Efficient sentence retrieval based on syntactic structure

Ichikawa Hiroshi, Hakoda Keita, Hashimoto Taiichi and **Tokunaga Takenobu**
Department of Computer Science, Tokyo Institute of Technology
{ichikawa,hokoda,taiichi,take}@cl.cs.titech.ac.jp

Abstract

This paper proposes an efficient method of sentence retrieval based on syntactic structure. Collins proposed Tree Kernel to calculate structural similarity. However, structual retrieval based on Tree Kernel is not practicable because the size of the index table by Tree Kernel becomes impractical. We propose more efficient algorithms approximating Tree Kernel: Tree Overlapping and Subpath Set. These algorithms are more efficient than Tree Kernel because indexing is possible with practical computation resources. The results of the experiments comparing these three algorithms showed that structural retrieval with Tree Overlapping and Subpath Set were faster than that with Tree Kernel by 100 times and 1,000 times respectively.

1 Introduction

Retrieving similar sentences has attracted much attention in recent years, and several methods have been already proposed. They are useful for many applications such as information retrieval and machine translation. Most of the methods are based on frequencies of surface information such as words and parts of speech. These methods might work well concerning similarity of topics or contents of sentences. Although the surface information of two sentences is similar, their syntactic structures can be completely different (Figure 1). If a translation system regards these sentences as similar, the translation would fail. This is because conventional retrieval techniques exploit only similarity of surface information such as words and parts-of-speech, but not more abstract information such as syntactic structures.

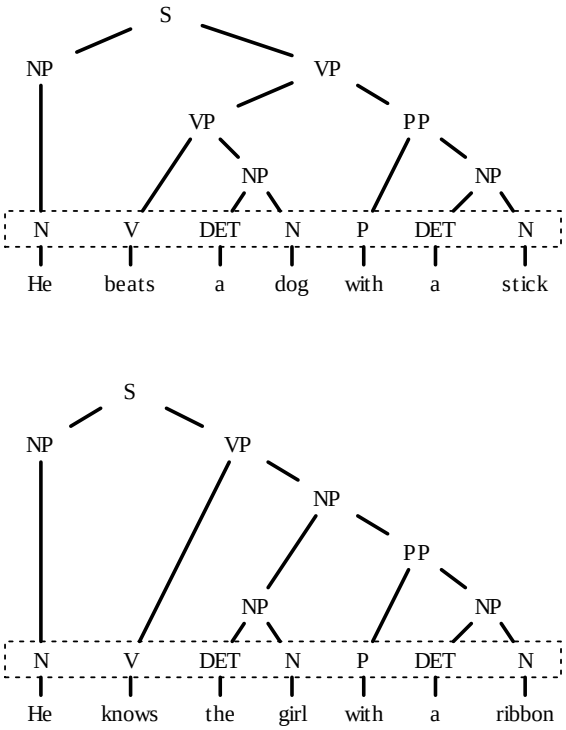

Figure 1: Sentences similar in appearance but differ in syntactic structure

Collins *et al.* (Collins, 2001a; Collins, 2001b) proposed Tree Kernel, a method to calculate a similarity between syntactic structures. Tree Kernel defines the similarity between two syntactic structures as the number of shared subtrees. Retrieving similar sentences in a huge corpus requires calculating the similarity between a given query and each of sentences in the corpus. Building an index table in advance could improve retrieval efficiency, but indexing with Tree Kernel is impractical due to the size of its index table.

In this paper, we propose two efficient algo-

Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions, pages 399–406,
Sydney, July 2006.

rithms to calculate similarity of syntactic structures: Tree Overlapping and Subpath Set. These algorithms are more efficient than Tree Kernel because it is possible to make an index table in reasonable size. The experiments comparing these three algorithms showed that Tree Overlapping is 100 times faster and Subpath Set is 1,000 times faster than Tree Kernel when being used for structural retrieval.

After briefly reviewing Tree Kernel in section 2, in what follows, we describe two algorithms in section 3 and 4. Section 5 describes experiments to compare these three algorithms and discussion on the results. Finally, we conclude the paper and look at the future direction of our research in section 6.

