Hierarchical Directed Acyclic Graph Kernel: Methods for Structured Natural Language Data

Jun Suzuki, Tsutomu Hirao, Yutaka Sasaki, and Eisaku Maeda
NTT Communication Science Laboratories, NTT Corp.
2-4 Hikaridai, Seika-cho, Soraku-gun, Kyoto, 619-0237 Japan
{jun, hirao, sasaki, maeda}@cslab.kecl.ntt.co.jp

Abstract

This paper proposes the "*Hierarchical Directed Acyclic Graph (HDAG) Kernel*" for structured natural language data. The HDAG Kernel directly accepts several levels of both chunks and their relations, and then efficiently computes the weighed sum of the number of common attribute sequences of the HDAGs. We applied the proposed method to question classification and sentence alignment tasks to evaluate its performance as a similarity measure and a kernel function. The results of the experiments demonstrate that the HDAG Kernel is superior to other kernel functions and baseline methods.

1 Introduction

As it has become easy to get structured corpora such as annotated texts, many researchers have applied statistical and machine learning techniques to NLP tasks, thus the accuracies of basic NLP tools, such as POS taggers, NP chunkers, named entities taggers and dependency analyzers, have been improved to the point that they can realize practical applications in NLP.

The motivation of this paper is to identify and use richer information within texts that will improve the performance of NLP applications; this is in contrast to using feature vectors constructed by a bag-of-words (Salton et al., 1975).

We now are focusing on the methods that use numerical *feature vectors* to represent the features of natural language data. In this case, since the original natural language data is symbolic, researchers convert the symbolic data into numeric data. This process, *feature extraction*, is ad-hoc in nature and differs with each NLP task; there has been no neat formulation for generating feature vectors from the semantic and grammatical structures inside texts.

Kernel methods (Vapnik, 1995; Cristianini and Shawe-Taylor, 2000) suitable for NLP have recently been devised. *Convolution Kernels* (Haussler, 1999) demonstrate how to build kernels over discrete structures such as strings, trees, and graphs. One of the most remarkable properties of this kernel methodology is that it retains the original representation of objects and algorithms manipulate the objects simply by computing kernel functions from the inner products between pairs of objects. This means that we do not have to map texts to the feature vectors by explicitly representing them, as long as an efficient calculation for the inner products between a pair of texts is defined. The kernel method is widely adopted in Machine Learning methods, such as the *Support Vector Machine (SVM)* (Vapnik, 1995). In addition, kernel function $K(x, y)$ has been described as a similarity function that satisfies certain properties (Cristianini and Shawe-Taylor, 2000). The similarity measure between texts is one of the most important factors for some tasks in the application areas of NLP such as Machine Translation, Text Categorization, Information Retrieval, and Question Answering.

This paper proposes the *Hierarchical Directed Acyclic Graph (HDAG) Kernel*. It can handle several of the structures found within texts and can calculate the similarity with regard to these structures at practical cost and time. The HDAG Kernel can be widely applied to learning, clustering and similarity measures in NLP tasks.

The following sections define the HDAG Kernel and introduce an algorithm that implements it. The results of applying the HDAG Kernel to the tasks of question classification and sentence alignment are then discussed.

2 Convolution Kernels

Convolution Kernels were proposed as a concept of kernels for a discrete structure. This framework defines a kernel function between input objects by applying convolution "sub-kernels" that are the kernels for the decompositions (parts) of the objects.

Let D be a positive integer and $X, X_1, \ldots, X_D$ be nonempty, separable metric spaces. This paper focuses on the special case that $X, X_1, \ldots, X_D$ are countable sets. We start with $x \in X$ as a *composite structure* and $\mathbf{x} = x_1, \ldots, x_D$ as its "*parts*", where $x_d \in X_d$. R is defined as a relation on the set $X_1 \times \cdots \times X_D \times X$ such that $R(\mathbf{x}, x)$ is true if $\mathbf{x}$ are the "parts" of x. $R^{-1}(x)$ is defined as $R^{-1}(x) = \{\mathbf{x} : R(\mathbf{x}, x)\}$.

