Tag-Annotated Text Search Using Extended Region Algebra

Katsuya MASUDA†a, Member and Jun’ichi TSUJII†b, Nonmember

SUMMARY This paper presents algorithms for searching text regions with specifying annotated information in tag-annotated text by using Region Algebra. The original algebra and its efficient algorithms are extended to handle both nested regions and crossed regions. The extensions are necessary for text search by using rich linguistic annotations. We first assign a depth number to every nested tag region to order these regions and write efficient algorithms using the depth number for the containment operations which can treat nested tag regions. Next, we introduce variables for attribute values of tags into the algebra to treat annotations in which attributes indicate another tag regions, and propose an efficient method of treating re-entrancy by incrementally determining values for variables. Our algorithms have been implemented in a text search engine for MEDLINE, which is a large textbase of abstracts in medical science. Experiments in tag-annotated MEDLINE abstracts demonstrate the effectiveness of specifying annotations and the efficiency of our algorithms. The system is made publicly accessible at http://www-tsujii.is.s.u-tokyo.ac.jp/medle/

doi: 10.1587/transinf.E92.D.2369

1. Introduction

Linguistic informations including part-of-speech, named entities, and syntactic/semantic structures of sentences have been found useful in many applications such as Question Answering, Information Retrieval, and Text Mining. However, because no effective and efficient methods for using various linguistic information in indexing a large set of texts exist, linguistic processing is usually applied on the fly to a set of texts after they are retrieved with a simple query. In consideration of the efficiency, the approach severely restricts the kind of linguistic processing that can be applied.

On the contrary, there is an emerging trend of enriching texts with different kinds of information in the form of the XML style tag-annotation, and using them for information services [1], [2], or for querying texts [3], [4]. The trend has brought on an exciting possibility in which sophisticated NLP techniques are applied in advance to make abstract levels of linguistic representation explicit, which are to be used to deduce more user-oriented information on the fly.

For such a scenario to work on a large textbase, one must devise access methods to use linguistic information computed in advance. To access the tag annotated data, we can apply XML databases, which have been developed based on frameworks including XPath [5], and XQuery [6]. However, since these XML databases are very generic for any type of application, their direct use leads to a rather inefficient system for large textbases. Text that are linguistically annotated by XML tags is neither typical of a data-oriented XML, or of a document-oriented XML. It does not have the homogeneity of data elements, which XML schema in data-oriented XML assume, while it has a much richer structure than those that a normal document-oriented XML can handle. As a result, the functionality provided for general XML databases is over specified for the specific use for large linguistically annotated textbases, and is thus inefficient. On the contrary, the functionality of full-text search engines with tags such as that of Lucene [7], is not expressive enough. In this paper, we propose a new retrieval system based on a region algebra [8], and show how operations of the region algebra can handle linguistic annotations effectively and efficiently.

We extend the original region algebra in order to treat the nested annotation; this is indispensable for dealing with the hierarchical nature of linguistic representation. Unlike previous work on nested annotation [9], our extension preserves the efficiency of the original framework. Further, our extension also allows for the use of local variables to denote shared structures, which are pervasive in semantic representation.

The access methods have been successfully embedded in our semantic search system for a large textbase: MEDLINE. We parsed the entire set of sentences in MEDLINE. The results of parsing were then transformed into in-line XML annotation, and stored. Our system is able to search texts from a large set of documents with complex queries, and output search results with the annotated linguistic information. We also illustrate some queries that exploit our extensions to region algebra, and report the performance of our system, which demonstrates the efficiency of our retrieval engine in real applications.

2. Tag-Annotation and Search

Consider the following sentences.
1. . . . requires p53 which is . . . to activate CD25
2. P53 does . . . by activating CD25.
3. CD25 can be activated . . . by p53

Although they contain the same essential pieces of information for biologists, “p53 activates CD25,” the surface
sequences of the words are very dissimilar. We cannot formulate a query in terms of the sequences of words or regular expressions. A proximity search with the words, "P53", "activate" and "CD25," will result in either a very low recall or a very low precision, depending on the width of the window.

In order to capture the fact that the three sentences contain the same piece of information, we need to go beyond even the simple constituent structure. The parser we used, Enju [10], is based on HPSG (Head-Driven Phrase Structure Grammar), which produces deeper representations (predicate-argument structure, PAS), as well as constituent structure. Figure 1 illustrates an example of Enju’s output with a tree structure, and Fig. 2 represents the corresponding in-line XML annotation.

By annotating text with the PAS information, we can search above three sentences with one query by specifying the annotated information. In our tag annotation, all phrases and words are assigned unique identifiers, and a relation between the phrases and words is expressed by using them. Note that, these tag-annotated texts do not have fixed formats of structures and have a large number of annotations unlike XML data that existing XML databases suppose.

