A Corpus Search System Utilizing Lexical Dependency Structure

Yoshihide Kato*, Shigeki Matsubara†, Yasuyoshi Inagaki‡

* Graduate School of International Development, Nagoya University
† Information Technology Center, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464-8601 Japan
‡ Faculty of Information Science and Technology, Aichi Prefectural University
1522-3 Ibaragabasama, Kumabari, Nagakute-cho, Aichi-gun, 480-1198 Japan
yosihide@gsid.nagoya-u.ac.jp

Abstract

This paper presents a corpus search system utilizing lexical dependency structure. The user's query consists of a sequence of keywords. For a given query, the system automatically generates the dependency structure patterns which consist of keywords in the query, and returns the sentences whose dependency structures match the generated patterns. The dependency structure patterns are generated by using two operations: combining and interpolation, which utilize dependency structures in the searched corpus. The operations enable the system to generate only the dependency structure patterns that occur in the corpus. The system achieves simple and intuitive corpus search and it is enough linguistically sophisticated to utilize structural information.

1. Introduction

Large text corpora increasingly become important resources for linguistic research, development of natural language processing systems, language teaching, etc. Corpus search systems are necessary to utilize text corpora effectively.

Several corpus search systems have been presented. Most systems provide keyword-based search functionality. The search is simple and intuitive, but not enough linguistically sophisticated to utilize structural information.

On the other hand, (Corley et al., 2001) and (Resnik and Elkiss, 2005) have presented corpus search systems utilizing syntactic structure, Gsearch and Linguist's Search Engine (LSE), respectively. These systems can search corpora by using phrase structure patterns. In the Gsearch, the user gives a phrase structure pattern and a grammar to the system. The system constructs parse trees of the sentences in the corpus by using the given grammar, and returns the sentences whose parse trees match the given pattern. In the LSE, the user first gives an example of sentences which he/she needs. The system parses the example by using a statistical parser and returns the parsing result. The user edits the resulting parse tree to specify a structural query. The system finally returns the sentences whose parse trees match the structural query. The Gsearch and LSE can search corpora by utilizing syntactic information. However, they do not achieve simple search like keyword-based systems.

This paper presents a corpus search system which automatically generates structural queries from keyword-based queries. The system searches corpora based on lexical dependency information. The user's query is a sequence of keywords. For a given query, it generates dependency structure patterns by using two operations: combining and interpolation. The user need neither to build a grammar like the Gsearch nor to edit structural query like the LSE, because of the automatic pattern generation. The system achieves simple and intuitive corpus search and it is enough to linguistically sophisticated to utilize structural information.

2. Corpus Search based on Dependency Structure

This section presents a corpus search system based on dependency structure.

We assume that corpus sentences are annotated with dependency structures. The user's query consists of a sequence of keywords (words or POSs). For a given query, the system tries to generate dependency structure patterns and returns the sentences whose dependency structures match one of the generated patterns.

2.1. An Algorithm of Generating Dependency Structure Patterns

This section proposes an algorithm of generating dependency structure patterns. The inputs are as follows:

query: $q_1 \cdots q_m$ ($q_1, \ldots, q_m$ are keywords)

sentence: $s = w_1 \cdots w_n$ ($w_1 \ldots, w_n$ are pairs of words and POSs)

dependency structure (a set of dependencies): D

where D is a set of dependencies between words in s. If w_i depends on w_j, the pair of the positions (i, j) is a element of D. We write $i \rightarrow j$ for (i, j).

We define the dependency structure pattern as a 3-tuple $d = (h, L, R)$, where h is a word position and L and R are lists of dependency structure patterns. h is called the *head* of d. The dependency structure patterns d represents that the heads of dependency structure patterns in L depend on h from left. Similarly for R, from right.

Our proposed algorithm generates dependency structure patterns by using the following two operations: *combining* and *interpolation*.

combining: Let $d = (h, L, R)$ and $d' = (h', L', R')$ be dependency structure patterns for $q_i \cdots q_j$ and $q_{j+1} \cdots q_k$, respectively. If $h \rightarrow h'$ and $R' = \varepsilon$, then generate a dependency structure pattern $(h', d \cdot L', R')$ (see Fig 1a)). If $h' \rightarrow h$, then generate a dependency structure pattern $(h, L, R \cdot d')$ (see Fig 1b)).