Suppose $x, y \in X$, $\mathbf{x}$ be the parts of x with $\mathbf{x} = x_1, \ldots, x_D$, and $\mathbf{y}$ be the parts of y with $\mathbf{y} = y_1, \ldots, y_D$. Then, the similarity $K(x, y)$ between x and y is defined as the following generalized convolution:

$$K(x,y) = \sum_{\mathbf{x} \in R^{-1}(x)} \sum_{\mathbf{y} \in R^{-1}(y)} \prod_{d=1}^{D} K_d(x_d, y_d). \qquad (1)$$

We note that Convolution Kernels are abstract concepts, and that instances of them are determined by the definition of *sub-kernel* $K_d(x_d, y_d)$. The *Tree Kernel* (Collins and Duffy, 2001) and *String Subsequence Kernel (SSK)* (Lodhi et al., 2002), developed in the NLP field, are examples of Convolution Kernels instances.

An explicit definition of both the Tree Kernel and SSK $K(x, y)$ is written as:

$$K(x,y) = \langle \phi(x) \cdot \phi(y) \rangle = \sum_{i=1}^{M} \phi_i(x) \cdot \phi_i(y). \qquad (2)$$

Conceptually, we enumerate all sub-structures occurring in x and y, where M represents the total number of possible sub-structures in the objects. ϕ, the feature mapping from the sample space to the feature space, is given by $\phi(x) = (\phi_1(x), \ldots, \phi_M(x))$.

In the case of the Tree Kernel, x and y be trees. The Tree Kernel computes the number of common subtrees in two trees x and y. $\phi_i(x)$ is defined as the number of occurrences of the i'th enumerated subtree in tree x.

In the case of SSK, input objects x and y are string sequences, and the kernel function computes the sum of the occurrences of i'th common subsequence $\phi_i(x)$ weighted according to the length of the subsequence. These two kernels make polynomial-time calculations, based on efficient recursive calculation, possible, see equation (1). Our proposed method uses the framework of Convolution Kernels.

3 HDAG Kernel

3.1 Definition of HDAG

This paper defines HDAG as a Directed Acyclic Graph (DAG) with hierarchical structures. That is, certain nodes contain DAGs within themselves.

In basic NLP tasks, chunking and parsing are used to analyze the text semantically or grammatically. There are several levels of chunks, such as phrases, named entities and sentences, and these are bound by relation structures, such as dependency structure, anaphora, and coreference. HDAG is designed to enable the representation of all of these structures inside texts, hierarchical structures for chunks and DAG structures for the relations of chunks. We believe this richer representation is extremely useful to improve the performance of similarity measure between texts, moreover, learning and clustering tasks in the application areas of NLP.

Figure 1 shows an example of the text structures that can be handled by HDAG. Figure 2 contains simple examples of HDAG that elucidate the calculation of similarity.

As shown in Figures 1 and 2, the nodes are allowed to have more than zero attributes, because nodes in texts usually have several kinds of attributes. For example, attributes include words, part-of-speech tags, semantic information such as Word-

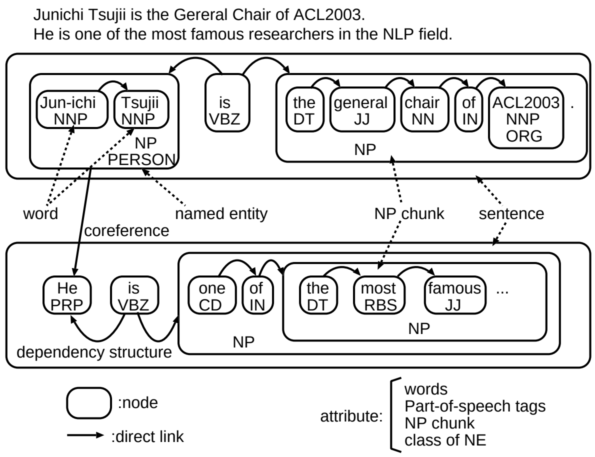

Figure 1: Example of the text structures handled by HDAG

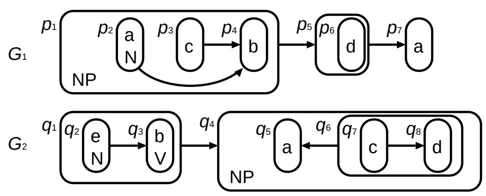

Figure 2: Examples of HDAG structure

Net, and class of the named entity.

