博士論文

SEMANTIC SEARCH USING ANNOTATIONS BY NATURAL LANGUAGE PROCESSING: PAPER SEARCH BASED ON EVENTS IN BIOMEDICAL SCIENCE

（自然言語処理アノテーションを利用した意味検索: 生命医学系論文に対する事象に基づく検索）

増田 勝也

ABSTRACT

This dissertation aims to implement a high-accuracy semantic search system by incorporating various natural language processing (NLP) techniques, and we proposes a retrieval framework for textbases which are tag-annotated with the result of natural language processing (NLP) technique in order to achieve the semantic search system. Basic NLP systems, such as part-of-speech taggers, named entity recognizers, syntactic/semantic parsers, etc. has been developed in recent years, and the accuracy of these systems is sufficient to use in more advance NLP applications. However, because there is no effective and efficient frameworks for storing and using the results from these basic NLP systems comprehensively, one usually applies linguistic processing on the fly to a set of text after they are retrieved by simple method, such as keyword-based search. On the other hand, there is an emerging trend of enriching text with various kinds of information as annotations and using or sharing the annotations of information or knowledge in various research area not only the NLP area.

This dissertation proposes a high-accuracy semantic search system incorporating NLP technologies, and shows effectiveness of NLP technologies for information retrieval. As a framework that implements the semantic search system, we propose a a semantic retrieval framework which realized advanced retrieval as compared with existing keyword-based retrieval system by specifying annotated information of the result of various types of NLP modules. In order to realize a semantic retrieval system using annotations of NLP results, we extend a region algebra and its algorithm, which is a search framework for textbases structured with tag-annotations. The extended framework can be applied to the annotations of NLP results which contain nesting structure in the annotations, and enables us to search using references by variable. Moreover, we proposes ranking retrieval for by extending the probabilistic model for keyword-based retrieval into a retrieval framework in which direct dependency is supposed among a set of structured queries.

As a real world application, we implements a semantic retrieval system for MEDLINE databases, which is a database of paper abstracts in the biomedical area. We apply a deep parser and a named entity recognizer as a basic NLP modules, and recognizers of events in biomedical research area, such as a protein-protein interaction recognizer, as advanced NLP modules to the documents in MEDLINE in order to annotate the NLP informations. By realizing semantic retrieval which using annotated information from NLP modules, we construct a retrieval system which enables us to search events like protein-protein interaction which is important in biomedical area, and demonstrate the effectiveness of the proposing search framework.

In the experiments, we evaluated accuracy improvement of our semantic search system on our own test data and the publically usable test collection, which is created in TREC Genomic Tracks, and shows effectiveness of incorporating NLP technologies into search systems. Moreover, we declare effectiveness of our framework by compare the framework with existing XML databases and evaluate our framework and algorithms as a retrieval framework for annotated documents.

論文要旨

本論文では自然言語処理技術を用いた高精度な意味検索システムの実現を目的として、自然言語処理技術を用いてアノテーション情報が付与されたテキストに対し、そのアノテーション情報を検索時に統合的に利用するための意味検索枠組みを提案する。近年、品詞タガーや構文解析器、固有表現認識器等の自然言語処理分野における基本的なシステムが開発され、十分な精度でより高度な自然言語処理アプリケーションにおいて利用されている。しかしながらそれらの基本的な自然言語処理システムの結果を統合的に利用・管理する効果的な枠組みが存在しないため簡単な絞り込みによる検索の後に自然言語処理を行うという手法が主流である。また一方でテキストに対し多種多様な情報をアノテーションとして付与し、利用・共有を行うという動向が自然言語処理分野のみならず様々な分野において存在する。

そこで本論文では、これらの種々の自然言語処理技術を統合的に利用した高度な意味検索システムを実現し、情報検索に対する自然言語処理技術の有用性を示すことを目的とする。また実現するための手法として、自然言語処理モジュールを利用してテキストに対して言語的情報を付与し、その情報を利用して従来のキーワードベースの検索に比べより高度な検索を実現する枠組みを提案する。アノテーションにより構造化されたテキストに対する検索枠組である領域代数を拡張し、自然言語処理アノテーションの特徴である入れ子構造に対応した検索アルゴリズム、および変数による参照を利用可能な検索枠組を構築する。また、従来のキーワードベースの検索で使用される確率的言語モデルを拡張し、依存関係が存在する、構造を持つクエリ集合を利用した検索に対するランキング検索手法を提案する。

提案システムの実世界への適用例として、生医学論文の要旨データベースである MEDLINE に対し検索システムを実装する。自然言語処理の基本的なモジュールとして、深い構文解析器、固有表現認識器を利用し、さらにはより高度なモジュールとしてタンパク質間相互作用等の生医学研究における event の認識器を利用し、それらの処理結果をテキストにアノテーションとして付与する。付与されたアノテーション情報を利用した検索を可能とすることで、生医学研究において重要とされる物質間の相互作用等の検索を可能とするシステムを構築し、本論文において提案する意味検索システムの有用性を示す。

実験において、提案する意味検索システムの検索精度の評価を独自に作成したテストデータ、および情報検索評価用のテストコレクションを用いて行い自然言語処理を利用した意味検索システムの有効性を示す。また、既存の XML データベースとの比較や種々のクエリによる実験を行い、アノテーションが付与されたテキストを対象とした、検索枠組・アルゴリズム自体の評価を行い、提案枠組の有用性を示す。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Basic natural language processing (NLP) systems, such as part-of-speech taggers, named entity recognizers, syntactic/semantic parsers, etc. has been developed, and the accuracy of these systems is sufficient to use in other applications. Linguistic information extracted by using these basic systems has been found useful in many QA, IR and Text Mining applications. However, because there is no effective and efficient frameworks for storing and using the results from these basic NLP systems comprehensively, one usually applies linguistic processing on the fly to a set of text after they are retrieved by simple method such as keyword-based search, or creates purpose-built database for each system.

The "retrieve-then-process" model is inevitable when one has to use a search engine as black box to retrieve the initial relevant set from a huge collection of text. However, the approach severely restricts linguistic processing that one can apply, e.g. one cannot apply computationally expensive processing on the fly, such as deep semantic parsers, etc, since a set of the documents retrieved with the simple query condition is very huge. Although the expensive processing can be executed in advance by using the purpose-built database, the database does not have expandability. When new NLP systems are developed, the database has to be rebuilt in order to incorporate the NLP systems into the search system.

On the other hand, there is an emerging trend of enriching text with various kinds of information in XML-style tag annotations and using them for intelligent information services [48, 18, 28]. The trend has opened up an interesting perspective in which sophisticated NLP technologies are applied to text in advance to make high levels of linguistic representation such as syntactic and semantic structures explicit, which in turn are to be used to deduce more user-oriented information on the fly.