2 Tree Kernel

2.1 Definition of similarity

Tree Kernel is proposed by Collins *et al.* (Collins, 2001a; Collins, 2001b) as a method to calculate similarity between tree structures. Tree Kernel defines similarity between two trees as the number of shared subtrees. Subtree S of tree T is defined as any tree subsumed by T, and consisting of more than one node, and all child nodes are included if any.

Tree Kernel is not always suitable because the desired properties of similarity are different depending on applications. Takahashi *et al.* proposed three types of similarity based on Tree Kernel (Takahashi, 2002). We use one of the similarity measures (equation (1)) proposed by Takahashi *et al.*

$$K_C(T_1, T_2) = \max_{n_1 \in N_1,\ n_2 \in N_2} C(n_1, n_2) \qquad (1)$$

where $C(n_1, n_2)$ is the number of shared subtrees by two trees rooted at nodes n_1 and n_2.

2.2 Algorithm to calculate similarity

Collins *et al.* (Collins, 2001a; Collins, 2001b) proposed an efficient method to calculate Tree Kernel by using $C(n_1, n_2)$ as follows.

- If the productions at n_1 and n_2 are different $C(n_1, n_2) = 0$
- If the productions at n_1 and n_2 are the same, and n_1 and n_2 are pre-terminals, then $C(n_1, n_2) = 1$
- Else if the productions at n_1 and n_2 are the same and n_1 and n_2 are not pre-terminals,

$$C(n_1, n_2) = \prod_{i=1}^{nc(n_1)} (1 + C(ch(n_1, i), ch(n_2, i))) \qquad (2)$$

where $nc(n)$ is the number of children of node n and $ch(n, i)$ is the i'th child node of n. Equation (2) recursively calculates C on its child node, and calculating Cs in postorder avoids recalculation. Thus, the time complexity of $K_C(T_1, T_2)$ is $O(mn)$, where m and n are the numbers of nodes in T_1 and T_2 respectively.

2.3 Algorithm to retrieve sentences

Neither Collins nor Takahashi discussed retrieval algorithms using Tree Kernel. We use the following simple algorithm. First we calculate the similarity $K_C(T_1, T_2)$ between a query tree and every tree in the corpus and rank them in descending order of K_C.

Tree Kernel exploits all subtrees shared by trees. Therefore, it requires considerable amount of time in retrieval because similarity calculation must be performed for every pair of trees. To improve retrieval time, an index table can be used in general. However, indexing by all subtrees is difficult because a tree often includes millions of subtrees. For example, one sentence in Titech Corpus (Noro et al., 2005) with 22 words and 87 nodes includes 8,213,574,246 subtrees. The number of subtrees in a tree with N nodes is bounded above by 2^N.

3 Tree Overlapping

3.1 Definition of similarity

When putting an arbitrary node n_1 of tree T_1 on node n_2 of tree T_2, there might be the same production rule overlapping in T_1 and T_2. We define $C_{TO}(n_1, n_2)$ as the number of such overlapping production rules when n_1 overlaps n_2 (Figure 2).

We will define $C_{TO}(n_1, n_2)$ more precisely. First we define $L(n_1, n_2)$ of node n_1 of T_1 and node n_2 of T_2. $L(n_1, n_2)$ represents a set of pairs of nodes which overlap each other when putting n_1 on n_2. For example in Figure 2, $L(b_1^1, b_1^2) = \{(b_1^1, b_1^2), (d_1^1, d_1^2), (e_1^1, e_1^2), (g_1^1, g_1^2), (i_1^1, j_1^2)\}$.
$L(n_1, n_2)$ is defined as follows. Here n_i and m_i are nodes of tree T_i, $ch(n, i)$ is the i'th child of node n.