3.2 Definition of HDAG Kernel

First of all, we define the set of nodes in HDAGs G_1 and G_2 as P and Q, respectively, p and q represent nodes in the graph that are defined as $\{p|p_i \in P, i = 1, \ldots, |P|\}$ and $\{q|q_j \in Q, j = 1 \ldots, |Q|\}$, respectively. We use the expression $p_1 \rightarrow p_4 \rightarrow p_7$ to represent the path from p_1 to p_7 through p_4.

We define "attribute sequence" as a sequence of attributes extracted from nodes included in a sub-path. The attribute sequence is expressed as 'A-B' or 'A-(C-B)' where ( ) represents a chunk. As a basic example of the extraction of attribute sequences from a sub-path, $q_2 \rightarrow q_3$ in Figure 2 contains the four attribute sequences 'e-b', 'e-V', 'N-b' and 'N-V', which are the combinations of all attributes in q_2 and q_3. Section 3.3 explains in detail the method of extracting attribute sequences from sub-paths.

Next, we define "terminated nodes" as those that do not contain any graph, such as p_2, p_6; "non-terminated nodes" are those that do, such as q_1, q_4.

Since HDAGs treat not only exact matching of sub-structures but also approximate matching, we allow node skips according to decay factor $\lambda (0 < \lambda \leq 1)$ when extracting attribute sequences from the sub-paths. This framework makes similarity evaluation robust; the similar sub-structures can be evaluated in the value of similarity, in contrast to exact matching that never evaluate the similar sub-structure. Next, we define parameter n ($n = 1, 2, \ldots$) as the number of attributes combined in the attribute sequence. When calculating similarity, we consider only combination lengths of up to n.

Given the above discussion, the feature vector of HDAG is written as $\phi(G) = (\phi_1(G), \ldots, \phi_M(G))$, where ϕ represents the explicit feature mapping of HDAG and M represents the number of all possible n attribute combinations. The value of $\phi_i(G)$ is the number of occurrences of the i'th attribute sequence in HDAG G; each attribute sequence is weighted according to the node skip. The similarity between HDAGs, which is the definition of the HDAG Kernel, follows equation (2) where input objects x and y are G_1 and G_2, respectively. According to this approach, the HDAG Kernel calculates the inner product of the common attribute sequences weighted according to their node skips and the occurrence between the two HDAGs, G_1 and G_2.

We note that, in general, if the dimension of the feature space becomes very high or approaches infinity, it becomes computationally infeasible to generate feature vector $\phi(G)$ explicitly. To improve the reader's understanding of what the HDAG Kernel calculates, before we introduce our efficient calculation method, the next section details the attribute sequences that become elements of the feature vector if the calculation is explicit.

3.3 Attribute Sequences: The Elements of the Feature Vector

We describe the details of the attribute sequences that are elements of the feature vector of the HDAG Kernel using G_1 and G_2 in Figure 2.

The framework of node skip

We denote the explicit representation of a node skip by "$*$". The attribute sequences in the sub-path under the "node skip" are written as 'a-$*$-c'. It costs λ to skip a terminated node. The cost of skipping a

Table 1: Attribute sequences and the values of nodes p_1 and q_4

p_1		
sub-path	a. seq.	val.
$n = 1$		
p_1	NP	1
p_2	a-∗	λ
p_2	N-∗	λ
p_3	c-∗	λ
p_4	∗-b	2λ
$n = 2$		
$p_2 \rightarrow p_4$	a-b	1
$p_2 \rightarrow p_4$	N-b	1
$p_3 \rightarrow p_4$	c-b	1

q_4		
sub-path	a. seq.	val.
$n = 1$		
q_4	NP	1
q_5	(∗-∗)-a	λ^2
q_6	(c-∗)-∗	λ^2
q_6	(∗-d)-∗	λ^2
$n = 2$		
q_6	(c-d)-∗	λ
$q_6 \rightarrow q_5$	(c-∗)-a	λ
$q_6 \rightarrow q_5$	(∗-d)-a	λ
$n = 3$		
$q_6 \rightarrow q_5$	(c-d)-c	1

