAL	REPTILE	0.8
ANIMAL	AMPHIBIA	0.8
PLANT	PLANT	1.0
PLANT	VEGITABLE	0.9
PLANT	FLUIT	0.9
	:	

(*1) person
(*2) words which indicate person, e.g. “社長”.

4.5 Candidate merger

Candidate merger part aggregates candidates in order to avoid longer NEs.

The part merges candidates *C1* and *C2* into *C1* adding *C2*'s score if one of the following conditions is satisfied:

- *C1* is prefix or suffix of *C2,* and *C1* is longer than four letters.
- *C1* is prefix of *C2,* and post-*C1* letter in *C2* is "・"(center dot).
- *C1* is suffix of *C2,* and pre-*C1* letter in *C2* is "・"(center dot).
- *C1* is infix of *C2,* and both pre- and post-*C1* letters in *C2* are "・"(center dot).

For example, C1="ヴァン・ゴッホ"(score=0.7) and C2="バミューダ島のヴァン・ゴッホ"(score=0.4) are merged into "ヴァン・ゴッホ"(score=1.1).

5. Filter subsystem

The filter subsystem eliminates inappropriate answer candidate NEs and limits number of answers that is required in subtask 1.

Our filter subsystem has two parts: low-score candidate eliminator and candidate on border line chooser.

The low-score candidate eliminator chooses top-N candidates from all the candidates.

In subtask-1, N is five. In subtask-2 and subtask-3 we fix N to three in order to avoid loss of precision.

Then the score of the candidates ranked in M-th order are compared to that of the (M-1)-th candidates. If the score for M-th candidates is less than half the score of the (M-1)-th candidates, the filter subsystem removes M-th and succeeding candidates.

The candidate on borderline chooser selects appropriate candidate among the same-scored candidates in the lowest rank. The subsystem orders the candidates by the following rules applied sequentially.

1. If the query contains "日本", non-alphabetical candidate wins; otherwise alphabetical candidate wins.
2. If the query contains "日本", non-katakana candidate wins; otherwise katakana candidate wins.
3. If the length of one candidate is equal to or more than 4 letters, the candidate wins.
4. If the length of every candidate is less than 4 letters, longer candidate wins.
5. Shorter candidate wins.
6. The candidate listed first wins.

After ordering, the subsystem trims the candidates until the number of them becomes N.

6. NE tagger

The NE tagger is used in the query analyze subsystem and the article parser.

Details are described in [4].

7. Experiments and results

In this section we describe the results and its evaluation. We first describe overall summary, and then we describe each subsystem evaluation. We run the experiments on a PC Linux system with 600MHz IA-32 cpu. We utilize the QAC2 subtask-1 query set in the experiments.

7.1 Result summary

Table 5 shows the result for each subsystem.

In this table, precision means the ratio of number of answer sets that contains correct subsystem answer / number of queries. The precisions are calculated as follows:

- **Query analyzer:** correct query type sets / all queries. We mark the answers by ourselves because no formal answer set is provided.
- **Article retriever:** result article set which contains correct articles for answer source / all queries. The correct article sets are taken from the source articles of correct answers in the answer data set given by the QAC-2 organizer.
- **Answer extractor:** answer candidates which contains correct NEs and correct source articles in top-5 candidates including the sixth or lower one whose score is equal to the fifth's one / queries that both query analyzer and article retriever return correct answer. The correct NE and its source data set are taken from the answer data set given by the QAC-2 organizer.
- **Filter:** answers that contains correct NEs / answer candidates which contains correct NEs in top-5 candidate including sixth or lower one whose score is equal to the fifth's one.

Table 5: Failure summary

Subsystem	Precision		time(sec)
Query analyzer	78%	156/200	483
Article retriever	87%	173/200	665
(both results correct)	67%	133/200	
Answer extractor	38%	50/133	4482
Filter	92%	46/50	159
(Total)	27%	53/200*	5789

* includes QAs that the query analyzer returns incorrect answer type, the answer extractor nevertheless returns correct answer.

The result shows that we should improve the precision and the speed of answer extractor subsystem. Consideration to the precision of the extractor is given in the later section.

7.2 Query analyzer evaluation

Table 6 shows reasons that the query analyzer fails. We think that the failures are caused by insufficiency of the learning data for type decision.

Table 6: Reasons of failure in Query analyzer

Incorrect type	23
Too many types	13
No applicable type	3
Supertype	4
Other	1
(Total)	44

The reasons are:

Incorrect type: The analyzer returns non-suitable query type. A common error is that LOCATION type is returned while correct answer is ORGANIZATION type.