In this dissertation, we propose a semantic search system using natural language processing. We constructed a framework of document search for tag-annotated documents, and implemented a semantic search system for articles in biomedical domain. In the system, the articles are processed with various types of NLP systems and annotated with the results from these modules. Text and annotations in the articles are indexed in advance, and the system searches articles matching to structured queries, which can specify the relationship between keywords, by using the index in search process.

1.1 Semantic Search

Traditional retrieval systems are constructed based on keywords. These systems construct index for keywords, the systems searches documents by calculating the relevance between a query and a document. The query and the document are considered as a set of keywords, and the relevance between a query and a document is calculated as a relevance between the keyword sets. Although these keyword-based search systems are very simple, it is difficult to specify search intention of users in a query since the user cannot specify relationship among keywords.

In recent years, some retrieval systems introduces relationship among keywords in queries and indices [72, 17, 15, 12]. These systems are called "Semantic Search" systems, which are defined as search concepts or technologies which improve search accuracy by recognizing intention of users from the queries or contexts and searching documents or information along the users' intention. In a semantic search system, we can specify complex condition by using relationship among keywords in linguistic information, such as the subject-verb-object relationship from syntactic/semantic parsers, the attributes of the keywords or more complex representation from NLP systems. By incorporating linguistic information in the search process, we can acquire search results with high accuracy which matches to our search intention.

These systems process target documents by various types of NLP systems, mainly syntactic parsers, in advance, and construct indices for both keywords and the results from NLP systems. In search process, these systems receives keywords or a natural language sentence as a query and process the query in the same way with documents. Then the system calculates the relevance between a document and a query using the results from NLP systems. These systems recognize users' intention by processing input keywords or sentences by using NLP systems. Our system is one of Semantic Search systems. The target documents are processed in advance by NLP systems, and annotated with XML-style tag annotations of the result by these modules. We constructed a search framework for tag-annotated documents, which constructs indices for keywords and annotations and searches documents using the indices. Since our framework employed tag-annotation for uniformed representation of the results from NLP systems, these NLP results can be managed easily, for example, these result can be integrated in search process. Moreover, since we adapted our framework to stand-off annotations, we can introduce newly developed NLP systems into the system easily by adding the stand-off annotations.

1.2 Search Framework for Tag-Annotated Documents

For documents structured with tags, effective and efficient access methods have to be devised to exploit linguistic information computed in advance. The access models for XML such as XPath [13] and XQuery [11] are neither effective nor efficient for accessing relevant information through convoluted structures of syntactic and semantic representations. Furthermore, significant revisions of XML-based access models are necessary if one allows independently motivated annotations to co-exist in the same text as envisioned in UIMA [18]. Even if each single layer of annotation is conformant to XML such as non-cross border restriction on tags, there is no guarantee that a group of autonomous annotations as a whole is XML conformant.

We proposed a search framework which can use annotated information in search process comprehensively [45]. In the framework, we employed XML-style tag annotations for representing and storing linguistic information for target documents, and also employed region algebra and its algorithms [14] for querying documents with specifying the annotated linguistic information in search process comprehensively. In order to apply the region algebra into annotated linguistic information, we extended an access method for tag-annotated documents based on region algebra [14] which provides a conceptually simpler and more efficient framework for convoluted structures of language and independently motivated annotations. We extend the original region algebra in order to treat nested annotations, which are indispensable for dealing with the hierarchical nature of linguistic representation. Unlike previous work of nested annotation [29], our extension preserves the efficiency of the original region algebra. Our extension also allows using logical variables for denoting shared structures which are pervasive in semantic representation. In the experiments, we evaluated the framework itself by search speed by comparing with existing XML databases.

Moreover, we proposed a ranking retrieval for our semantic search system [44]. We extended a probabilistic model for keyword-based retrieval into our retrieval framework, in which queries used in scoring are not keywords but structured queries specifying annotations. Although we can suppose independency among keywords in keyword-based retrieval, we cannot ignore dependency among structured queries because the dependency is directly expressed in the queries in our framework. So we incorporated the dependency into a probabilistic retrieval model, and derived a scoring function for retrieval using structured queries. In the experiments, the ranking algorithm is evaluated as a accuracy of a whole semantic search system in biomedical domain, since the accuracy of ranking is depend on target data.

1.3 Search Systems in Biomedical Science

In biomedical science area, the number of journal articles is already numerous and rapidly increasing. These articles are collected into a large database, MEDLINE. The MEDLINE database has around 20 million of articles in biomedical area, and these articles come from wide variety of research fields. Since the huge number and variety of research area of articles, it is hard for researchers to find articles themselves or knowledges written in articles concerning their fields. Computer-assisted access for the articles are needed to acquire target articles or knowledges.

Although there is the official primary search service for MEDLINE, PubMed, the search function is only keyword-based search, and the search results are ordered only in chronological order. In order to support researchers to find articles or knowledges from the MEDLINE database, various types of search systems have been developed for the database. Several systems outputs results with ranking based on the relevance of articles to search intention of users by using information retrieval techniques [76, 19, 3]. Some systems enriches search results by analysis using NLP techniques, especially analysis of biological entities or relationships between concepts [35, 61].

Our semantic search system, *MEDIE* [51, 56], constructed by implementing our proposed search framework, is one of the sohpisticated search systems for the MEDLINE database. In the MEDIE system, the target text from MEDLINE were annotated with results from various kinds of NLP systems, such as a deep semantic parser, named entity recognizer, and other advance NLP systems which recognize biological events. The main search function of the MEDIE system is to search “subject-verb-object” relation. Users can specify the “subject,” “verb,” “object” as an input, and the system searches sentences or articles which contain the relation specified in the input.

In the experiments, we evaluated our semantic search system from the several point of view. First, we evaluated our system as a whole semantic search system, that is, we evaluated the improvement of accuracy by incorporating linguistic information into the search process. We evaluated the accuracy in two types of data, which is constructed by us, which contains the queries and the judgment of relevant documents assigned by specialists in biological domain by hand, and the pubically usable test collection, which is constructed in TREC Genomic Track. In the evaluation on originally constructed data, we evaluated accuracy of exact match search of our semantic search system. In the evaluation on the test collection of TREC Genomic Track, we evaluated accuracy of semantic search system with raking retrieval, and show the applicability of our system into general task.

1.4 Organization of The Dissertation

The remainder of this dissertation is divided into the following chapters. The chapter 2 describes the background of our proposal. First, we explain about the current status of researches in information retrieval with natural language processing. Next, we explain about the retrieval for structured document, such as XML databases, XML retrieval, and other frameworks. Finally, we shows the researches which are the base of our research, a region algebra and its search algorithm, which is a representation of queries for structured documents and basic algorithm for our research, and the probabilistic retrieval model for keyword-based search, which is extended for structured document retrieval in our research.