non-terminated node is the same as skipping all the graphs inside the non-terminated node. We introduce decay functions $N_\lambda(p)$, $T_\lambda(p)$ and $B_\lambda(p)$; all are based on decay factor λ. $N_\lambda(p)$ represents the cost of node skip p. For example, $N_\lambda(p_1) = 2\lambda^2$ represents the cost of node skip $p_2 \rightarrow s_4$ and that of $p_3 \rightarrow p_4$; $N_\lambda(p_2) = \lambda$ is the cost of just node skip p_2. $T_\lambda(p)$ represents the sum of the multiplied cost of the node skips of all of the nodes that have a path to p, $T_\lambda(p_4) = 2\lambda$ that is the sum cost of both p_2 and p_3 that have a path to p_4, $T_\lambda(p_1) = 1(\lambda^0)$. $B_\lambda(p)$ represents the sum of the multiplied cost of the node skips of all the nodes that p has a path to. $B_\lambda(p_2) = \lambda$ represents the cost of node skip p_4 where p_2 has a path to p_4.

Attribute sequences for non-terminated nodes

We define the attributes of the non-terminated node as the combinations of all attribute sequences including the node skip. Table 1 shows the attribute sequences and values of p_1 and q_4.

Details of the elements in the feature vector

The elements of the feature vector are not considered in any of the node skips. This means that 'A-∗-B-C' is the same element as 'A-B-C', and 'A-∗-∗-B-C' and 'A-∗-B-∗-C' are also the same element as 'A-B-C'. Considering the hierarchical structure, it is natural to assume that '(N-∗)-(d)-a' and '(N-∗)-((∗-d)-a)' are different elements. However, in the framework of the node skip and the attributes of the non-terminated node, '(N-∗)-(∗)-a' and '(N-∗)-((∗-∗)-a)' are treated as the same element. This framework

Table 2: Similarity values of G_1 and G_2 in Figure 2

G_1		G_2		
att. seq.	value	att. seq.	value	
$n = 1$				
NP	1	NP	1	1
N	1	N	1	1
a	2	a	1	2
b	1	b	1	1
c	1	c	1	1
d	1	d	1	1
$n = 2$				
(N-∗)-(∗)-a	λ^2	(N-∗)-((∗-∗)-a)	λ^3	λ^5
N-b	1	N-b	1	1
(N-∗)-(d)	λ	(N-∗)-((∗-d)-∗)	λ^3	λ^4
(∗-b)-(∗)-a	$2\lambda^2$	(∗-b)-((∗-∗)-a)	λ^3	$2\lambda^5$
(∗-b)-(d)	2λ	(∗-b)-((∗-d)-∗)	λ^3	$2\lambda^4$
(c-∗)-(∗)-a	λ^2	((c-∗)-a)	λ	λ^3
(c-∗)-(d)	λ	c-d	1	λ
(d)-a	1	(c-∗)-a	λ	λ
$n = 3$				
(N-b)-(∗)-a	λ	(N-b)-((∗-∗)-a)	λ^2	λ^3
(N-b)-(d)	1	(N-b)-((∗-d)-∗)	λ^2	λ^2

achieves approximate matching of the structure automatically, The HDAG Kernel judges all pairs of attributes in each attribute sequence that are inside or outside the same chunk. If all pairs of attributes in the attribute sequences are in the same condition, inside or outside the chunk, then the attribute sequences judge as the same element.

Table 2 shows the similarity, the values of $K_{HDAG}(G_1, G_2)$, when the feature vectors are explicitly represented. We only show the common elements of each feature vector that appear in both G_1 and G_2, since the number of elements that appear in only G_1 or G_2 becomes very large.

Note that, as shown in Table 2, the attribute sequences of the non-terminated node itself are not addressed by the features of the graph. This is due to the use of the hierarchical structure; the attribute sequences of the non-terminated node come from the combination of the attributes in the terminated nodes. In the case of s_1, attribute sequence 'N-∗' comes from 'N' in s_2. If we treat both 'N-∗' in p_1 and 'N' in p_2, we evaluate the attribute sequence 'N' in p_2 twice. That is why the similarity value in Table 2 does not contain 'c-∗' in p_1 and '(c-∗)-∗' in q_4, see Table 1.