In the following sections, we discuss how we have extended the original region algebra to construct an efficient and effective search system.

3. Region Algebra and Annotations

3.1 Region Algebra

In a tag-annotated text that is a linear sequence of words and tags, a position is attached to each appearance of words and tags, or characters, and a continuous sub-sequence is named a region. A region r is represented by a tuple (r.b, r.e), in which r.b and r.e are the begin and end positions of the region in the sequence. A region expressed by the tags can be any meaningful units such as title, section, or sentence.

Region Algebra by [8] defined a set of algebraic operators on the sets of regions. Operators in the original algebra are shown in Table 1. Operator $\bigodot$ (followed-by), for example, takes two sets of regions, $S_A$ and $S_B$, and produces a set of new regions ($S_{A\bigodot B}$), each of which starts with a region in $S_A$ and ends with one in $S_B$. More precisely, there are no intervening regions of $S_A$ or $S_B$ inside a new region in $S_{A\bigodot B}$.

That is, $A \bigodot B = \{(r_A.b, r_B.e) | r_A \in S_A, r_B \in S_B, s.t. r_A.e < r_B.b, and \exists r'_A \in S_A s.t. r_A.b < r'_A.b and r'_A.e < r_B.b, and \exists r''_B \in S_B s.t. r_A.e < r''_B.b < r_B.b$.}

Let us denote the set of regions of the start tag $<$A$, and one of the end tag $</A$, by $S(<A)$, and $S</A), respectively. Then, the set of tag-regions enclosed by $<A$ and $</A$, is expressed by $S(<A) \bigodot S</A)$, which is abbreviated as $[A]$ in the following: $[A \ att\_r1=val1, att\_r2=val2 \ldots]$ denotes $S(<A att\_r1=val1, att\_r2=val2 \ldots</A)$ $\bigodot S</A)$, where $att\_r$ expresses an attribute name and $val$ expresses an attribute value. In the following, we omit the $S$ function to simplify the expressions.

Since the operators are algebraic, query formulas may be recursively embedded in arguments. The query $[S] \bigtriangledown (activate \wedge CD3)$ represents a query to retrieve all “S” regions in which both the words “activate” and “CD53” appear. When the meaning of tag “<S>” is “sentence” in the definition of the annotation, this expression means “sentences containing the words ‘activate’ and ‘CD53’.”

The major attraction of the framework of region algebra by [8] is in its efficient algorithms for finding regions one by one that satisfy a query formulated in an algebraic formula. The algorithms for $\text{Containing}(\cdot)$ and $\text{Followed-by}(\cdot)$ are shown in Fig. 3. The algorithms for the other

![Fig. 1 Syntactic/semantic structure.](image)

![Fig. 2 Tag annotation of Fig. 1.](image)
operators can also be defined in similar ways by using the following functions:

- \( \tau(A \triangleright B, p) \): Return the first region of \( A \) beginning at or after \( p \).
- \( \rho(A, p) \): Return the first region of \( A \) ending at or after \( p \).
- \( \tau'(A, p) \): Return the last region of \( A \) ending at or before \( p \).
- \( \rho'(A, p) \): Return the last region of \( A \) beginning at or before \( p \), where \( p \) is a position, and \( A \) is a query.

Figure 4 illustrates how a region of ‘\( A \triangleright B \)’ can be found efficiently. In Fig. 4, the horizontal axis represents the position in the text. \( r_{a1}, r_{a2}, \ldots r_a \) are regions of ‘\( A \)’ and \( r_{b1} \) is a region of ‘\( B \)’. \( k \) is the starting position of the search. First, the algorithm finds the first region of ‘\( A \)’, \( r_{a1} \). Next, the algorithm finds the first region of ‘\( B \)’, \( r_{b1} \), from the beginning of \( r_{a1} \). The algorithm outputs \( r_{a1} \) if \( r_{a1} \) contains \( r_{b1} \). If \( r_{a1} \) does not contain \( r_{b1} \) (as in the case of Fig. 4), it looks for another region of ‘\( A \)’ where the end is nearest among ‘\( A \)’ regions after the end of \( r_{a1} \); that is, \( r_{a2} \), in Fig. 4. The algorithm then checks whether \( r_{a2} \) contains \( r_{b1} \). For the case illustrated in Fig. 4, since \( r_{a2} \) contains \( r_{b1} \), it outputs \( r_{a2} \), and repeats the same process until no region of ‘\( A \)’ or ‘\( B \)’ exists after the present position.