1. $(n_1, n_2) \in L(n_1, n_2)$

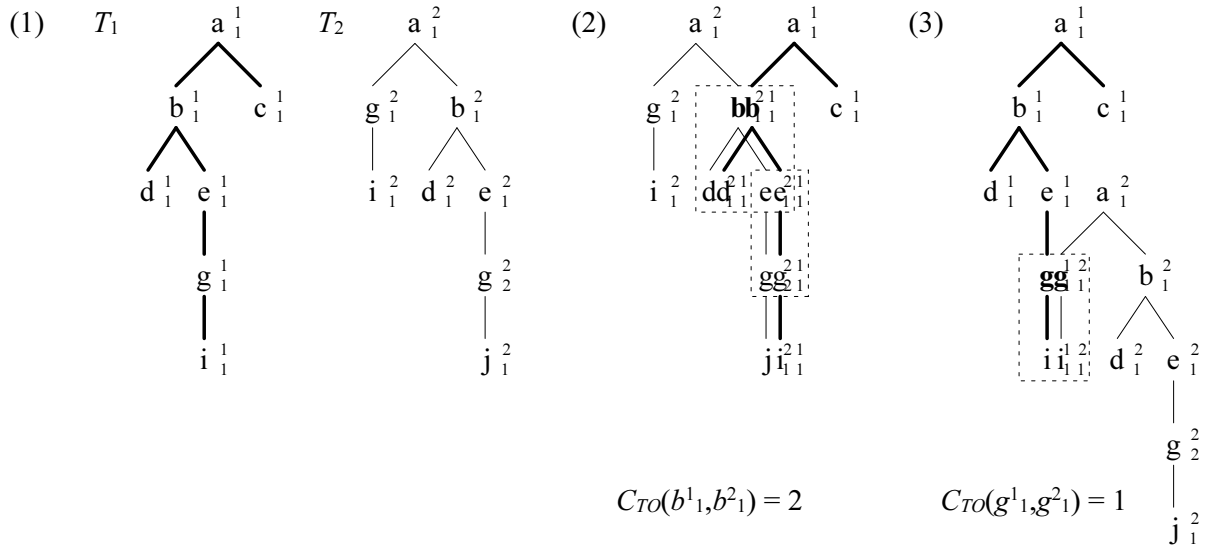

Figure 2: Example of similarity calculation

2. If $(m_1, m_2) \in L(n_1, n_2)$, $(ch(m_1, i), ch(m_2, i)) \in L(n_1, n_2)$

3. If $(ch(m_1, i), ch(m_2, i)) \in L(n_1, n_2)$, $(m_1, m_2) \in L(n_1, n_2)$

4. $L(n_1, n_2)$ includes only pairs generated by applying 2. and 3. recursively.

$C_{TO}(n_1, n_2)$ is defined by using $L(n_1, n_2)$ as follows.

$$C_{TO}(n_1, n_2) = \left| \left\{ (m_1, m_2) \;\middle|\; \begin{array}{l} m_1 \in NT(T_1) \\ \wedge\, m_2 \in NT(T_2) \\ \wedge\, (m_1, m_2) \in L(n_1, n_2) \\ \wedge\, PR(m_1) = PR(m_2) \end{array} \right\} \right|, \quad (3)$$

where $NT(T)$ is a set of nonterminal nodes in tree T, $PR(n)$ is a production rule rooted at node n.

Tree Overlapping similarity $S_{TO}(T_1, T_2)$ is defined as follows by using $C_{TO}(n_1, n_2)$.

$$S_{TO}(T_1, T_2) = \max_{n_1 \in NT(T_1)\ n_2 \in NT(T_2)} C_{TO}(n_1, n_2) \quad (4)$$

This formula corresponds to equation (1) of Tree Kernel.

As an example, we calculate $S_{TO}(T_1, T_2)$ in Figure 2 (1). Putting b_1^1 on b_1^2 gives Figure 2 (2) in which two production rules $b \rightarrow d\ e$ and $e \rightarrow g$ overlap respectively. Thus, $C_{TO}(b_1^1, b_1^2)$ becomes 2. While overlapping g_1^1 and g_1^2 gives Figure 2 (3) in which only one production rule $g \rightarrow i$ overlaps. Thus, $C_{TO}(g_1^1, g_1^2)$ becomes 1. Since there are no other node pairs which gives larger C_{TO} than 2, $S_{TO}(T_1, T_2)$ becomes 2.