Too many types: The analyzer returns more than 3 types. This failure is caused by lack of learning data that is similar to the query in these cases.

No applicable type: The system does not have appropriate NE type. We can reduce this failure by adding appropriate NE type for the tagger, by adding learning data to query analyzer, and by adding matcheness into type matcheness table (table 4)

Supertype: The analyzer returns more generic type, for example NUMBER type instead of LENGTH.

7.3 Answer extractor evaluation

Some of the considerable extraction failure reasons are shown in Table 7.

More than half of the failures are caused in dependency matching part. We think that improving more efficient dependency analyzer and introducing more dependency revision rules reduce these failures.

Table 7: Reasons of failure in Answer extractor

NE not recognized	4
Dependency not match	50
Scoring	15
Incorrect merge	3
Incorrect authority	10
Other	1
(Total)	83

We details each failure below:

NE not recognized: (Strictly, this failure is not caused in answer extractor.) The NE tagger fails to retrieve the answer NE. Improving the NE tagger will reduce this type of failure.

Dependency not match: (A) The parser fails to extract dependencies that contain the correct answer or (B) the dependencies that contain correct answer do not match to the dependencies in the query or revised ones. (A) is caused by following two reasons: unsupported dependencies and dependency analyzer precision. An example for the former is "何大学" (which university). This example does not have obvious dependency. In this case, our solution is revising. For the example, we will resolve the problem by adding a rule "何 ORG" to X(wh-flagged)-[の]-ORG.

(B) is caused mainly by insufficiency of revision rules and by lack of synonym unification.

Scoring: Although the correct answer is listed in the candidate list before filtering (section 5), the filter drops correct answer because of its score or because the borderline choice selects incorrect answer. The cause is mainly dependency analyzer, not scorer, because only few dependencies in the query are analyzed. Then they produce candidates of the same score because the factors to the score decrease. Finally, the correct answer is buried into incorrect answers of the same score.

Incorrect merge: Candidate merging (see 4.5) produces incorrect answer. For example, the candidate merger merges "１・２メートル"(1.2 meters) and "２メートル"(2 meters) incorrectly. This failure will be removed by optimizing the rules in the merger.

Incorrect authority: The selected article that authorizes the answer is incorrect while answer NE is correct. The cause is that the authority selection does not consider any scores of NE and article. Therefore selection with scores will enhance the system.

7.4 Query reviser evaluation

In order to prove our revision technique improves precision, we examined the effect of the query reviser described in section 4.2. We run the system for QAC2 formal-run query set with turning on and off the query reviser. Table 8 shows the result. In this table, precision means the ratio of number of answer sets that contains correct answer / number of queries. The result indicates that our revision technique works effectively.

Table 8: Query revision effect

	Precision
With reviser	32.5% (65/200)
Without reviser	28.5% (57/200)

*Precisions differ to the ones in the table 4 because the results are measured in updated system after the formal-run..

8. Remaining problems and future work

The experiments described in the above section make it clear that we should improve the following points: dependency revision rules in answer extractor, learning more queries in query analyzer, and reconsideration to candidate mergence algorithm.

In addition to these improvements, our future work comprises:

- revision rule learning, in order to avoid too complex rule authoring.
- direct/indirect anaphora analyzing.
- robust NE-tagging and parsing for broken sentence, in order to accept generic documents, e.g. web pages.

9. Conclusion

We described our QA system for QAC-2.

The results of experiments indicate that our dependency-matching based system is not optimum for QACs yet. We find that the system leaves much room for improvement. We believe our system becomes more effective when we add more learning data for query analyzer or when we introduce more dependency revision rules.

References

[1] A. Ikeno, H. Ohnuma,
"Oki QA System for QAC-1"
In working notes of the third NTCIR workshop meeting

[2] A. Takano, S. Nishioka, O. Imaichi, M. Iwayama, Y. Niwa, T. Hisamitsu, M. Fujio, T. Tokunaga, M. Okumura, H. Mochizuki, T. Nomoto
"Development of the generic association engine for processing large corpora" (In Japanese)
http://geta.ex.nii.ac.jp/

[3] Y. Matsumoto, A. Kitauchi, T. Yamashita, Y. Hirano, H. Matsuda, K. Takaoka, M. Asahara,
"Japanese Morphological Analysis System ChaSen version 2.2.1"
http://chasen.aist-nara.ac.jp/

[4] H. Ohnuma, A. Ikeno.
"Answer Extraction of Question and Answering System on HTML Documents"
. In Proceedings of 63rd annual meeting of IPSJ, 3-41, 2001. (in Japanese)