3.4 Calculation

First, we determine $V_m(s,t)$, which returns the sum of the common attribute sequences of the m-combination of attributes between nodes p and q.

$$V_m(s,t) = \begin{cases} V'_m(p,q) + val(p,q), & \text{if} \quad m=1 \\ V'_m(p,q), & \text{otherwise} \end{cases} \tag{3}$$

$$V'_m(p,q) = \begin{cases} 0, \quad \text{if} \quad in(p)=\emptyset \quad \text{and} \quad in(q)=\emptyset \\ \displaystyle\sum_{s \in in(q)} T_\lambda(s) \cdot B_\lambda(s) \cdot val(s,q), \\ \qquad \text{if} \quad in(p) \neq \emptyset \quad \text{and} \quad in(q)=\emptyset \\ \displaystyle\sum_{t \in in(q)} T_\lambda(t) \cdot B_\lambda(t) \cdot val(p,t), \\ \qquad \text{if} \quad in(p)=\emptyset \quad \text{and} \quad in(q) \neq \emptyset \\ \displaystyle\sum_{s \in in(p)} \sum_{t \in in(q)} B_\lambda(s) \cdot B_\lambda(t) \cdot A_m(s,t), \\ \qquad \text{otherwise} \end{cases} \tag{4}$$

$val(p,q)$ returns the number of common attributes of nodes p and q, not including the attributes of nodes inside p and q. We define function $in(p)$ as returning a set of nodes inside a non-terminated node p. $in(p)=\emptyset$ means node p is a terminated node. For example, $in(p_1)=\{p_2,p_3,p_4\}$ and $in(p_2)=\emptyset$.

We define functions $A_m(p,q)$, $A'_m(p,q)$ and $A''_m(p,q)$ to calculate $V_m(p,q)$.

$$A_m(p,q) = V_m(p,q) + \sum_{a=1}^{m-1} A'_a(p,q) \cdot V_{m-a}(p,q) \tag{5}$$

$$A'_m(p,q) = \sum_{t \in rel(q)} \big(N_\lambda(t) \cdot A'_m(p,t) + A''_m(p,t)\big) \tag{6}$$

$$A''_m(p,q) = \sum_{s \in rel(p)} \big(N_\lambda(s) \cdot A''_m(s,q) + A_m(s,q)\big) \tag{7}$$

The boundary conditions are

$$A_m(p,q) = T_\lambda(p) \cdot T_\lambda(q) \cdot V_m(p,q), \quad \text{if} \quad m=1 \tag{8}$$
$$A'_m(p,q) = 0, \quad \text{if} \quad rel(q)=\emptyset \tag{9}$$
$$A''_m(p,q) = 0, \quad \text{if} \quad rel(p)=\emptyset. \tag{10}$$

Function $rel(p)$ returns the set of nodes that have direct links to node p. $rel(p)=\emptyset$ means no nodes have direct links to s. $rel(p_4)=\{p_2,p_3\}$ and $rel(p_1)=\emptyset$.

Next, we define $K(p,q)$ as representing the sum of the common attribute sequences that are the m-combinations of attributes extracted from the subpaths whose sinks are p and q, respectively.

$$K_m(p,q) = \begin{cases} val(p,q), & \text{if} \quad m=1 \\ \displaystyle\sum_{a=1}^{m-1} C'_a(p,q) \cdot V_{m-a}(p,q), & \text{otherwise} \end{cases} \tag{11}$$

Functions $C_m(p,q)$, $C'_m(p,q)$ and $C''_m(p,q)$, needed for the recursive calculation of $K_m(p,q)$, are written in the same form as $A_m(p,q)$, $A'_m(p,q)$ and $A''_m(p,q)$ respectively, except for the boundary condition of $C_m(p,q)$, which is written as:

$$C_m(p,q) = V_m(p,q), \quad \text{if} \quad m=1. \tag{12}$$

Finally, an efficient similarity calculation formula is written as

$$K_{HDAG}(G_1,G_2) = \sum_{m=1}^{n} \sum_{p \in P} \sum_{q \in Q} K_m(p,q). \tag{13}$$

According to equation (13), given the recursive definition of $K_m(p,q)$, the similarity between two HDAGs can be calculated in $O(n|P||Q|)$ time[1].

3.5 Efficient Calculation Method

We will now elucidate an efficient processing algorithm. First, as a pre-process, the nodes are sorted under the following condition: all nodes that have a path to the focused node and are in the graph inside the focused node should be set before the focused node. We can get at least one set of ordered nodes since we are treating an HDAG. In the case of G_1, we can get $\langle p_2, p_3, p_4, p_1, p_6, p_5, p_7 \rangle$. We can rewrite the recursive calculation formula in "for loops", if we follow the sorted order. Figure 3 shows the algorithm of the HDAG kernel. Dynamic programming technique is used to compute the HDAG Kernel very efficiently because when following the sorted order, the values that are needed to calculate the focused pair of nodes are already calculated in the previous calculation. We can calculate the table by following the order of the nodes from left to right and top to bottom.