The above algorithm searches only one region nearest to the input position. In order to find all the regions satisfying the query, once it finds the \((i-1)\)-th region of \( A \) in the answer set, the algorithm repeats the same process with \( r_{a_{i-1}, b+1} \) as \( k \) to find the \( i \)-th region. That is,

1. Find the first region \( r_{a1} \) by \( \tau(A, 0) \), where 0 is the beginning of the textbase.
2. Find the region \( r_{a2} \) by \( \tau(A, r_{a1}, b+1) \) while the result region of \( \tau(A, r_{a2}, b+1) \) exists.

The algorithm is efficient since it can skip ‘\( A \)’ regions \( r_{a2}, \ldots r_{a_{i-1}} \) in Fig. 4. Since the algorithm is symmetrical for the two arguments, the order of complexity is proportional to the cardinality of the smaller set of the two argument sets. In particular, it is highly efficient when the cardinality of one of the two sets is very small. Note also that the algorithm computes regions from the beginning to the end of the textbase, and that the argument sets \( A \) and \( B \) are simultaneously computed from the beginning to the end by the algorithms for the corresponding algebraic operators.

However, the efficiency of the original algorithms in [8] are based on the assumption that the regions are never nested in the argument sets for operators. This assumption, which is invalid for linguistic annotation, makes skipping the regions of \( r_{a2}, \ldots r_{a_{i-1}} \) possible.

### 3.2 Nested Region Algebra and Re-Entrancy

While the original region algebra can treat rather flat sequences of regions like document structure (title, author, section etc.), structure of sentences that we would like to treat is very different in nature. Consider Fig. 2 of the XML representation of the parser’s output.

The major differences of structures in sentences from the document structure are:

1. **Nested Structures**: Since some regions contain smaller regions of the same tag, the algorithm by Clarke can not be applied to these texts directly.
2. **Re-Entrancy**: While the constituent structure can be specified by the containment relation, the semantic structure require a more powerful representation.

The algorithm for region algebra treating nested regions was proposed in [9]. The proposed algorithm can find all the regions for the argument sets that may contain nested regions. However, since it does not skip regions as is the case for the original one, it does not retain the same efficiency as the original one, which significantly reduces the attraction of region algebra for a large textbase.

As for re-entrancy, our representation uses node-identifiers assigned to all phrase nodes. For example, in Fig. 2, the NP node of P53 (id = 1) is shared by the two predicate nodes (id = “9” for “phosphorylate” and id = “15” for “activate”).

Re-entrancy also has implications on algorithms. We have to extend the algorithms in Sect. 3.1 without losing their simplicity and efficiency of the recursive simultaneous computation of the regions from left to right, by passing the position parameter of \( k \).

### 4. Extended Region Algebra

#### 4.1 Operations for Nested Regions

Figure 4 depicts how the original algorithm for the ‘containing’ operation fails when applied to nested tag regions. The cause of the failure is that, once it finds the first region of \( A \) \((r_{a1})\) containing a region of \( B \) \((r_{b1})\), it proceeds to the next \( A \)-region whose begin position is after the beginning position of \( r_{a1} \). This skipping is possible only when \( r_{a1} \) is not contained in another larger region of \( A \). In this example, it fails to recognize \( r_{a2} \) which properly contains \( r_{a1} \) and \( r_{b1} \), but whose begin position is before \( r_{a1} \).

In order to resolve this difficulty, we introduce ‘depth’
to distinguish between the different levels of the regions with the same tag in a nested construction, and we also introduce two new operators, “nested containing” (⫷), and “nested contained in” (⫶). The algorithms for the two nested operators use the depth to handle nested regions efficiently.

4.1.1 Functions for Region Algebra with Nested Tags

The depth of a region \( r \) of \([t]\), where \( t \) is a name of a tag, is defined as follows: When \( r \) is not contained in \( r' \) of \([t]\) and not overlapped by \( r'' \) of \([t]\) such that \( r'' \land b < r,b, r, depth = 0 \). Otherwise, \( r, depth = \max(r', depth) + 1 \) where \( r' \) of \([t]\) is a region such that \( r,b > r',b \) and \( r',b < r'',e \), which means that the region \( r' \) contains the region \( r \) or \( r' \) overlaps \( r \).

The depth defined by the above definition has the following characteristics:
1. A region \( r \) never contains or overlaps another region \( r' \) such that \( r, depth = r', depth \).
2. Only one or no region in the regions whose depth is \( d \) contains or overlaps \( r \) such that \( r, depth \neq d \).

By using the depth value, the algorithm searches a region efficiently.