Table 1: Example of the index table

p	$I[p]$
$a \rightarrow b\ c$	$\{a_1^1\}$
$b \rightarrow d\ e$	$\{b_1^1, b_1^2\}$
$e \rightarrow g$	$\{e_1^1, e_1^2\}$
$g \rightarrow i$	$\{g_1^1, g_1^2\}$
$a \rightarrow g\ b$	$\{a_1^2\}$
$g \rightarrow j$	$\{g_1^2\}$

3.2 Algorithm

Let us take an example in Figure 3 to explain the algorithm. Suppose that T_0 is a query tree and the corpus has only two trees, T_1 and T_2.

The method to find the most similar tree to a given query tree is basically the same as Tree Kernel's (section 2.2). However, unlike Tree Kernel, Tree Overlapping-based retrieval can be accelerated by indexing the corpus in advance. Thus, given a tree corpus, we build an index table $I[p]$ which maps a production rule p to its occurrences. Occurrences of production rules are represented by their left-hand side symbols, and are distinguished with respect to trees including the rule and

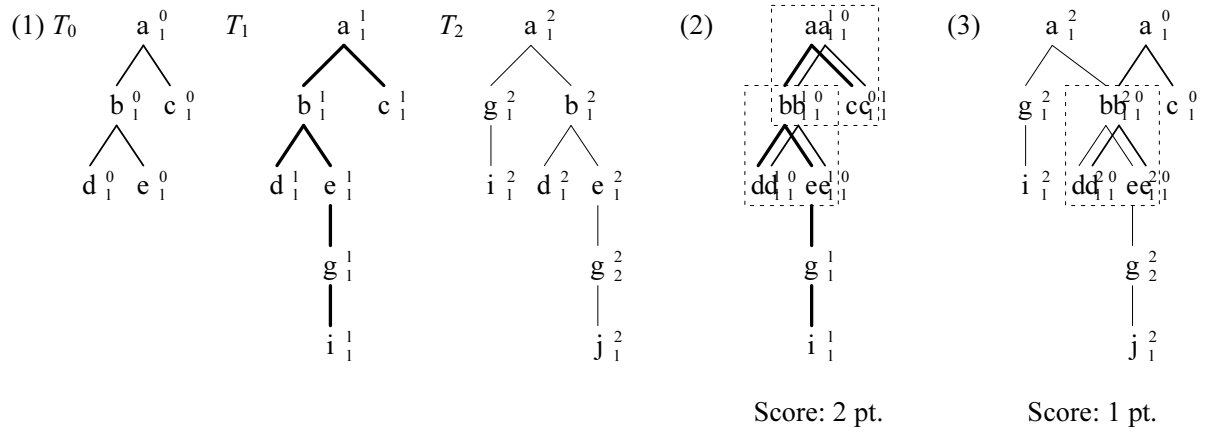

Figure 3: Example of Tree Overlapping-based retrieval

the position in the tree. $I[p]$ is defined as follows.

$$I[p] = \left\{ m \;\middle|\; \begin{array}{l} T \in F \\ \wedge\, m \in NT(T) \\ \wedge\, p = PR(m) \end{array} \right\} \qquad (5)$$

where F is the corpus (here $\{T_1, T_2\}$) and the meaning of other symbols is the same as the definition of C_{TO} (equation (3)).

Table 1 shows an example of the index table generated from T_1 and T_2 in Figure 3 (1). In Table 1, a superscript of a nonterminal symbol identifies a tree, and a subscript identifies a position in the tree.

By using the index table, we calculate $C[n, m]$ with the following algorithm.

for all (n, m) **do** $C[n, m] := 0$ **end**
foreach n **in** $NT(T_0)$ **do**
  foreach m **in** $I[PR(n)]$ **do**
    $(n', m') := top(n, m)$
    $C[n', m'] := C[n', m'] + 1$
  end
end

where $top(n, m)$ returns the upper-most pair of overlapped nodes when node n and m overlap. The value of top uniquely identifies a situation of overlapping two trees. Function $top(n, m)$ is calculated by the following algorithm.

function $top(n, m)$;
begin
  $(n', m') := (n, m)$
  while $order(n') = order(m')$ **do**
    $n' := parent(n')$
    $m' := parent(m')$
  end
  return (n', m')
end

where $parent(n)$ is the parent node of n, and $order(n)$ is the order of node n among its siblings. Table 2 shows example values of $top(n, m)$ generated by overlapping T_0 and T_1 in Figure 3. Note that top maps every pair of corresponding nodes in a certain overlapping situation to a pair of the upper-most nodes of that situation. This enables us to use the value of top as an identifier of a situation of overlap.