We normalize the computed kernels before their use within the algorithms. The normalization corresponds to the standard unit norm normalization of

[1]We can easily rewrite the equation to calculate all combinations of attributes, but the order of calculation time becomes $O(|P||Q|)$.

```
Algorithm HDAG Kernel n_combination
for (i = 1; i ≤ |P|; i++)
  for (j = 1; j ≤ |Q|; j++)
    K_1[p_i, q_j] = V_1(p_i, q_j) = val(p_i, q_j)
    if   in(p_i) ≠ ∅   and   in(q_j) ≠ ∅
      foreach s ∈ in(p_i)
        foreach t ∈ in(q_j)
          for(m = 1; m ≤ n; m++)
            V_m(p_i, q_j)+=B_λ(s)B_λ(t)A(s, t)
          end
        end
      end
    else if   in(p_i) ≠ ∅
      foreach s ∈ in(p_i)
        V_1(p_i, q_j)+=B_λ(s)T_λ(s)val(s, q_j)
      end
    else if   in(q_j) ≠ ∅
      foreach t ∈ in(q_q)
        V_1(p_i, q_j)+=B_λ(t)T_λ(t)val(p_i, t)
      end
    end
    foreach s ∈ rel(p_i)
      for (m = 1; m ≤ n; m++)
        C''_m[p_i, q_j]+=N_λ(s)C''_m[s, q_j] + C_m[s, q_j]
        A''_m[p_i, q_j]+=N_λ(s)A''_m[s, q_j] + A_m[s, q_j]
      end
    end
    foreach t ∈ rel(q_j)
      for (m = 1; m ≤ n; m++)
        C'_m[p_i, q_j]+=N_λ(t)C'_m[p_i, t] + C''_m[p_i, t]
        A'_m[p_i, q_j]+=N_λ(t)A'_m[p_i, t] + A''_m[p_i, t]
      end
    end
    C_1[p_i, q_j] = V_1[p_i, q_j]
    A_1[p_i, q_j] = T_λ(p_i) · T_λ(q_j) · V_1[p_i, q_j]
    for (m = 2; m ≤ n; m++)
      C_m[p_i, q_j] = A_m[p_i, q_j] = V_m[p_i, q_j]
      for (a = 1; a < m; a++)
        K_m[p_i, q_j]+=C'_a[p_i, q_j] · V_{m−a}[p_i, q_j]
        C_m[p_i, q_j]+=C''_a[p_i, q_j] · V_{m−a}[p_i, q_j]
        A_m[p_i, q_j]+=A'_a[p_i, q_j] · V_{m−a}[p_i, q_j]
      end
    end
  end
end
return   Σ_{m=1}^{n} Σ_{p_i∈P} Σ_{q_j∈Q} K_m[p_i, q_j]
```

Figure 3: Algorithm of the HDAG Kernel

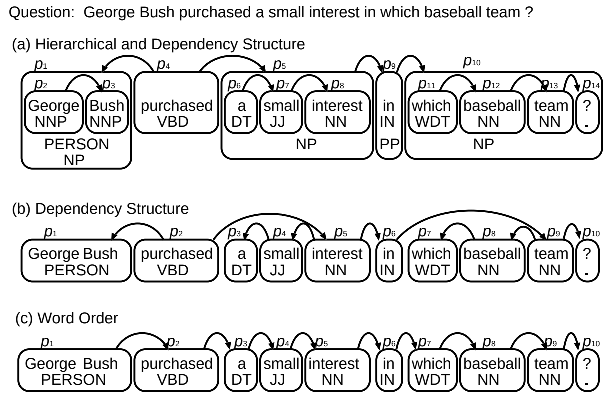

Figure 4: Examples of Input Object Structure: (a) HDAG, (b) DAG and DSK', (c) SSK'

examples in the feature space corresponding to the kernel space (Lodhi et al., 2002).

$$\hat{K}(x, y) = \frac{K(x, y)}{\sqrt{K(x, x) \cdot K(y, y)}} \tag{14}$$

4 Experiments

We evaluated the performance of the proposed method in an actual application of NLP; the data set is written in Japanese.