Moreover, we extended the definition of the basic functions \( \tau, \rho, \tau', \) and \( \rho' \) with adding two arguments, a depth restriction and a region restriction as follows:

\[
\begin{align*}
\tau(A, p, d, r) : & \text{ Return the ‘first’ region of } A \text{ beginning at or after } p, \text{ whose depth is } d \text{ and contained in } r, \\
\rho(A, p, d, r) : & \text{ Return the ‘first’ region of } A, \text{ ending at or after } p, \text{ whose depth is } d \text{ and contained in } r, \\
\tau'(A, p, d, r) : & \text{ Return the ‘last’ region of } A, \text{ ending at or before } p, \text{ whose depth is } d \text{ and contained in } r, \\
\rho'(A, p, d, r) : & \text{ Return the ‘last’ region of } A, \text{ beginning at or before } p, \text{ whose depth is } d \text{ and contained in } r.
\end{align*}
\]

The order of the region used in the above definition is in the following:

\[
\begin{align*}
\tau, \rho' : & \ r < r' \Leftrightarrow r.b < r',b \land (r.b = r',b \land r.e > r',e) \\
\rho, \tau' : & \ r < r' \Leftrightarrow r.e < r',e \land (r.e = r',e \land r.b > r',b).
\end{align*}
\]

The first and second arguments, which represent the query expression and the begin position of search respectively, are the same as for the original function, and the third argument is the depth which the output region must be, and the fourth argument is for the region restriction in which the output region must be contained. When the depth and region restriction are unspecified, the functions are exactly the same as for the corresponding functions in the original version.

4.1.2 Algorithms for Nested Regions

Figures 5 and 6 show the algorithms for the two operators: containing and contained in, for the nested regions. The first step of the function \( \tau(A \bowtie B, p, d, r) \) is similar to the original algorithm. The algorithm searches the first region \( r_a \) of \( A \) with \( \tau \) function and the first ending region \( r_b \) of \( B \) from \( r_{a1}, b \). It outputs \( r_{a1} \) if \( r_{a1} \) contains \( r_b \), since no region of \( A \) exists whose begin position is between \( p \) and \( r_{a1}, b \). When \( r_{a1} \) does not contain \( r_b \), the algorithm searches for the next candidate region \( r_{a2} \), which is the first ending region of \( A \) from \( r_{b1}, e \).

The algorithm checks if \( r_{a2} \) contains \( r_b \) and stores \( r_{a2} \) as a candidate of the output \( r_{output} \) if \( r_{a2} \) contains \( r_b \). Then, the algorithm searches a larger region of \( A \) \( (r_{a3}) \), whose begin position is before \( r_{a2}, b \) and the nearest to \( r_{a2} \), with changing the depth restriction from the \( r_{a2}, depth - 1 \) to 0, and stores \( r_{a3} \) as \( r_{output} \) if \( r_{a3} \) contains \( r_b \) and the region \( r_{b2} \). If \( r_{output} \) exists after the search of larger region \( r_{a3} \) is finished, \( r_{output} \) is returned as an output. Otherwise, the algorithm searches the next region for \( A \bowtie B \) from the position \( p_{next} \). In this case, no region \( r' \) for \( A \bowtie B \) such that \( p < r', b < p_{next} \) exists because no region \( r' b \) of \( B \) such that \( p < r', b < r_b, e \) and no region of \( A \) contains \( r_b \).
regions of $A$. The algorithm does not need to check other regions of $A$.

The first step of the algorithm for the function $\tau(A \sqcap B, p, d, r_c)$ is also similar for the original algorithm. The algorithm searches the first region $r_a$ of $A$, and the first ending region $r_b$ of $B$ from $r_a$, and outputs $r_a$ if $r_a$ is contained in $r_b$. If $r_a$ is not contained in $r_b$, it searches for a region $r_b$ of $B$ whose begin position is before and the nearest to $r_a$, with changing depth restriction and checks whether $r_b$ contains $r_a$. When the region $r_b$ containing $r_a$ exists, the algorithm outputs $r_a$. If no region of $B$ contains $r_a$, the algorithm searches the next region for $A \sqcap B$. In the case there are regions of $A$ contained in $r_a$, the next search starts from the next position of $r_a, b$ to check these regions of $A$. Otherwise, the next search starts from $p_{next}$ calculated while the loop because if there is a region $r'_b$ of $B$ containing a region $r'_a$ of $A$ such that $r'_a, b < p_{next}$, $r'_a$ is contained in $r_a$ or $r'_b$ should be searched as $r_b$ or $r_b$ in the algorithm. Algorithms for the other functions are also constructed in a similar way.

We developed algorithms for the operators $\diamond$ and $\triangle$ to search innermost regions which satisfy the description in Table 1.