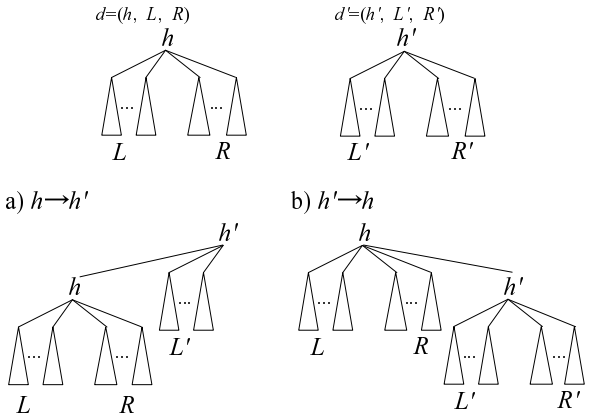

Figure 1: Combining

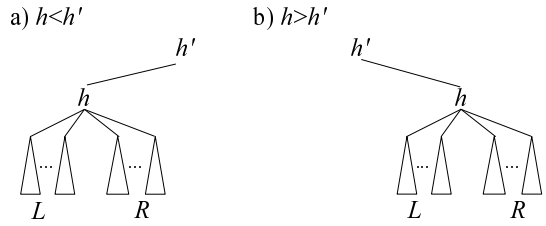

Figure 2: Interpolation

interpolation: Let d be a dependency structure pattern for $q_i \cdots q_j$ whose head is h. For h' such that $h \rightarrow h'$, if $h < h'$, then generate a dependency structure pattern (h'^*, d, ε) (see Fig. 2a)). If $h > h'$, then generate a dependency structure pattern (h'^*, ε, d) (see Fig. 2b)). A symbol $*$ means that h' is introduced by interpolation.

By applying the combining operation to the given query, all dependency structure patterns that directly connect keywords in the query can be generated. The generated patterns are guaranteed to match the dependency structure D, so the system returns the sentences for which some patterns are generated.

In some cases, the user may not intend that some keywords in the query directly depend on the other keywords. To process such queries robustly, we introduce the interpolation operation. This operation can generate the dependency structure patterns which include words not occurring in the query.

To avoid useless application of the operation, we introduce a cost defined as the number of occurrence of * in the dependency structure pattern. The algorithm does not generate the dependency structure patterns whose costs are greater than a threshold.

Figure 3 illustrates the algorithm of generating dependency structure patterns. θ is the threshold of cost. $D[i, j, c]$ is used for recording the dependency structure patterns with cost c for $q_{i+1} \cdots q_j$. $rm(d)$ and $lm(d')$ are the rightmost word position in d and the leftmost word position in d', respectively. These positions are used for checking the order of keywords in dependency structures patterns generated.

input: query $q_1 \cdots q_m$,
sentence $w_1 \cdots w_n$,
dependency structure D

initialization:
for $i = 1$ **to** m
  for each j s.t. $w_j = q_i$ **do**
    push $(j, \varepsilon, \varepsilon)$ to $D[i-1, i, 0]$;

for $cost = 0$ **to** θ
  combining:
  for $k = 2$ **to** m
    for $j = k-1$ **down to** 1
      for $i = j-1$ **down to** 0
        for $c = 0$ **to** $cost$
          for each $d = (h, L, R) \in D[i, j, c]$,
              $d' = (h', L', R') \in D[j, k, cost - c]$
              s.t. $rm(d) < lm(d')$ **do**
            if $h \rightarrow h' \in D \wedge R' = \varepsilon$ **then**
              push (h', dL', R') to $D[i, k, cost]$;
            if $h' \rightarrow h$ **then**
              push (h, L, Rd') to $D[i, k, cost]$;

  interpolation:
  for $j = 1$ **to** m
    for $i = j-1$ **down to** 0
      if $i \neq 0 \vee j \neq q$ **then**
        for each $d = (h, L, R) \in D[i, j, cost]$
          for h' s.t. $h \rightarrow h' \in D$
            if $h < h'$ **then**
              push (h'^*, d, ε) to $D[i, j, cost + 1]$;
            else
              push (h'^*, ε, d) to $D[i, j, cost + 1]$;

return $D[0, m, 0] \cup \cdots \cup D[0, m, cost]$;

Figure 3: An algorithm of generating dependency structure patterns

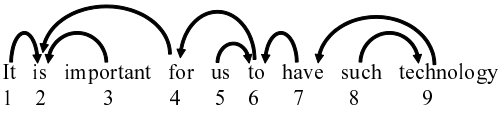

Figure 4: Dependencies for "It is important for us to have such technology"

2.1.1. An example of combining operation

Let us consider an example of generating dependency structure patterns for the following query:

$$\text{it is for to} \tag{1}$$

and the following sentence:

$$\text{It is important for us to have such technology.} \tag{2}$$

Assume that the dependencies are as illustrated in Figure 4.