Table 2: Examples of $top(n, m)$

(n, m)	$top(n, m)$
(a_1^0, a_1^1)	(a_1^0, a_1^1)
(b_1^0, b_1^1)	(a_1^0, a_1^1)
(c_1^0, c_1^1)	(a_1^0, a_1^1)

Now $C[top(n, m)] = C_{TO}(n, m)$, therefore the tree similarity between a query tree T_0 and each tree T in the corpus $S_{TO}(T_0, T)$can be calculated by:

$$S_{TO}(T_0, T) = \max_{n \in NT(T_0),\, m \in NT(T)} C[top(n, m)] \qquad (6)$$

3.3 Comparison with Tree Kernel

The value of $S_{TO}(T_1, T_2)$ roughly corresponds to the number of production rules included in the largest sub-tree shared by T_1 and T_2. Therefore, this value represents the size of the subtree shared

by both trees, like Tree Kernel's K_C, though the definition of the subtree size is different.

One difference is that Tree Overlapping considers shared subtrees even though they are split by a nonshared node as shown in Figure 4. In Figure 4, T_1 and T_2 share two subtrees rooted at b and c, but their parent nodes are not identical. While Tree Kernel does not consider the superposition putting node a on h, Tree Overlapping considers putting a on h and assigns count 2 to this superposition.

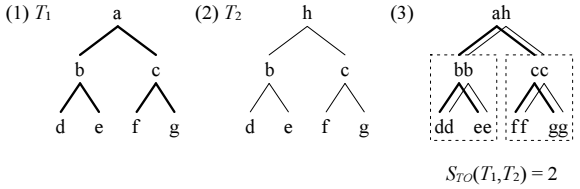

Figure 4: Example of counting two separated shared subtrees as one

Another, more important, difference is that Tree Overlapping retrieval can be accelerated by indexing the corpus in advance. The number of indexes is bounded above by the number of production rules, which is within a practical index size.

4 Subpath Set

4.1 Definition of similarity

Subpath Set similarity between two trees is defined as the number of subpaths shared by the trees. Given a tree, its subpaths is defined as a set of every path from the root node to leaves and their partial paths.

Figure 5 (2) shows all subpaths in T_1 and T_2 in Figure 5(1). Here we denotes a path as a sequence of node names such as (a, b, d). Therefore, Subpath Set similarity of T_1 and T_2 becomes 15.

4.2 Algorithm

Suppose T_0 is a query tree, TS is a set of trees in the corpus and $P(T)$ is a set of subpaths of T. We can build an index table $I[p]$ for each production rule p as follows.

$$I[p] = \{T | T \in TS \wedge p \in P(T)\} \qquad (7)$$

Using the index table, we can calculate the number of shared subpaths by T_0 and T, $S[T]$, by the following algorithm:

for all T $S[T] := 0$;
foreach p **in** $P(T_0)$ **do**
  foreach T **in** $I[p]$ **do**
    $S[T] := S[T] + 1$
  end
end

4.3 Comparison with Tree Kernel

As well as Tree Overlapping, Subpath Set retrieval can be accelerated by indexing the corpus. The number of indexes is bounded above by $L \times D^2$ where L is the maximum number of leaves of trees (the number of words in a sentence) and D is the maximum depth of syntactic trees. Moreover, considering a subpath as an index term, we can use existing retrieval tools.

Subpath Set uses less structural information than Tree Kernel and Tree Overlapping. It does not distinguish the order and number of child nodes. Therefore, the retrieval result tends to be noisy. However, Subpath Set is faster than Tree Overlapping, because the algorithm is simpler.