The chapter 3 describes main retrieval method in our research. We decries the extension of the region algebra and its algorithm in order to search documents with XML-style annotations of NLP results. We extended the region algebra and its algorithm which can apply to textbases with annotations from NLP systems. First, we describes the extension in search algorithm for nested regions. Next, we proposes the incorporation of variables into the region algebra and an algorithm to calculate values for variables and to search regions for the queries containing variables. We explain the index structure and its implementation which enables the above two extension for the region algebra, and we also proposes an extension for the algorithm, which can be applied to the documents with stand-off annotations. Finally, we proposes the ranking retrieval model for the structured query in region algebra by extending the probabilistic model for keyword-based search into the structured queries.

The chapter 4 shows the actual semantic search system, MEDIE, in which the proposed framework is implemented. The MEDIE system is a semantic search system for the MEDLINE database, which is a database of paper abstracts in bio-medical domain. We shows the detail of the MEDIE system in the chapter. We first describe the overview of the MEDIE system, and explain about the NLP systems which is applied to the MEDLINE abstracts in indexing phase, such as the syntactic/semantic parser, named entity recognizer, protein-protein recognizer, event expression recognizer and sentence role classifier. We also describes the database structure of the MEDIE system, in which our proposed retrieval framework is implemented, and the interface of the MEDIE system.

The chapter 5 describes experiments on our semantic search system. We conducted two types of experiments, the evaluation of our system as a semantic search system, and the evaluation of our algorithms used in our system. In the former experiment, we evaluated accuracy of our semantic search system, on two types of search, the exact match retrieval and ranking retrieval. The target documents of the experiment are the MEDLINE abstract databases, and the documents are annotated by NLP systems used in MEDIE system. For evaluation of the exact match retrieval, we used queries and relevant judgment of documents created on our own. For evaluation of the ranking retrieval, we used the publically usable test collection created by TREC Genomic Track. In the latter experiments, we compared our framework and other existing XML databases, MonetDB/XQuery [7] and eXist [46]. Since these XML databases cannot store data invalid XML, in which the tag-annotated regions crosses with each other, we apply only the syntactic/semantic parser to the target data in the experiment. We discussed about the scalability of our system, test collection for semantic search and search accuracy by using NLP results. The final chapter concludes the dissertation with some future works for the problems discussed in the previous chapter.

Chapter 2

Background

2.1 Information Retrieval and Natural Language Processing

Recently, various kinds of document retrieval system are developed. Google is the most famous and powerful retrieval system for web pages. The web pages are ordered based on the PageRank [58], which is calculated based on the number of hyperlinks from the page and into the page. Lucene [22] and Solr are open source full-text search systems developed by Apache Software Foundation. Although these retrieval system are powerful, many of these retrieval systems can receive only a set of keywords, or keywords with some attributes such as "the 'title' field contains the word 'retrieval' " as a query, and retrieve all the documents which satisfy the query or a ranking list of documents in the order of the relevance of the documents to the query, which is defined in each retrieval system. These retrieval systems cannot specify the relationship among words because the system indexed only words of the texts.

On the other hand, some retrieval systems introduce relationship among words in search algorithm [72, 17, 15, 12] These systems are called "Semantic Search" systems. Semantic search is defined as search concepts or technologies which improve search accuracy by recognizing intention of users from the queries or contexts and searching documents or information along the users' intention. In order to recognize the intent of users and search documents along the intention, various types of NLP technologies are applied. For example, word sense disambiguation is applied in order to recognize the users' intention in the query correctly. Synonym or ontology is used to search documents in which the keywords in queries are written in different representation. Moreover, by annotating documents with meta-data, system can search the documents with enriched information. The concept of semantic search for Web is called "Semantic Web" [6]. In the Semantic Web, the contents in the web is written in XML and annotated with meta-data by using RDF [36] or OWL [4]. Since the annotations considered in the Semantic Web is a simple data containing attributes like meta-data, more complex and huge annotations like annotations from NLP technologies is difficult to applied to the Semantic Web framework.

Some of search systems incorporate NLP technologies into search process.

TSUBAKI [72], which is a search system for web pages in Japanese, is one of the systems which include the relation between the keywords into scoring of pages. The system parses the text in web pages by Japanese dependency parser KNP [37], and constructs indices of words, synonym and dependency relation between the words. In the search phase, the system receive a natural language sentence as a input query, and analyze the input sentence with the same dependency parser used in indexing. Then the relevance between the query and the documents are calculated using the frequency of both words and dependency in the documents. Powerset [15], which is already acquired by Microsoft, is a similar engine in English. The target documents of Powerset are English Wikipedia pages. Users can query not only keyword set but also a natural language sentence question into the system, and the system searches documents relevant to the the query using NLP technologies, such as paring and synonyms. These systems using NLP technologies are very powerful to search not a document but a information which is relevant to the query sentence or can be an answer to the question of a query.

In the area of researches whose target is texts, there is an emerging trend of enriching text with various kinds of informations. The format commonly used is XML-style annotation, which is specify the text region with a start tag and an end tag, and expresses the annotated information with a tag name and attribute values in start tags. Figure 2.1 shows an example of XML-style annotations. Annotations for text are created by both automatically and manually. In the natural language processing area, text are enriched by the annotations expressing natural language processing results, for example the part-of-speech or named entities, and the annotations are used as an input of other NLP systems [48, 18]. For the types of information which is difficult to create automatically, the text are annotated by hand. This types of annotations are mainly used as a training data for machine learning. In Digital Humanities area, which is an area of research concerned with the intersection of computing and the disciplines of the humanities, Text Encoding Initiative consortium [28] standardized the annotation method of historical documents in humanities in order to share the documents themselves and the knowledge about them, and the digitalizing and annotating tools are developed. The trend of enriching text has opened up an interesting perspective in which sophisticated NLP technologies are applied to text in advance to make high levels of linguistic representation such as syntactic and semantic structures explicit, which in turn are to be used to deduce more user-oriented information on the fly.

2.2 Structured Document Retrieval

One of simple implementation of semantic search system is a structured document retrieval. Text of target documents are enriched with structure of documents and annotation of more detailed information by NLP.

```
<sentence>
 <phrase id="0" cat="S" head="4" lex_head="6">
  <phrase id="1" cat="NP" head="2" lex_head="3">
   <phrase id="2" cat="NP" head="3" lex_head="3">
    <word id="3" pos="NN" cat="NP" base="p53">p53</word>
   </phrase>
  </phrase>
 <phrase id="4" cat="VP" head="5" lex_head="6">
  <phrase id="5" cat="VP" head="6" lex_head="6">
   <word id="6" pos="VBZ" cat="VP" base="is" arg1="1" arg2="7">
   is
   </word>
  </phrase>
  <phrase id="7" cat="VP" head="8" lex_head="9">
   <phrase id="8" cat="VP" head="9" lex_head="9">
    <word id="9" pos="VBN" cat="VP" base="phosphorylate"
          arg2="1" arg1="-1" arg3="10"
          rel_type="phosphorylation">
    phosphorylated
    </word>
   </phrase>
   <phrase id="10" cat="VP" head="11" lex_head="12">
    <phrase id="11" cat="VP" head="12" lex_head="12">
     <word id="12" pos="TO" cat="VP" arg1="1" arg2="13">
     to
     </word>
    </phrase>
    <phrase id="13" cat="VP" head="14" lex_head="15">
     <phrase id="14" cat="VP" head="15" lex_head="15">
      <word id="15" pos="VB" cat="VP" base="activate" arg1="1"
            arg2="16">
       activate
      </word>
     </phrase>
     <phrase id="16" cat="NP" head="17" lex_head="18">
      <phrase id="17" cat="NP" head="18" lex_head="18">
       <word id="18" pos="NN" cat="NP" base="cd25" arg1="1">
        CD25
       </word>
      </phrase>
     </phrase>
    </phrase>
   </phrase>
  </phrase>
 </phrase>
</sentence>
```