The first keyword "it" matches the first word in sentence (2), so the following dependency structure pattern is generated:

$$(1, \varepsilon, \varepsilon) \tag{3}$$

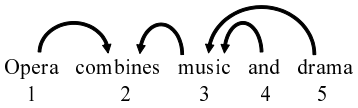

Figure 5: Dependencies for "Opera combines music and drama"

Similarly, the following dependency structure patterns are generated for keywords "is", "for" and "to", respectively:

$$(2, \varepsilon, \varepsilon) \tag{4}$$

$$(4, \varepsilon, \varepsilon) \tag{5}$$

$$(6, \varepsilon, \varepsilon) \tag{6}$$

For the heads of dependency structure patterns (3) and (4), the dependency $1 \rightarrow 2$ holds. Therefore, the following dependency structure pattern is generated:

$$(2, (1, \varepsilon, \varepsilon), \varepsilon) \tag{7}$$

Similarly, the following dependency structure pattern is generated for "for to":

$$(4, \varepsilon, (6, \varepsilon, \varepsilon)) \tag{8}$$

Furthermore, since $4 \rightarrow 2$, dependency structure patterns (7) and (8) are combined and the following dependency structure pattern is generated:

$$(2, (1, \varepsilon, \varepsilon), (4, \varepsilon, (6, \varepsilon, \varepsilon))) \tag{9}$$

This means that "it is for to" occur in sentence (2) in the form of dependency structure pattern (9).
On the other hand, for sentences in which some keywords do not depend directly on the others (for instance, "It is clear whether support for the proposal will be broad enough to a serious challenge"), the algorithm generates no dependency structure pattern.

2.1.2. An example of interpolation operation

Let us consider another example of dependency structure pattern generation. The query and the sentence are as follows:

$$\text{combines and} \tag{10}$$

$$\text{Opera combines music and drama} \tag{11}$$

The dependencies are as illustrated in Figure 5.
For "combines" and "and", the following dependency structure patterns are generated:

$$(2, \varepsilon, \varepsilon) \tag{12}$$

$$(4, \varepsilon, \varepsilon) \tag{13}$$

Since neither $2 \rightarrow 4$ nor $4 \rightarrow 2$, no dependency structure pattern is generated by the combining operation.
By applying the interpolation operation to dependency structure pattern (13), the following pattern is generated:

$$(3^*, \varepsilon, (4, \varepsilon, \varepsilon)) \tag{14}$$

Table 1: Precision and recall for a query "it is for to."

threshold θ	precision	recall
0	100.0% (17/17)	81.0% (17/21)
1	90.5% (19/21)	90.5% (19/21)
2	70.0% (21/30)	100.0% (21/21)
3	60.0% (21/35)	100.0% (21/21)
baseline	27.3% (21/77)	100.0% (21/21)

Since $3 \rightarrow 2$, the algorithm combines (12) and (14) to generate the dependency structure pattern:

$$(2, \varepsilon, (3^*, \varepsilon, (4, \varepsilon, \varepsilon))) \tag{15}$$

This example demonstrates that the interpolation operation allows the generation of the dependency structure pattern in which some keywords indirectly depend on the other keyword.

3. Implementation and Evaluation

The system is implemented in CMUCL [1]. The system provides a simple KWIC display of the result. The returned sentences are classified according to the dependency structure patterns matched. Figure 6 shows a screen shot of the system.
To evaluate the performance of our proposed system, we performed an experiment. We searched Penn Treebank (Marcus et al., 1993), which is annotated with phrase structures. The phrase structures in the corpus are converted to dependency structures by using the method in the literature (Collins, 1999).
We built several queries and assigned relevant sentences to each query manually. We measured the precision and recall of the search system for the cost threshold of 0 to 3. The precision and recall are defined as follows:

$$\textit{Precision} = \frac{\text{the number of relevant sentences returned}}{\text{the number of sentences returned}}$$

$$\textit{Recall} = \frac{\text{the number of relevant sentences returned}}{\text{the number of relevant sentences}}$$

Moreover, we measured the precision of the search which returns the sentences including all keywords in the query in the order. The recall of the search is always 100%. We call it the search baseline.

3.1. The result for a query "it is for to"

Let us consider the result for a query "it is for to." The precision and recall are shown in Table 1. When $\theta = 0$, the system achieves high precision and recall, and all the dependency structure patterns that the system generated are the same as the example of Section 2.1.1. The relevant sentences for which the system does not generate the dependency structure pattern with cost 0 have the dependency structures where "for" depends not on "is" but on a complement. However, these relevant sentences were able to be found by using the interpolation operation.