5 Experiments

This section describes the experiments which were conducted to compare the performance of structure retrieval based on Tree Kernel, Tree Overlapping and Subpath Set.

5.1 Data

We conducted two experiments using different annotated corpora. Titech corpus (Noro et al., 2005) consists of about 20,000 sentences of Japanese newspaper articles (Mainiti Shimbun). Each sentence has been syntactically annotated by hand. Due to the limitation of computational resources, we used randomly selected 2,483 sentences as a data collection.

Iwanami dictionary (Nishio et al., 1994) is a Japanese dictionary. We extracted 57,982 sentences from glosses in the dictionary. Each sentences was analyzed with a morphological analyzer, ChaSen (Asahara et al., 1996) and the MSLR parser (Shirai et al., 2000) to obtain syntactic structure candidates. The most probable structure with respect to PGLR model (Inui et al., 1996) was selected from the output of the parser. Since they were not investigated manually, some sentences might have been assigned incorrect structures.

5.2 Method

We conducted two experiments Experiment I and Experiment II with different corpora. The queries

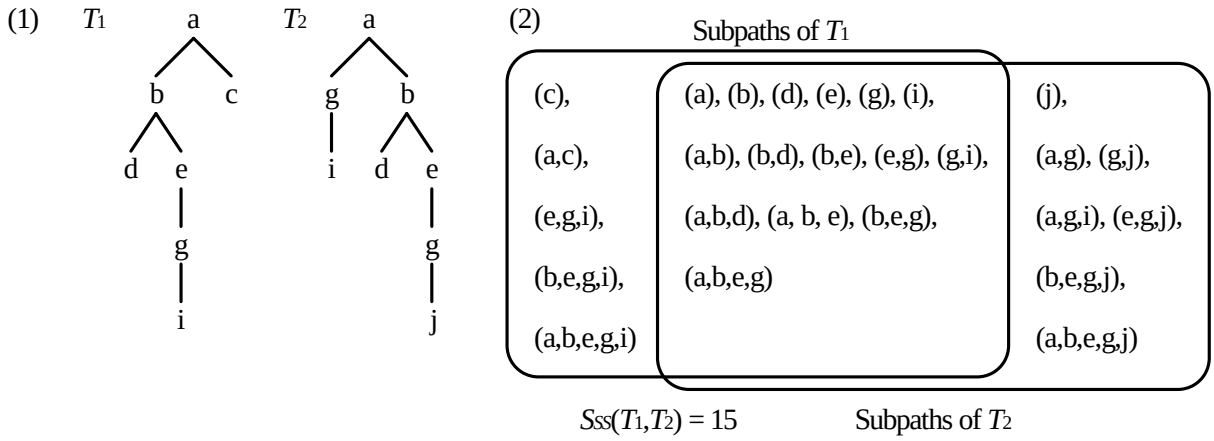

Figure 5: Example of subpaths

were extracted from these corpora. The algorithms described in the preceding sections were implemented with Ruby 1.8.2. Table 3 outlines the experiments.

Table 3: Summary of experiments

Experiment	I	II
Target corpus	Titech Corpus	Iwanami dict.
Corpus size	2,483 sent.	57,982 sent.
No. of queries	100	1,000
CPU	Intel Xeon (2.4GHz)	PowerPC G5 (2.3GHz)
Memory	2GB	2GB

5.3 Results and discussion

Since we select a query from the target corpus, the query is always ranked in the first place in the retrieval result. In what follows, we exclude the query tree as an answer from the result.

We evaluated the algorithms based on the following two factors: average retrieval time (CPU time) (Table 4) and the rank of the tree which was top-ranked in other algorithm (Table 5). For example, in Experiment I of Table 5, the column "$\geq$5th" of the row "TO/TK" means that there were 73 % of the cases in which the top-ranked tree by Tree Kernel (TK) was ranked 5th or above by Tree Overlapping (TO).

We consider Tree Kernel (TK) as the baseline method because it is a well-known existing similarity measure and exploits more information than others. Table 4 shows that in both corpora, the retrieval speed of Tree Overlapping (TO) is about 100 times faster than that of Tree Kernel, and the retrieval speed of Subpath Set (SS) is about 1,000 times faster than that of Tree Kernel. This results show we have successfully accelerated the retrieval speed.