Figure 2.1: Example of XML annotations

2.2.1 XML Database and XML Retrieval

The most commonly used framework for annotating or structuring documents is XML. The access methods for the well-formed XML are defined as query languages, XPath[13] or XQuery[11]. XPath is a query language which point a specific element in XML documents. For example, the query

```
/sentence/phrase[@id='0']/phrase[@id='1']/phrase[@id='2"]/word
```

in XPath expresses the element of "word" tag for the word "p53" in Figure 2.1. The query specify the elements of "word" tag, which is contained in three "phrase" tags whose value of id attribute is "2," "1" and "0", and the sentence tag. The containment relation of "phrase" tags is also specified in the expression, The "phrase" tag whose id is "2" is contained in the "phrase" tag whose id is "1,", and the "phrase" tag whose id is "1," is contained in the "phrase" tag whose id is "0." The "phrase" tag whose id is "0" is contained in the "sentence" tag. XPath expression can specify the element in the XML documents by the containment relation of tags.

XQuery is a query languages for XML documents to extract and manipulate data in XML documents. By XQuery, we can not only search elements in the XML Documents but create a formalized data from retrieved elements by using SQL-like expressions such as `For`, `Let`, `Where` and `Return`. For example, the XQuery expression

```
For $i in document("sample.xml")//word
  Let $a := $i[@id]
  Return $a
```

output the "id" attribute of "word" tags in the XML document. Many XML databases in which XPath or XQuery can be used as a query language have been implemented [7, 46].

One of the free XML databases is MonetDB [7]. MonetDB itself is not a XML databases but a column-oriented database management system. But the extension for XQuery, MonetDB/XQuery, can store XML data in column-oriented databases, and users can query stored XML data with XQuery language. The XPath and XQuery are query languages for accessing XML documents, by regarding the documents as databases. On the other hand, some researches for XML document search have been proposed, which regards the XML documents as a text documents with annotations of document structure. This type of search for XML documents is called structured document retrieval. For the structured document retrieval, INEX (Initiative for the Evaluation of XML retrieval) [5] have created test collections of retrieval for XML documents. In INEX, the participants assign judgment for a element of XML documents whether the element is relevant to the information need expressed in queries.

Although the XML databases or XML retrieval systems are very effective and efficient to search structured documents, it is difficult to apply these systems to the documents with NLP annotations. The main reason is the characteristics of NLP annotations, such as the nesting of tag annotations and overlapping of annotated regions. For example, the paring results of the text have a nesting structure of "phrases" as shown in Figure 2.1, and when the results of different types of NLP modules, such as parsing results and named entities, are annotated to the same texts, there are no guarantee that the annotated regions from two modules are never crossed each other.

Text that are linguistically annotated by XML tags is neither typical of a data-oriented XML, which is typified by the XML documents accessed by using XPath or XQueries, or of a document-oriented XML, which is typified by the target documents in INEX. 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 [7, 46] are over specified for the specific use for large linguistically annotated text bases, and is thus inefficient. on the contrary, the functionality of full-text search engines with tags such as that of Lucene [22], is not expressive enough.

2.2.2 Other Frameworks on Retrieval for Annotated Documents

Some retrieval systems for structured documents which does not suppose well-formed XML structure have been proposed. These systems suppose only the documents are structured or annotated with some informations. Indri [47, 2] is a search system for structured documents based on the inference network. The Indri system does not suppose the well-formed XML, and can handle partially structured documents such as HTML documents. The results of Indri is ranking list of documents which are relevant to a query. The query language of Indri can specify the terms, proximity of terms, synonyms, context that words appear and weighting method of the terms etc. Although the language can specify the context of words, for example, "the word 'dog' appears in the 'title' " and the system enables a ranking retrieval, the query cannot specify the detailed context for the words, and the relation between words except for proximity.

The Region Algebra [14, 10] is a framework for specifying the structure of documents by operating sets of text fragments, which are called as "regions" in the framework, in the documents. The efficient algorithm for search documents or the matched regions in a document is also proposed. We employ the framework of the Region Algebra because the retrieval by operation of sets of regions is directly matches to our targets, which are documents structured with data-structure tags and annotated with results from NLP applications. Although this framework search documents or part of documents efficiently, the query language,

Name(Expression)	Description
Containing ($> AB$)	'A' regions containing a 'B' region.
Contained In($< AB$)	'A' regions contained in a 'B' region.
Followed By($-AB$)	Regions beginning with a 'A' region and ending with a 'B' region.
Both Of ($\&AB$)	Regions beginning with a 'A' region and ending with a 'B' region and the reverse.
One Of($\|AB$)	Regions of 'A' regions or 'B' regions.

Table 2.1: Operators of Region Algebra

i.e. region algebra itself, cannot specify a relation between words except for the relation that words are in the same regions, for example, "the words 'information' and 'retrieval' appear in the same sentence." We incorporated several aspects to search documents annotated with results from NLP modules specifying the relation between words expressed in the annotations.

2.3 Region Algebra and Annotations

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 "*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, or more detailed informations.

Region Algebra by [14] defined a set of algebraic operators on the sets of regions. Operators in the original algebra are shown in Table 2.1. The input for the operators are supposed a set of regions in which no region contains other regions, that is, an input set of regions S is expressed as $\{r | \nexists r' \text{ s.t. } r.b < r'.b < r'.e < r.e\}$. Operator $-$ (followed-by), for example, takes two sets of regions, S_A and S_B, and produces a set of new regions (S_{A-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-B}. That is,

$$\begin{aligned}(\text{- A B}) = \{(r_A.b, r_B.e) \mid\ & r_A \in S_A,\ r_B \in S_B, \text{ s.t. } r_A.e < r_B.b,\\ & \text{and } \nexists r'_A \in S_A \text{ s.t. } r_A.b < r'_A.b \text{ and } r'_A.e < r_B.b,\\ & \text{and } \nexists r'_B \in S_B \text{ s.t. } r_A.e < r'_B.b < r_B.b\}.\end{aligned}$$