Because the above $\tau$ functions output the region whose beginning position is nearest to the start search position in regions that satisfy the query condition, we can calculate all regions satisfying the query condition by the same algorithm in the case of the non-nested regions except for the case that the regions expressed with $\diamond$ or $\triangle$ contains other regions in the query.

4.2 Queries with Local Variables

In tag-annotation of sentential structure as in Fig. 2, identifiers are assigned to tags for expressing a relation between tag regions. For example, the relation, “a phrase is the deep subject of a verb,” is expressed by the equality of the value of the attribute $arg$ in a word tag for the verb with the value of the attribute $id$ in the phrase in Fig. 2. When we search the relation, we need to express the equality and confirm the equality of the two values. By using the variables, we can express the equality easily in queries such as: `((word arg=x base="activate") $\triangle$ (phrase id=x1) $\triangleright$ "p53")" where $x$ is a variable. We suppose that variables appear only in an attribute value. The re-entrancy indicates that single nodes in a syntactic tree can be pointed by an arbitrary number of nodes in different positions in the tree as their semantic arguments.

To express the instantiation of variables simply, we express the query containing variables as $Q(x_1, \ldots, x_n)$, where $x_i$ is a variable. For example, the above query can be simplified to $Q(x)$.

We define the result of search with the query $Q(x_1, \ldots, x_n)$ as

$$ S(Q(x_1, \ldots, x_n)) = \bigcup_{a_1 \in A} \ldots \bigcup_{a_n \in A} S(Q(a_1, \ldots, a_n)), $$

which means a set of the results retrieved by the query $Q(a_1, \ldots, a_n)$, which is the query $Q(x_1, \ldots, x_n)$ instantiated with an instantiation $[a_1/x_1, \ldots, a_n/x_n]$. The straightforward treatment of variables leads to either non-deterministic algorithms for complex variable binding, or to a kind of unrestricted joins between independently computed sets of regions. Either of these approaches results in inefficient algorithms. This result is because the use of variables destroys the restricted interdependency between the sub-queries, which the original algorithms of the region algebra cleverly exploit. In other words, sub-queries in the original version only need to communicate the position $k$ from which they start to search.

4.3 Algorithms for Queries with Variables

4.3.1 Main Algorithm

The main algorithm to search all regions of $S(Q(x_1, \ldots, x_n))$ is in the following description. Here, we assume that a textbase is a collection of texts and that the texts are annotated by tags. All variables have the same region of scope. This means that all variables are local, in the sense that their scopes are inside the same text, and that the re-entrancy by the same ID is only valid in the scope. For example, the scope region will be “sentence” when the algorithm searches “sentences.”

[Step 1] Enumeration of Scope Regions

Search a scope region from the current position of the textbase. The current position is initially set to the beginning of the textbase. When no scope regions are left at the end of the text collection, terminate.

[Step 2] Generation of Instantiations

Generate an instantiation $[a_1/x_1, \ldots, a_n/x_n]$ for the variables for the scope region chosen at [step 1] and go to [step 3].

If there are no instantiations left, move the current position to the next position of the beginning of the current scope region chosen at [Step 1], and return to [Step 1].

[Step 3] Evaluation of Instantiated Queries

Compute $S(Q(a_1, \ldots, a_n))$ in the scope region, add them to the solution set, and return to [Step 2].

The generator of an instantiation at [Step 2] works as a co-routine with the main routine, and when it is called, generates the next instantiation. The basic generator generates all possible assignments of values in the domain. However, more efficient algorithms choose a set of sub-queries...
that generate effective instantiations for the set in succession when called.

4.3.2 Enumeration of Scope Regions

Since the equality of the value of a variable is only valid in its scope, the enumeration of effective instantiations should be carried out inside a scope region. Therefore, effective instantiations are to be computed in a scope region, and $S(Q(x_1, \ldots, x_n))$ is to be computed inside the same region. [Step 1] enumerates all scope regions, and passes them to [Step 2] one by one.

Nevertheless, it is inefficient to evaluate an original query against all scope regions in a textbase. Most of the scope regions in a textbase can be easily filtered out by approximated queries, which can be derived from an original query. We used keyword search query as an approximated query in the current implementation. The query is constructed by connecting the words in the original query with $\triangle$ and $\triangledown$ with retaining the condition of words that a scope region should contain.

4.3.3 Generation of Instantiations

We can compute $S(Q(x_1, \ldots, x_n))$ by computing $S(Q(a_1, \ldots, a_n))$ for all possible instantiations of $[a_1/x_1, \ldots, a_n/x_n]$. However, we do not need to consider all possible instantiations, but rather only the instantiations that are consistent with instantiations for a given set of sub-queries. In practice, we can enumerate a set of instantiations to be considered systematically, by choosing a set of appropriate sub-queries.