Table 4: Average retrieval time per query [sec]

Algorithm	Experiment I	Experiment II
TK	529.42	3796.1
TO	6.29	38.3
SS	0.47	5.1

The retrieval time of Tree Overlapping, 6.29 and 38.3 sec./per query, seems be a bit long. However, we can shorten this time if we tune the implementation by using a compiler-type language. Note that the current implementation uses Ruby, an interpreter-type language.

Comparing Tree Overlapping and Subpath Set with respect to Tree Kernel (see rows "TK/TO" and "TK/SS"), the top-ranked trees by Tree Kernel are ranked in higher places by Tree Overlapping than by Subpath Set. This means Tree Overlapping is better than Subpath Set in approximating Tree Kernel.

Although the corpus of Experiment II is 20 times larger than that of Experiment I, the figures of Experiment II is better than that of Experiment I in Table 5. This could be explained as follows. In Experiment II, we used sentences from glosses in the dictionary, which tend to be formulaic and short. Therefore we could find similar sentences easier than in Experiment I.

To summarize the results, when being used in

Table 5: The rank of the top-ranked tree by other algorithm [%]

Experiment I

A/B	1st	$\geq$ 5th	$\geq$ 10th
TO/TK	34.0	73.0	82.0
SS/TK	16.0	35.0	45.0
TK/TO	29.0	41.0	51.0
SS/TO	27.0	49.0	58.0
TK/SS	17.0	29.0	37.0
TO/SS	29.0	58.0	69.0

Experiment II

A/B	1st	$\geq$ 5th	$\geq$ 10th
TO/TK	74.6	88.0	92.0
SS/TK	65.3	78.8	84.1
TK/TO	71.1	81.0	84.6
SS/TO	73.4	86.0	89.8
TK/SS	65.5	75.9	79.7
TO/SS	76.1	87.7	92.0

similarity calculation of tree structure retrieval, Tree Overlapping approximates Tree Kernel better than Subpath Set, while Subpath Set is faster than Tree Overlapping.

6 Conclusion

We proposed two fast algorithms to retrieve sentences which have a similar syntactic structure: Tree Overlapping (TO) and Subpath Set (SS). And we compared them with Tree Kernel (TK) to obtain the following results.

- Tree Overlapping-based retrieval outputs similar results to Tree Kernel-based retrieval and is 100 times faster than Tree Kernel-based retrieval.
- Subpath Set-based retrieval is not so good at approximating Tree Kernel-based retrieval, but is 1,000 times faster than Tree Kernel-based retrieval.

Structural retrieval is useful for annotationg corpora with syntactic information (Yoshida et al., 2004). We are developing a corpus annotation tool named "eBonsai" which supports human to annotate corpora with syntactic information and to retrieve syntactic structures. Integrating annotation and retrieval enables annotators to annotate a new instance with looking back at the already annotated instances which share the similar syntactic structure with the current one. For such purpose, Tree Overlapping and Subpath Set algorithms contribute to speed up the retrieval process, thus make the annotation process more efficient.

However, "similarity" of sentences is affected by semantic aspects as well as structural aspects. The output of the algorithms do not always conform with human's intuition. For example, the two sentences in Figure 6 have very similar structures including particles, but they are hardly considered similar from human's viewpoint. With this respect, it is hardly to say which algorithm is superior to others.

As a future work, we need to develop a method to integrate both content-based and structure-based similarity measures. To this end, we have to evaluate the algorithms in real application environments (e.g. information retrieval and machine translation) because desired properties of similarity are different depending on applications.

References

Asahara, M. and Matsumoto, Y., Extended Models and Tools for High-performance Part-of-Speech Tagger. Proceedings of COLING 2000, 2000.

Collins, M. and Duffy, N. Parsing with a Single Neuron: Convolution Kernels for Natural Language Problems. Technical report UCSC-CRL-01-01, University of California at Santa Cruz, 2001.