The operations of the other for operators are also defined in a similar way as follows:

$$\begin{aligned}(> \text{A B}) = \{r_A \mid\ & r_A \in S_A,\ r_B \in S_B, \text{ s.t. } r_A.b < r_B.b < r_B.e < r_A.e,\\ & \text{and } \nexists r'_A \in S_A \text{ s.t. } r_A.b < r'_A.b < r_B.b < r_B.e < r'_A.e < r_A.e\}\\ (< \text{A B}) = \{r_A \mid\ & r_A \in S_A,\ r_B \in S_B, \text{ s.t. } r_B.b < r_A.b < r_A.e < r_B.e,\\ & \text{and } \nexists r'_A \in S_A \text{ s.t. } r_B.b < r_A.b < r'_A.b < r'_A.e < r_A.e < r_B.e\}\end{aligned}$$

(& A B) $= \{(min(r_A.b, r_B.b), max(r_A.e, r_B.e)) \mid r_A \in S_A,\ r_B \in S_B,$
s.t. $\nexists r'_A \in S_A$ s.t. $min(r_A.b, r_B.b) < r'_A.b < r'_A.e < max(r_A.e, r_B.e)$,
and $\nexists r'_B \in S_B$ s.t. $min(r_A.b, r_B.b) < r'_B.b < r'_B.e < max(r_A.e, r_B.e)\}$
(| A B) $= \{r \mid r \in S_A \cup S_B$ s.t. $\nexists r' \in S_A \cup S_B$ s.t. $r.b < r'.b < r'.e < r.b\}$.
Note that there is no nesting region, which means a region containing other regions in a set, in a result set of region by all operations.

A set of tag-annotated regions is expressed by using the followed by operator. 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 `(- <A> </A>)`, which is abbreviated as [A] in the followings. When the start tag contains attributes, that is, the start tag is `<A attr_1=val1 attr2=val2 ...>`, a set of tag-regions enclosed by the tag is expressed by `(- <A attr1=val1 attr2=val2 ...> </A>)`, where `attri` expresses an attribute name and `vali` expresses the corresponding attribute value, and we also abbreviated this expression as `[A attr1=val attr2=val2 ..]`.

Since the operators are algebraic, query formulas may be recursively embedded in arguments. The query `(> [S] (& activate CD25))` represents a query to retrieve all "S" regions in which both the words "activate" and "CD25" appear. When the meaning of tag `<s>` is "sentence" in the definition of the annotation, this expression means "sentences containing the words 'activate' and 'CD25'. "

```
$\tau((> AB), p)$
  $r = \tau(A, p)$
  **return** $\rho((> AB), r.e)$

$\rho((> AB), p) =$
  $r = \rho(A, p)$
  $r' = \tau(B, r.b)$
  **if** $r'.e \leq r.e$ **then**
    **return** $r$
  **else**
    **return** $\rho((> AB), r'.e)$
```

```
$\tau((< AB), p) =$
  $r = \tau(A, p)$
  $r' = \rho(B, r.e)$
  **if** $r'.b \leq r.b$ **then**
    **return** $r$
  **else**
    **return** $\tau((< AB), r'.b)$

$\rho((< AB), p)$
  $r = \rho(A, p)$
  **return** $\tau((< AB), r.b)$
```

Figure 2.2: τ and ρ functions for containing ($>$) and contained in ($<$) operators

The major attraction of the framework of region algebra proposed by [14] is in its efficient algorithms for finding regions one by one that satisfy a query formulated in an algebraic formula. The algorithms consists of combination of four functions, τ, ρ, τ' and ρ'. The definition of the τ, ρ in the following:
$\tau(A, p)$: 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.

```
$\tau((\&AB),p)$
  $r = \tau(A,p)$
  $r' = \tau(B,p)$
  $r'' = \tau'(A, max(r.e, r'e))$
  $r''' = \tau'(B, max(r.e, r'e))$
  **return** $(min(r''.b, r'''.b),$
                $max(r''.e, r'''.e))$

$\rho((\&AB),p) =$
  $r = \tau'((\&AB), p-1)$
  **return** $\tau((\&AB), r.b+1)$
```

```
$\tau((|AB),p)$
  $r = \tau(A,p)$
  $r' = \tau(B,p)$
  **if** $r.e < r'.e$ **then**
    **return**$r$
  **elseif**$r.b > r'b$ **then**
    **return**$r'$
  **else**
    **return**$(max(r.b, r'.b), r.e)$

$\rho((|AB),p)$
  $r = \rho(A,p)$
  $r' = \rho(B,p)$
  **if** $r.e < r'.e$ **then**
    **return**$r$
  **elseif**$r.b > r'b$ **then**
    **return**$r'$
  **else**
    **return**$(max(r.b, r'.b), r.e)$
```

Figure 2.3: τ and ρ functions for bothof (&) and one of (|) operators

```
$\tau((-AB),p)$
  $r = \tau(A,p)$
  $r' = \tau(B, r.e+1)$
  $r'' = \tau'(A, r'.b-1)$
  **return** $(r''.b, r'.e)$
```

```
$\tau((-AB),p)$
  $r = \tau'((-AB), p-1)$
  **return** $\tau((-AB), r.b+1)$
```

Figure 2.4: τ and ρ functions for followed-by ($-$)

where p is a position, and A is a query. The algorithms of function τ and ρ for each operator are shown in Figure 2.2, 2.3 and 2.4. The algorithms for τ' and ρ' functions is defined in a similar way since the tau' and rho' functions are symmetry of τ and ρ function respectively.

Figure 2.5 shows a simple example of text annotated with tags expressing document structure. Index number is attached to each word appearance. In the case of the text of Figure 2.5, when the query is '(> [title] retrieval)', the process to find the regions matching to the query is shown in Figure 2.6 (we omit the process to find the regions matching the query '[title]' from the process).

Because the algorithm operates only a set of non-nested regions, these functions can find the region incrementally. The time complexity of the algorithm is linear to the number of the query nodes and the frequency of the word whose frequency is lowest in the query except for the "One Of" (|) operation. In the case of the "One Of" operation, the time complexity is linear to the sum of the size

⟨book⟩	⟨title⟩	ranking	retrieval	⟨/title⟩	⟨chapter⟩
1	2	3	4	5	6
⟨title⟩	tf	and	idf	⟨/title⟩	ranked
7	8	9	10	11	12
retrieval	⟨/chapter⟩	⟨/book⟩	⟨book⟩	⟨title⟩	structured
13	14	15	16	17	18
text	⟨/title⟩	⟨chapter⟩	⟨title⟩	search	for
19	20	21	22	23	24
structured	text	⟨/title⟩	retrieval	⟨/chapter⟩	⟨/book⟩
25	26	27	28	29	30

Figure 2.5: An example of text and positions of words

```
$\tau$( (> [title] "retrieval"), -)
    $(p_b, p_e) = \tau$([title], -) = (2,5)
    $\rho$((> [title] "retrieval"), 5)
        $(p_b, p_e) = \rho$([title] , 5) = (2,5)
        $(p'_b, p'_e) = \tau$("retrieval", 2) = (4,4)
        $p'_e < p_e$ is true
        return (2,5)
$\tau$( ( > [title] "retrieval"), 6)
    $(p_b, p_e) = \tau$([title], 6) = (7,11)
    $\rho$( (> [title] "retrieval"), 11)
        $(p_b, p_e) = \rho$([title], 11) = (7,11)
        $(p'_b, p'_e) = \tau$("retrieval", 7) = (13,13)
        $p'_e < p_e$ is false
        $\rho$(( > [title] "retrieval"), 13)
            $(p_b, p_e) = \rho$([title], 13) = (17,20)
            $(p'_b, p'_e) = \tau$("retrieval", 17) = (-, -)
            return (-, -)
```

Figure 2.6: Example of the process finding the regions matching the query

of the frequency of the words. This is because the number of the exact matches is linear to the frequency of the least frequent word, and calls of access functions $(\tau, \rho, \tau', \rho')$ in the process to search a exact match is linear to the number of nodes in the query as shown in Figure 2.6.