By definition, variables appear only as the values of attributes in tags such as, $[\text{tag attr}_1 = x_1, \ldots, \text{attr}_m = x_m]$. We can create an instantiation for all variables by picking up some sub-queries in the form of the above query as all variables appear in at least one sub-query and calculating the variable value from the queries.

From a sub-query $[\text{tag attr}_1 = x_1, \ldots, \text{attr}_m = x_m]$, we can create an instantiation $[a_1/x_1, \ldots, a_m/x_m]$ by the following algorithm:
1. Create a query $[\text{tag}]$ by removing attributes containing variable from the sub-query $[\text{tag attr}_1 = x_1, \ldots, \text{attr}_m = x_m]$.
2. Calculate a region $r$ from the query $[\text{tag}]$. Extract the value $a_j$ for $x_j$ from the corresponding text for $r$.

In the above, sub-queries are in the form of $[\text{tag attr}_1 = x_1, \ldots, \text{attr}_m = x_m]$. However, the above algorithm can be applied to more general form of sub-queries. We call such queries as value determining query and define recursively as follows:
1. $[\text{tag attr}_1 = x_1, \ldots, \text{attr}_m = x_m]$
2. $Q_r \sqsubseteq Q$

where $Q_r$ is a value determining query, and $Q$ is a query that contains no variables. $\sqsubseteq$ is one of the containment operators, $\text{Containing}$ ($\triangleright, \triangleright\triangleright$) or $\text{Contained in}$ ($\triangleright, \triangleleft$). By adding the condition of $Q$ in the above definition, the number of candidate instantiations decreases because the number of regions retrieved with the query decreases.

4.3.4 Evaluation Order of Sub-Queries

Consider the following query: $[\text{sentence}] \triangleright (([\text{word arg} arg_1 = x_1, \text{arg} 2 = x_2] \triangleright ([\text{phrase id} \triangleright = x_1] \triangleright p_53) \triangleright ([\text{phrase id} \triangleright = x_2] \triangleright \text{cd} 25))$.

This query retrieves all sentences in which $p_53$ and $\text{cd} 25$ appear in the phrases of the deep subject, and the deep object of a predicate, respectively. Actual predicates are not specified. The following five sub-queries can be chosen as value determining sub-queries: (1) $[\text{phrase id} = x_1]

\triangle (2) [\text{phrase id} = x_2]

\triangle (3) [\text{word arg} arg_1 = x_1 \text{arg} 2 = x_2]

\triangle (4) [\text{phrase id} = x_1] \triangleright p_53
\triangle (5) [\text{phrase id} = x_2] \triangleright \text{cd} 25$. Any sets of sub-queries containing at least one occurrence of each of the two variables can be used by the generator of effective instantiations. However, the sub-queries (1) and (2) generate instantiations for the variables from all the phrases, while the sub-queries (4) and (5) generate far less instantiations, leading to an efficient evaluation of $Q$. The sub-query (3) may also generate many instantiations, since it can be matched with all predicative words in a sentence, and since it is less restrictive than (4) and (5). Once either (4) or (5) is used to produce an instantiation, it can be used to partially instantiate (3). Then, (3) will become more restrictive.

In practice, as the first value determining sub-query, we choose the query that is estimated as most restrictive. Then, the first sub-query is used to produce an effective instantiation. By using the instantiation, we partially instantiate the original query, and choose the next value determining sub-query. The process is repeated to obtain an effective interpretation for all the variables in the original query.

However, since it takes time to determine the order for each instantiation, we estimate the frequency of the instantiated value for each variable as $\text{effreq}(\text{Sub}Q_j) = \min_{\text{word} \text{Sub}Q_j} \text{freq}(w)$, where $\text{Sub}Q_j$ is a sub-query which is evaluated to determine the value for $x$ and $\text{freq}(w)$ is the frequency of word $w$ in the textbase. By this estimations, we can determine the evaluation order of sub-queries from the frequency of words in advance. Figure 8 shows the algorithm for the evaluation order.

Figure 8. The algorithm for the evaluation order.
5. Experiments

We developed a text search engine by implementing our algorithms. This engine is a core component of our intelligent search engine, MEDIE (http://www-tsujii.is.s.u-tokyo.ac.jp/medie/), designed to search sentences from MEDLINE. In our system, we can specify biomedical events by entering a subject, an object, and a verb. The system outputs sentences containing the "subject-verb-object" relation specified by the user. In the following experiments, we used only a part of MEDLINE data because of the difficulty of applying the existing XML database to whole MEDLINE data and manual judgement.

In the following experiments, we used a computer that had Opteron 2.8 GHz CPU, and 16 GB memory. The system was implemented in C++.