We compared HDAG and DAG (the latter had no hierarchy structure) to the String Subsequence Kernel (SSK) for word sequence, Dependency Structure Kernel (DSK) (Collins and Duffy, 2001) (a special case of the Tree Kernel), and Cosine measure for feature vectors consisting of the occurrence of attributes (BOA), and the same as BOA, but only the attributes of noun and unknown word (BOA')were used.

We expanded SSK and DSK to improve the total performance of the experiments. We denote them as SSK' and DSK' respectively. The original SSK treats only exact n string combinations based on parameter n. We consider string combinations of up to n for SSK'. The original DSK was specifically constructed for parse tree use. We expanded it to be able to treat the n combinations of nodes and the free order of child node matching.

Figure 4 shows some input objects for each evaluated kernel, (a) for HDAG, (b) for DAG and DSK', and (c) for SSK'. Note, though DAG and DSK' treat the same input objects, their kernel calculation methods differ as do the return values.

We used the words and semantic information of "Goi-taikei" (Ikehara et al., 1997), which is similar to WordNet in English, as the attributes of the node. The chunks and their relations in the texts were analyzed by cabocha (Kudo and Matsumoto, 2002), and named entities were analyzed by the method of (Isozaki and Kazawa, 2002).

We tested each n-combination case with changing parameter λ from 0.1 through 0.9 in the step of 0.1. Only the best performance achieved under parameter λ is shown in each case.

Table 3: Results of the performance as a similarity measure for question classification

n	1	2	3	4	5	6
HDAG	-	.580	**.583**	.580	.579	.573
DAG	-	.577	**.578**	.573	.573	.563
DSK'	-	**.547**	.469	.441	.436	.436
SSK'	-	.568	**.572**	.570	.562	.548
BOA	.556					
BOA'	.555					

Table 4: Results of the performance as a similarity measure for sentence alignment

n	1	2	3	4	5	6
HDAG	-	**.523**	.484	.467	.442	.423
DAG	-	**.503**	.478	.461	.439	.420
DSK'	-	**.174**	.083	.035	.020	.021
SSK'	-	**.479**	.444	.422	.412	.398
BOA	.394					
BOA'	.451					

4.1 Performance as a Similarity Measure

Question Classification

We used the 1011 questions of NTCIR-QAC1 [2] and the 2000 questions of CRL-QA data [3] We assigned them into 148 question types based on the CRL-QA data.

We evaluated classification performance in the following step. First, we extracted one question from the data. Second, we calculated the similarity between the extracted question and all the other questions. Third, we ranked the questions in order of descending similarity. Finally, we evaluated performance as a similarity measure by Mean Reciprocal Rank (MRR) (Voorhees and Tice, 1999) based on the question type of the ranked questions.

Table 3 shows the results of this experiment.

Sentence Alignment

The data set (Hirao et al., 2003) taken from the "Mainichi Shinbun", was formed into abstract sentences and manually aligned to sentences in the "Yomiuri Shinbun" according to the meaning of sentence (did they say the same thing).

This experiment was prosecuted as follows. First, we extracted one abstract sentence from the "Mainichi Shinbun" data-set. Second, we calculated the similarity between the extracted sentence and the sentences in the "Yomiuri Shinbun" data-set. Third, we ranked the sentences in the "Yomiuri Shinbun" in descending order based on the calculated similarity values. Finally, we evaluated performance as a similarity measure using the MRR measure.

Table 4 shows the results of this experiment.

[2]http://www.nlp.cs.ritsumei.ac.jp/qac/

[3]http://www.cs.nyu.edu/~sekine/PROJECT/CRLQA/

Table 5: Results of question classification by SVM with comparison kernel functions

n	1	2	3	4	5	6
HDAG	-	.862	.865	**.866**	.864	.865
DAG	-	**.862**	**.862**	.847	.818	.751
DSK'	-	**.731**	.595	.473	.412	.390
SSK'	-	**.850**	.847	.825	.777	.725
BOA+poly	.810	**.823**	.800	.753	.692	.625
BOA'+poly	**.807**	**.807**	.742	.666	.558	.468

4.2 Performance as a Kernel Function

Question Classification

The comparison methods were evaluated the performance as a kernel function in the machine learning approach of the Question Classification. We chose SVM as a kernel-based learning algorithm that produces state-of-the-art performance in several NLP tasks.