Figure 2.7 illustrates how a region of '> A B' can be found efficiently in the algorithm. In Figure 2.7, the horizontal axis represents the position in the text. r_{a_1}, r_{a_2} ... r_{a_n} are regions of 'A,' and r_{b_1} is a region of 'B.' k is the starting position of the search. First, the algorithm finds the first region of 'A,' r_{a_1}. Next, the algorithm finds the first region of 'B,' r_{b_1}, from the beginning of r_{a_1}. The

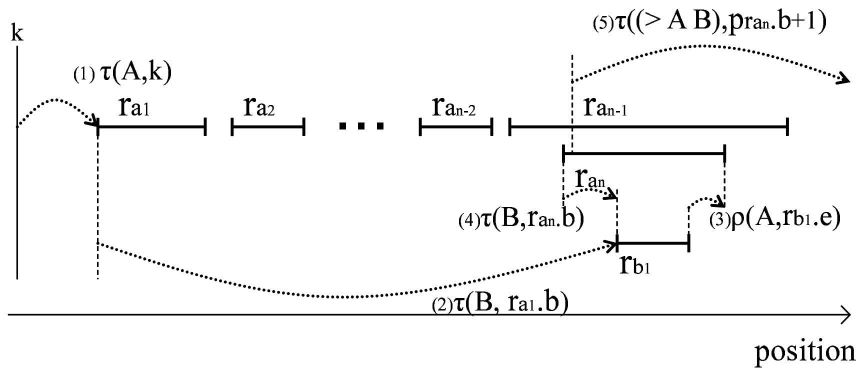


Figure 2.7: Evaluating the function 'τ(> A B,p)'

algorithm outputs r_{a_1} if r_{a_1} contains r_{b_1}. If r_{a_1} does not contain r_{b_1} (as in the case of Figure 2.7), it looks for another region of 'A,' where the end is nearest among 'A' regions after the end of r_{b_1}; that is, r_{a_n} in Figure 2.7. The algorithm then checks whether r_{a_n} contains r_{b_1}. For the case illustrated in Figure 2.7, since r_{a_n} contains r_{b_1}, it outputs r_{a_n}, 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 finds 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 p to find the i-th region. That is,
1. Find the first region r_{A_1} by $\tau(A, 0)$, where 0 is the beginning of the textbase.
2. Find the region r_{A_i} by $\tau(A, r_{A_{i-1}}.b{+}1)$ while the result region of $\tau(A, r_{A_{i-1}}.b+1)$ exists.

The algorithm is efficient since it can skip 'A' regions r_{a_2} ... $r_{a_{n-2}}$ in Figure 2.7. 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 [14] 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_{a_2}, ... $r_{a_{n-2}}$ possible.

2.4 Ranking Retrieval Model

The users would like to find only the documents which is especially relevant to the users' information request. The ranking retrieval systems calculate the relevance of documents and users' requests, and rank the documents in the order of the relevance. The modeling of queries and documents, and the relevance functions for ranking documents have been studied.

2.4.1 Vector Space Model

The traditional standard model for modeling the documents and queries is vector space model [69, 16], which model the documents and queries with the vector of weights of words appeared in the document and queries,

$$v_d = (w_1^d, w_2^d, ..., w_n^d)$$

$$v_q = (w_1^q, w_2^q, ..., w_n^q)$$

where d is a document, q is a query, w_i^d is a weight of the word w_i in the document d, and w_i^q is a weight of the word w_i in the query q.

The most standard value for the weight of words is TF-IDF weighting [32, 70]. TF (Term Frequency) is a weighting method of words based on the frequency of words in the documents or query. The motivation of the TF weighting is that "the words appeared many times in the documents or query are important in them," that is, the TF value expresses the local importance of words in documents or queries. The TF weighting have some variations, such as the boolean value, the frequency of words itself(f_w), the logarithmically scaled frequency $(1+\log(f_w))$, the normalized frequency($\frac{f_w}{\max_w f_w}$), and so on. IDF (Inverse Document Frequency) is a weighting method os words based on the document frequency of words, which means the number of documents in which the words appear, in the whole target document set. The motivation of the IDF weighting is the "the words appeared only the small number of documents are important in the document set," the IDF value expresses the global importance of words in the whole document set, that is different from the TF value. The IDF weighting also have some variations, such as the inverse number of document frequency ($\frac{1}{df_w}$), inverse number of normalized document frequency ($\frac{N}{df_w}$) or logarithmically scaled these values ($\log \frac{1}{df_w}$, $\log \frac{N}{df_w}$). TF-IDF weighting combined these two weight, TF and IDF weighting, by multiplying the two values.

In the vector space model, the relevance between a document d and a query q is defined as a similarity of two vectors. The similarity of two vectors is defined as the cosine measure,

$$Sim(d, q) = \frac{v_d \cdot v_q}{|v_d||v_q|}$$

The TF-IDF weighting is used in various retrieval systems and based on the empirical assumption, but this weighting method have only little theoretical grounding. On the other hand, probabilistic model, which is constructed based on the assumption that whether the document is relevant or not is determined stochastically, are proposed [43, 20].

2.4.2 Probabilistic Retrieval Model for Keyword-based Search

The ranking retrieval is modeled as follows:
For a query q and a document d, R is defined as a variable that expresses whether the document d is relevant to the given query q. R take a binary value, i.e, R takes 1 when the document d is relevant and R takes 0 otherwise. In the probabilistic model, the ranking should be created by ranking documents in the order of the estimated probability $P(R = 1|d, q)$. This is a basis of *Probability Ranking Principle*, which is proposed in [62, 63].

One of the probabilistic model traditionally used for keyword-based retrieval is the *Binary Independent Model* [75, 64, 65]. In this model, the document d and query q are modeled by a vector whose elements is a binary value, whether the word appears or not in the document or query. That is,

$$d = (w_1^d, w_2^d, ..., w_n^d)$$

$$q = (w_1^q, w_2^q, ..., w_n^q)$$

where w_i^d and w_i^q are binary values which express the appearance of a word t_i in the document d and the query q respectively, that is, $w_1^d = 1$ when the term t_i appears in the document d and $w_1^d = 0$ when the term does not appear.