MEDLINE is a collection of abstracts in the biomedical field. It contains about 16 million articles, a half of which have abstracts, with the other half only containing a title. We parsed all of these MEDLINE abstracts by Enju [10], and annotated the documents with the parsing results. Table 2 shows the size of data that we used in the MEDIE system. We constructed 213 GB of position-depth database, and 291 GB of attribute-value database from 799 GB of tag-annotated texts. (Some of the unnecessary attributes in tag-annotated texts are removed while indexing.) These tag-annotated texts are constructed by annotating the 13 GB of text with the results of parsing, the results of named entity recognition and tags annotated in the original MEDLINE data. The position-depth database stored the list of positions and depth data for each word or tag, and used to search a position of words or tags. Attribute value database stored the map from the position to the attribute value data, and used while the calculation of variable values.

In the following experiments, we used only a part of MEDLINE data because of the difficulty of applying the existing XML database to whole MEDLINE data and manual judgement.

5.1 Efficiency

Table 3 shows the search times for three queries in our system, and an existing XML database, MonetDB/XQuery [11]. We indexed 150,000 MEDLINE articles in both systems for this experiment. We converted the queries into XQuery form as shown in Table 4 and applied to MonetDB. The results showed that the search time of both systems was nearly linear to the number of abstracts, and that our system (‘Our’) could search documents in a shorter amount of time than MonetDB in our tasks.

‘Our (Cache)’ denotes that search time of our system
when the data concerning to the query processing is cached on memory. This search time is the ideal search time, which means that the time to access disks is nearly 0. Because the time to access disks is a large part of search time, we will be able to reduce the search time by improving the access method to disks. When the system applied to the real application, the search time will be between the time ‘Our (Cache)’ and ‘Our,’ because some of the data will be cached on memory.

5.2 Effectiveness of Specifying Structure

We evaluated the effectiveness of specifying the predicate argument structure by comparing the accuracy of “keyword search” with that of “semantic search,” which executes queries specifying structures. Table 5 shows a list of queries used in the comparison of two types of search and search results. “*” indicates that any word can appear. These queries were selected by biologists, and express relational concepts that biologists want to find. The system outputs sentences containing the relations of the query in a “semantic search,” and outputs the sentences containing all words of the query in “keyword search.” In “semantic search,” the system converts the query into a expression like 

\[
[senten]\Rightarrow (\text{word arg1=}$subject$\text{ arg2=}$object$ base=verb) \Delta (\text{phrase cat=}$NP$\text{ id=}$subject$}) \Rightarrow (\text{word} \Rightarrow \text{subject}) \Delta (\text{phrase cat=}$NP$\text{ id=}$object$})
\]

At most, 100 sentences were retrieved for each query, and the results of two types of search were merged and shuffled. A biologist judged whether these sentences contain an expression that represents all of the relations described in the query, and the accuracy was measured by the judgement. The results in Table 5 show that semantic search exhibited impressive improvements in precision, thus indicating that specifying the predicate argument structure is an effective way to retrieve relations in the document.

5.3 Discussion about Annotations

We annotated the documents with large amount of parsing results, but it seems to be too many annotations for searching only “subject-verb-object” relation. In the example of data in Fig. 2, the phrase tags whose id number is 2, 4, 5, 7, 8, 11, 13, 14 and 17 can be eliminated because these phrases does not correspond to the “subject-verb-object” relation. Furthermore, the attributes ‘head’ and ‘token’ also can be eliminated in the same reason. By eliminating such tags and attributes, the size of data will decrease, and the search time will be also reduced.

But by using these annotations, we can search with other types of queries. For example, when users want to search sentences containing “colon cancer,” users can send a query [sentence] \( \Rightarrow (\text{colon} \Delta \text{cancer}) \), which expresses sentences containing both “colon” and “cancer,” as shown in the query 5 in the experiments. But users can also specify the relation “colon modifies cancer” \( \Rightarrow (\text{colon} \Rightarrow \text{cancer}) \), by a query [sentence] \( \Rightarrow (\text{colon} \Rightarrow \text{cancer}) \), because such relation is expressed by “mod” attribute in the data as shown in Fig. 10.

6. Conclusion and Future Work

The algorithms in this paper are implemented as MEDIE system and successfully dealt with a large textbase. We also showed that the search based on linguistic annotation could significantly improve the precision and recall in sentence retrieval. However, several functionalities are lacking in the current system. The most critical lack of functionality is the difficulty of adding new annotations to existing textbases. When we add a new set of tags to existing textbase, we need to reconstruct the whole index data. We are planning to redesign the whole system by using stand-off annotation. Another lack of functionality is that the system supports only exact match search and outputs the result in the order of appearance in the database. We plan to incorporate scoring method using annotated informations to our system.