We used the same data set as used in the previous experiments with the following difference: if a question type had fewer than ten questions, we moved the entries into the upper question type as defined in CRL-QA data to provide enough training samples for each question type. We used *one-vs-rest* as the multi-class classification method and found a highest scoring question type. In the case of BOA and BOA', we used the polynomial kernel (Vapnik, 1995) to consider the attribute combinations.

Table 5 shows the average accuracy of each question as evaluated by 5-fold cross validation.

5 Discussion

The experiments in this paper were designed to evaluated how the similarity measure reflects the semantic information of texts. In the task of Question Classification, a given question is classified into Question Type, which reflects the intention of the question. The Sentence Alignment task evaluates which sentence is the most semantically similar to a given sentence.

The HDAG Kernel showed the best performance in the experiments as a similarity measure and as a kernel of the learning algorithm. This proves the usefulness of the HDAG Kernel in determining the similarity measure of texts and in providing an SVM kernel for resolving classification problems in NLP tasks. These results indicate that our approach, incorporating richer structures within texts, is well suited to the tasks that require evaluation of the semantical similarity between texts. The potential use of the HDAG Kernel is very wider in NLP tasks, and we believe it will be adopted in other practical NLP applications such as Text Categorization and Question Answering.

Our experiments indicate that the optimal parameters of combination number n and decay factor λ depend the task at hand. They can be determined by experiments.

The original DSK requires exact matching of the tree structure, even when expanded (DSK') for flexible matching. This is why DSK' showed the worst performance. Moreover, in Sentence Alignment task, paraphrasing or different expressions with the same meaning is common, and the structures of the parse tree widely differ in general. Unlike DSK', SSK' and HDAG Kernel offer approximate matching which produces better performance.

The structure of HDAG approaches that of DAG, if we do not consider the hierarchical structure. In addition, the structure of sequences (strings) is entirely included in that of DAG. Thus, the framework of the HDAG Kernel covers DAG Kernel and SSK.

6 Conclusion

This paper proposed the HDAG Kernel, which can reflect the richer information present within texts. Our proposed method is a very generalized framework for handling the structure inside a text.

We evaluated the performance of the HDAG Kernel both as a similarity measure and as a kernel function. Our experiments showed that HDAG Kernel offers better performance than SSK, DSK, and the baseline method of the Cosine measure for feature vectors, because HDAG Kernel better utilizes the richer structure present within texts.

References

M. Collins and N. Duffy. 2001. Parsing with a Single Neuron: Convolution Kernels for Natural Language Problems. In *Technical Report UCS-CRL-01-10*. UC Santa Cruz.

N. Cristianini and J. Shawe-Taylor. 2000. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

D. Haussler. 1999. Convolution Kernels on Discrete Structures. In *Technical Report UCS-CRL-99-10*. UC Santa Cruz.

T. Hirao, H. Kazawa, H. Isozaki, E. Maeda, and Y. Matsumoto. 2003. Machine Learning Approach to Multi-Document Summarization. *Journal of Natural Language Processing*, 10(1):81–108. (in Japanese).

S. Ikehara, M. Miyazaki, S. Shirai, A. Yokoo, H. Nakaiwa, K. Ogura, Y. Oyama, and Y. Hayashi, editors. 1997. *The Semantic Attribute System,* Goi-Taikei — A Japanese Lexicon, volume 1. Iwanami Publishing. (in Japanese).

H. Isozaki and H. Kazawa. 2002. Efficient Support Vector Classifiers for Named Entity Recognition. In *Proc. of the 19th International Conference on Computational Linguistics (COLING 2002)*, pages 390–396.

T. Kudo and Y. Matsumoto. 2002. Japanese Dependency Analysis using Cascaded Chunking. In *Proc. of the 6th Conference on Natural Language Learning (CoNLL 2002)*, pages 63–69.

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. 2002. Text Classification Using String Kernel. *Journal of Machine Learning Research*, 2:419–444.

G. Salton, A. Wong, and C. Yang. 1975. A Vector Space Model for Automatic Indexing. *Communication of the ACM*, 11(18):613–620.

V. N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer.

E. M. Voorhees and D. M. Tice. 1999. The TREC-8 Question Answering Track Evaluation. *Proc. of the 8th Text Retrieval Conference (TREC-8)*.