This model supposes two assumptions. One is the independency of words, that is, the words appears in a document independently of the other words. The other is the independency of relevance, that is, the relevance between a document and a query is determined independently of the other documents.

In order to make ranking list based on the Probability Ranking Principle, we would like to calculate the probability that the document d is relevant to the query q, $P(R = 1|d, q)$. The probability that the document d is relevant to the query q ($P(R = 1|d, q)$) and that the document is not relevant ($P(R = 0|d, q)$) can be deformed by using Bayes rules,

$$P(R = 1|d, q) = \frac{P(d|R = 1, q)P(R = 1|q)}{P(d|q)}$$

$$P(R = 0|d, q) = \frac{P(d|R = 0, q)P(R = 0|q)}{P(d|q)}.$$

In order to ranking the documents based on the relevance to the query, we have to calculate the probability ($P(R = 1|d, q)$). However, since only the order of documents is important for the ranking list of documents, the odds of relevance,

$$O(R|d, q) = \frac{P(R = 1|d, q)}{P(R = 0|d, q)}$$

is used for making ranking list in order to simplify calculation. The probability that the documents and the odds of relevance give the same ordering of the documents and the calculation of the value became easier because the common denominator can be ignore:

$$\begin{aligned} O(R|d,q) &= \frac{P(R=1|d,q)}{P(R=0|d,q)} && (2.1)\\ &= \frac{\frac{P(d|R_{d,q}=1,q)P(R=1|q)}{P(d|q)}}{\frac{P(d|R_{d,q}=0,q)P(R=0|q)}{P(d|q)}} && (2.2)\\ &= \frac{P(R=1|q)}{P(R=0|q)} \cdot \frac{P(d|R=1,q)}{P(d|R=0,q)} && (2.3)\end{aligned}$$

Moreover, in the above formula, the left term $\frac{P(R=1|q)}{P(R=0|q)}$ is a constant for a query q, only the right term,

$$\frac{P(d|R=1,q)}{P(d|R=0,q)}$$

should be estimated for each document in order to create ranking results.

This value is deformed by using the *Naive Bayes conditional independence assumption*, which is the assumption that the appearance of a word is independent of the appearance of other words:

$$\frac{P(d|R=1,q)}{P(d|R=0,q)} = \prod_{i=1}^{n} \frac{P(w_i^d|R=1,q)}{P(w_i^d|R=0,q)}$$

Since w_i^d is a binary value, the above formula can be deformed as follows:

$$\frac{P(d|R=1,q)}{P(d|R=0,q)} = \prod_{i:w_i^d=1}^{n} \frac{P(w_i^d=1|R=1,q)}{P(w_i^d=1|R=0,q)} \cdot \prod_{i:w_i^d=0}^{n} \frac{P(w_i^d=0|R=1,q)}{P(w_i^d=0|R=0,q)}$$

Here, let $p_i = P(w_i^d = 1|R=1,q)$, which is the probability of a term appearing in a document relevant to the query, and $u_i = P(w_i^d = 1|R=0,q)$, which is the probability of a term appearing in a nonrelvant document. With the simplifying assumption that the terms not appearing in the query appear in relevant and non-relevant documents in the same probability, that is, $p_i = u_i$ when $w_i^q = 0$, the above formula can be transformed,

$$\begin{aligned} & \prod_{i=w_i-d=1}^{n} \frac{P(w_i^d=1|R=1,q)}{P(w_i^d=1|R=0,q)} \cdot \prod_{i:w_i^d=0}^{n} \frac{P(w_i^d=0|R=1,q)}{P(w_i^d=0|R=0,q)}\\ =& \prod_{i:w_i^d=w_i^q=1} \frac{p_i}{u_i} \cdot \prod_{i:w_i^d=0,w_i^q=1} \frac{1-p_i}{1-u_i}\\ =& \prod_{i:w_i^d=w_i^q=1} \frac{p_i(1-u_i)}{u_i(1-p_i)} \cdot \prod_{i:w_i^q=1} \frac{1-p_i}{1-u_i}\end{aligned}$$

Since the right product term, $\prod_{i:w_i^q=1} \frac{1-p_i}{1-u_i}$, is a constant when a query is given, only the left product term should be estimated in order to make ranking list of documents. *Retrieval status value* (RSV) is defined by the logarithm of the term,

$$RSV_d = \log \prod_{i:w_i^d=w_i^q=1} \frac{p_i(1-u_i)}{u_i(1-p_i)} = \sum_{i:w_i^d=w_i^q=1} \log \frac{p_i(1-u_i)}{u_i(1-p_i)}.$$

Here, we define the value c_i for a term t_i as follows:

$$c_i = \log \frac{p_i(1-u_i)}{u_i(1-p_i)} = \log \frac{p_i}{1-p_i} + \log \frac{1-u_i}{u_i}.$$

This value can be calculated using the number of documents in the collections as follows:

$$c_i = \log \frac{p_i(1-u_i)}{u_i(1-p_i)} = \log \frac{\frac{s}{S-s}}{\frac{df_{t_i}-s}{(N-df_{t_i})-(S-s)}}$$

where N is the number of documents in the collections, df_{t_i} is the number of documents in which appears the term t_i, S is the number of documents relevant to the query q and s is the number of documents which are relevant to the query q and in which the term t_i appears. In practice, since the relevant documents account for only a small percentage in a document collection, statistics for non-relevant documents can be estimated by statistics for the whole document collections. Under this assumption, the probability u_i that the term t_i appears in non-relevant documents for a query can be estimated $\frac{df_{t_i}}{/}N$, and the left term in c_i definition becomes as follows:

$$\log \frac{1-u_i}{u_i} = \log \frac{(N-df_{t_i})}{df_{t_i}} \approx \log \frac{N}{df_{t_i}}$$

This formula can provide a theoretical justification for IDF weighting.

Although Binary Independent Model supposes the assumption that appearance of words is independent from the appearance of words, this assumption is not true in actual documents. For example, the words "information retrieval" are frequently cooccured in documents because the words are compound words. Some researches have been studied incorporating dependency between words into a probabilistic retrieval model [74, 39, 67].

Chapter 3

Extended Region Algebra for Retrieval using Annotations by NLP

This chapter proposes the model and algorithms for the retrieval of tag-annotated documents. Before we propose the model and algorithms, we explain characteristics of semantic retrieval using NLP results and functions required to realize the semantic retrieval. Then we describes the extension of a region algebra and its algorithms, and probabilistic model for retrieval of documents with annotations of NLP results.

3.1 Annotations by NLP and Semantic Retrieval

Consider the following sentences in the biological domain.