### Table 5

| Query                                | # results | Keyword search time (first/all) | precision | Semantic search time (first/all) | precision |
|--------------------------------------|-----------|--------------------------------|-----------|---------------------------------|-----------|
| * inhibit ERK2                      | 252       | 0.00 s/ 1.5 s                  | 74/100 (74%) | 143                              | 0.01 s/ 2.5 s | 96/100 (96%) |
| * trigger diabetes                  | 125       | 0.00 s/ 1.8 s                  | 45/100 (45%) | 27                               | 0.02 s/ 2.9 s | 23/27 (85%)   |
| adiponectin increase *              | 287       | 0.00 s/ 1.5 s                  | 20/100 (20%) | 30                               | 0.05 s/ 2.4 s | 25/30 (80%)   |
| macrophage induce *                 | 10698     | 0.00 s/ 42.8 s                 | 14/100 (14%) | 1559                             | 0.01 s/ 301.45 s | 65/100 (65%) |
| * suppress MAP phosphorylation      | 87        | 0.04 s/ 2.7 s                  | 34/ 87 (39%) | 15                               | 0.05 s/ 4.2 s | 10/15 (67%)   |
| * enhance p53 (negative)            | 1812      | 0.01 s/ 7.6 s                  | 19/100 (19%) | 84                               | 0.20 s/ 29.2 s | 73/84 (87%)   |

Fig. 10 Example of data which express “modification”.

### References

[1] D. Ferrucci and A. Lally, “UIIM: An architectural approach to unstructured information processing in the corporate research environment,” Natural Language Engineering, vol.10, no.3-4, pp.327–348, 2004.

[2] H. Mima, S. Ananiadou, G. Nenadic, and J. Tsujii, “A methodology for terminology-based knowledge acquisition and integration,” Proc. Coling 2002, pp.667–673, 2002.

[3] W. Alink, V. Jijkoun, D. Ahn, M. de Rijke, P. Boncz, and A. de Vries, “Representing and querying multi-dimensional markup for question
answering,” Proc. 5th Workshop on NLP and XML (NLPXML-2006), pp.3–9, April 2006.
[4] P. Siniakov, “Querying XML documents with multi-dimensional markup,” Proc. 5th Workshop on NLP and XML (NLPXML-2006), pp.43–50, 2006.
[5] J. Clark and S. DeRose, “XML Path Language (XPath) Version 1.0,” 1999.
[6] D. Chamberlin, “XQuery: An XML query language,” IBM Syst. J., pp.597–615, 2002.
[7] B. Goetz, “The Lucene search engine: Powerful, flexible, and free,” JavaWorld, 2000.
[8] C.L.A. Clarke, G.V. Cormack, and F.J. Burkowski, “An algebra for structured text search and a framework for its implementation,” Comput. J., vol.38, no.1, pp.43–56, 1995.
[9] J. Jaakkola and P. Kilpeläinen, “Nested text-region algebra,” Tech. Rep. C-1999-2, University of Helsinki, 1999.
[10] Y. Miyao and J. Tsujii, “Probabilistic disambiguation models for wide-coverage HPSG parsing,” Proc. ACL 2005, pp.83–90, 2005.
[11] P. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner, “MonetDB/XQuery: A fast XQuery processor powered by a relational engine,” Proc. 2006 ACM SIGMOD International Conference, pp.479–490, June 2006.

Katsuya Masuda is a project researcher in Center for Knowledge Structuring, University of Tokyo. He received BSc, MSc degrees from the University of Tokyo in 2002, 2004, respectively. His research interests include Information Retrieval and Information Extraction.

Jun’ichi Tsujii is Professor of Natural Language Processing in Interfaculty Initiative in Information Studies, University of Tokyo. He is also scientific director of the National Centre for Text Mining (NaCTeM) and professor of Text Mining in the School of Computer Science University of Manchester. He has worked since 1973 in Natural Language Processing, Question Answering, Text Mining and Machine Translation, He was Associate Professor of Kyoto University from 1979 to 1988 with a break at CNRS Grenoble, France as invited senior researcher in 1981–1982. Before taking up the position at the University of Tokyo, He was Professor of Computational Linguistics of University of Manchester Institute of Science and Technology (UMIST) in the UK from 1988 to 1995, where he kept the position of Research Professor until 2001. He was President of ACL (Association for Computational Linguistics, 2006) and President of IAMT (International Association for Machine Translation 2002-2004). His recent research achievements include Deep semantic parsing based on feature forest model, Efficient search algorithms for statistical parsing, Improvement of estimator for maximum entropy model, and Construction of the gold standard corpus for Bio Text Mining.