Collins, M. and Duffy, N. Convolution Kernels for Natural Language. In Proceedings of NIPS 2001, 2001.

Inui, K., Shirai, K., Tokunaga T. and Tanaka H., The Integration of Statistics-based Techniques in the Analysis of Japanese Sentences. Special Interest Group of Natural Language Processing, Information Processing Society of Japan, Vol. 96, No. 114, 1996.

Nagao, M. A framework of a mechanical translation between Japanese and English by analogy principle. In Alick Elithorn and Ranan Banerji, editors, Artifical and Human Intelligence, pages 173-180. Amsterdam, 1984.

Noro, T., Koike, C., Hashimoto, T., Tokunaga, T. and Tanaka, H. Evaluation of a Japanese CFG Derived from a Syntactically Annotated Corpus with respect to Dependency Measures, The 5th Workshop on Asian Language Resources, pp.9-16, 2005.

Nishio, M., Iwabuchi, E. and Mizutani, S. (ed.) Iwanami Kokugo Jiten, Iwanamishoten, 5th Edition, 1994.

Shirai, K., Ueki, M. Hashimoto, T., Tokunaga, T. and Tanaka, H., MSLR Parser Tool Kit - Tools for Natural Language Analysis. Journal of Natural Language

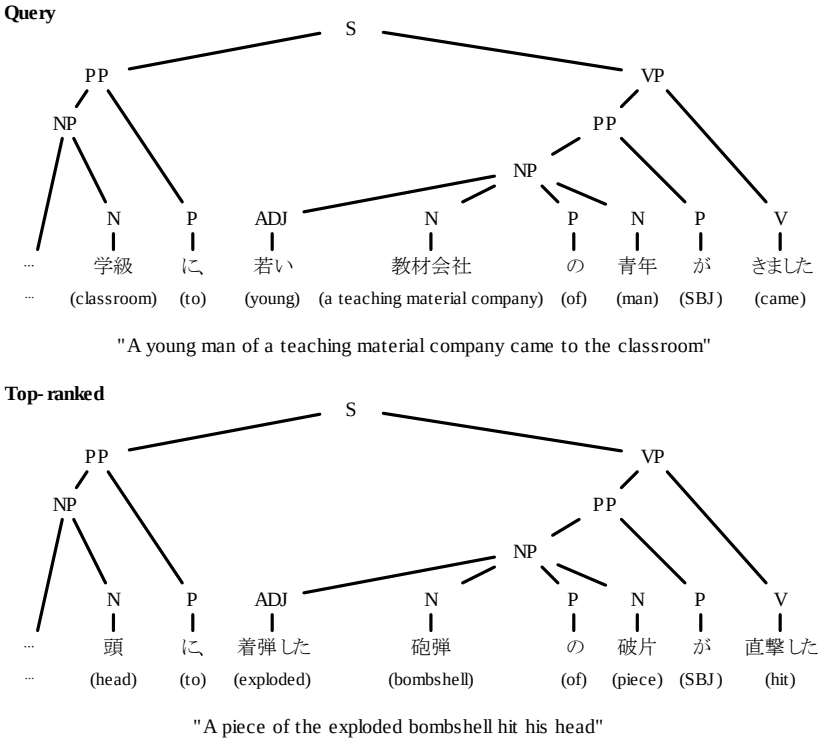

Figure 6: Example of a retrieved similar sentence

Processing, Vol. 7, No. 5, pp. 93-112, 2000. (in Japanese)

Somers, H., McLean, I., Jones, D. Experiments in multilingual example-based generation. CSNLP 1994: 3rd conference on the Cognitive Science of Natural Language Processing, Dublin, 1994.

Takahashi, T., Inui K., and Matsumoto, Y.. Methods of Estimating Syntactic Similarity. Special Interest Group of Natural Language Processing, Information Processing Society of Japan, NL-150-7, 2002. (in Japanese)

Yoshida, K., Hashimoto, T., Tokunaga, T. and Tanaka, H.. Retrieving annotated corpora for corpus annotation. Proceedings of 4th International Conference on Language Resources and Evaluation: LREC 2004. pp.1775 – 1778. 2004.