3.2. The result for a query "combine and"

Let us consider the result of a query "combine and." The precision and recall are shown in Table 2. No relevant sen-

[1] http://www.cons.org/cmucl/

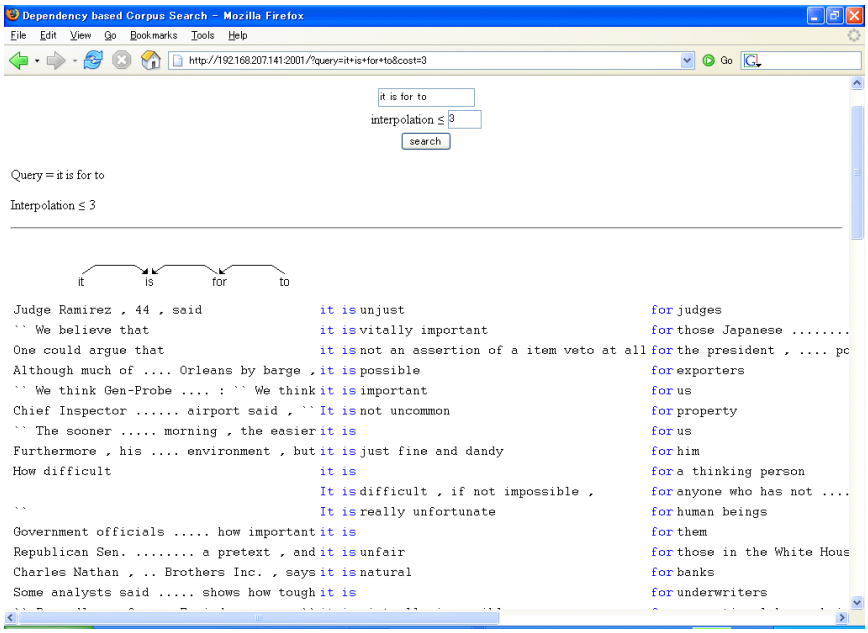

Figure 6: A screen shot of the system

Table 2: Precision and recall for a query "combine and."

threshold θ	precision	recall
0	0.0% (0/ 2)	0.0% (0/11)
1	55.0% (11/20)	100.0% (11/11)
2	42.3% (11/26)	100.0% (11/11)
3	32.4% (11/24)	100.0% (11/11)
baseline	24.5% (11/45)	100.0% (11/11)

Table 3: Precision and recall for a query "have `preposition` mind."

threshold θ	precision	recall
0	100.0% (10/10)	90.9% (10/11)
1	73.3% (11/15)	100.0% (11/11)
2	61.1% (11/18)	100.0% (11/11)
3	61.1% (11/18)	100.0% (11/11)
baseline	50.0% (11/22)	100.0% (11/11)

tence was found by using the dependency structure patterns with cost 0. However, all the relevant sentences are found by using the dependency structure patterns with cost 1. The result demonstrates that the interpolation operation is necessary for the system to process such queries robustly. The precision is not good, but we expect that the user can seek relevant sentences easily because the returned sentences are classified according to the patterns and relevant sentences are expected to belong to a few classes. In this example, the relevant sentences belonged to 2 classes.

3.3. The result for query "have `preposition` mind"

Let us consider the result for a query "have `preposition` mind". `preposition` is not a word but a part of speech. Table 3 shows the precision and recall. The system achieves high precision and recall for the query. Moreover, we can know the instance of `preposition` from the result (in this example, most words matching `preposition` were "in").

4. Conclusion

This paper presents a corpus search system based on dependency structure. The system automatically generates dependency structure patterns by utilizing corpus annotated with dependency structures so the user needs not to construct a structural query. The experimental results demonstrated that the system achieves high precision corpus search.

We would like to evaluate the system performance from the viewpoint of not only the precision and recall but also the usability.

Acknowledgments

This work is partially supported by the Grant-in-Aid for Young Scientists (B)(No. 17700145) of JSPS.

5. References

M. Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

S. Corley, M. Corley, F. Keller, M. Crocker, and S. Trewin. 2001. Finding syntactic structure in unparsed corpora: The gsearch corpus query system. *Computers and the Humanities*, 35:2:81–94.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: the penn treebank. *Computational Linguistics*, 19(2):310–330.

P. Resnik and A. Elkiss. 2005. The linguist's search engine: An overview. In *Proceedings of the 43rd ACL Interacrive Poseter and Demonstration Sessions*, pages 33–36.