1. ... requires <u>p53</u> which is ... <u>to activate</u> <u>CD25</u>
2. <u>P53</u> does ... <u>by activating</u> <u>CD25</u>.
3. <u>CD25</u> can be <u>activated</u> ... by <u>p53</u>

Although they contain the same essential pieces of information which biologists would like to search, “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, because of the lack of the specification of relations between the words.

In order to capture the fact that these three sentences contain the same piece of information, we need to go beyond even the simple constituent structure. For example, the parser Enju[50], which is based on HPSG(Head-Driven Phrase Structure Grammar), produces deeper representations (predicate-argument structure, PAS), as well as constituent structure. Figure 3.1 illustrates an example of Enju’s output with a tree structure for the sentence “P53 is phosphorylated to activate CD25.” S, NP, and VP in Figure 3.1 are called phrase markers, which characterize the syntactic units (phrases) under them. S, NP, and VP mean Sentence, Noun Phrase, and Verb Phrase, respectively. The syntactic structure is expressed by the tree structure, and the semantic structure is expressed by arrows with “arg1”

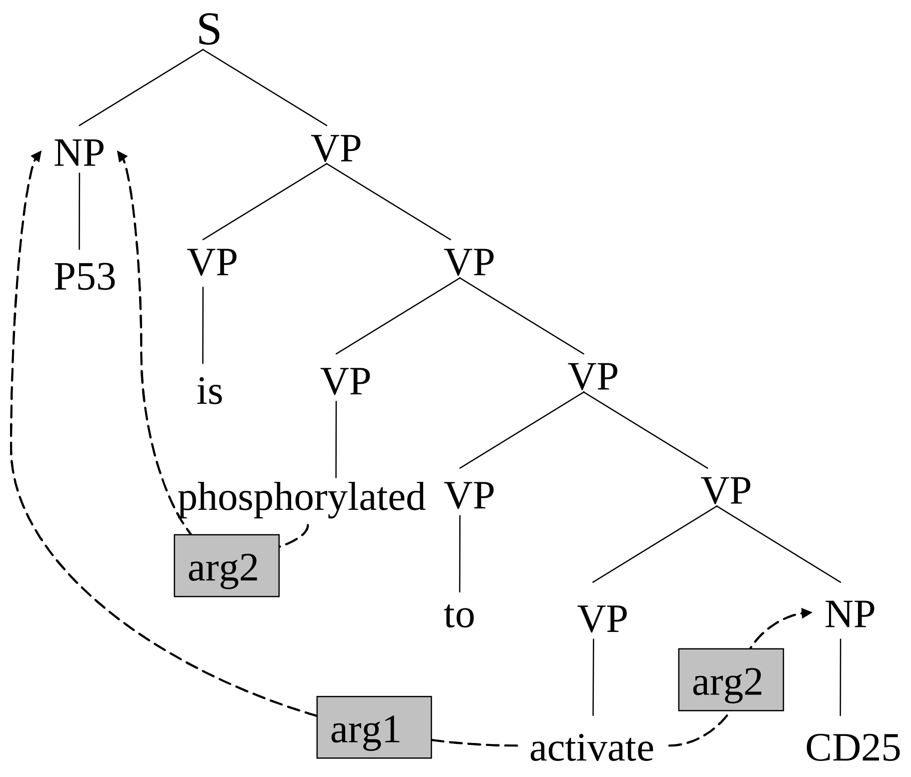


Figure 3.1: Syntactic/semantic structure

or “arg2” label. The three arrows in Figure 3.1 explain that “the subject of the verb ‘phosphorylated’ is ‘P53’,” “the subject of the verb ‘activate’ is ‘P53’ ” and “the object of the verb ‘activate’ is ‘CD25’.“ By using these parsing results for the sentences, we can search above three sentences by one query specifying the semantic structure “the subject of ‘activate’ is ‘p53’ and the object is ‘CD25’.”

Although the difference of expression can be absorbed by using the parsing results, the difference of words, such as synonyms, cannot be absorbed by paring. Consider the another example:

4. TP53 activates CD25

Although this sentence also contains the same fact as above three sentences, the word corresponding to “p53” in above three sentences is expressed with the synonymous word “TP53.” In order to recognize the fact that this sentence also contains the same piece of information with the above three sentences, we have to combine results from two kinds of NLP modules, a deep parser which outputs the subject-verb-object relations and a named entity recognizer or a dictionary, which recognized the words “p53” and “TP53” express the same object.

For more advanced retrieval using results of natural language processing like the above example, we will have to incorporate results of various types of natural language processing modules. In order to enable the incorporation of NLP results, we employed a method that expanding target documents with XML style annoraw text

p53 is phosphrylated to activate CD25.

⇓ annotating the text with parsing results

tag-annotated text

```
<sentence>
 <phrase id="0" cat="S" head="4" lex_head="6">
  <phrase id="1" cat="NP" head="2" lex_head="3">
   <phrase id="2" cat="NP" head="3" lex_head="3">
    <word id="3" pos="NN" cat="NP" base="p53">p53</word>
   </phrase>
  </phrase>
 <phrase id="4" cat="VP" head="5" lex_head="6">
  <phrase id="5" cat="VP" head="6" lex_head="6">
   <word id="6" pos="VBZ" cat="VP" base="is" arg1="1" arg2="7">
   is
   </word>
  </phrase>
  <phrase id="7" cat="VP" head="8" lex_head="9">
   <phrase id="8" cat="VP" head="9" lex_head="9">
    <word id="9" pos="VBN" cat="VP" base="phosphorylate"
          arg2="1" arg1="-1" arg3="10"
          rel_type="phosphorylation">
    phosphorylated
    </word>
   </phrase>
   <phrase id="10" cat="VP" head="11" lex_head="12">
    <phrase id="11" cat="VP" head="12" lex_head="12">
     <word id="12" pos="TO" cat="VP" arg1="1" arg2="13">
     to
     </word>
    </phrase>
    <phrase id="13" cat="VP" head="14" lex_head="15">
     <phrase id="14" cat="VP" head="15" lex_head="15">
      <word id="15" pos="VB" cat="VP" base="activate" arg1="1"
            arg2="16">
       activate
      </word>
     </phrase>
     <phrase id="16" cat="NP" head="17" lex_head="18">
      <phrase id="17" cat="NP" head="18" lex_head="18">
       <word id="18" pos="NN" cat="NP" base="cd25" arg1="1">
        CD25
       </word>
      </phrase>
     </phrase>
    </phrase>
   </phrase>
  </phrase>
 </phrase>
</sentence>
```

Figure 3.2: Tag annotation of Figure 3.1

tations, which express the results from NLP modules, and retrieve the expanded documents with the query specifying the annotations. Figure 3.2 represents an example of expanded documents with annotations of NLP results.

By annotating text with the PAS information and synonyms, we can search above four sentences with one query by specifying the annotated information. In the example annotations of parsing results in Figure 3.2, all phrases and words are assigned unique identifiers, and a relation between the phrases and words is expressed by using them. The relation " 'p53' is a subject of 'activate' " is expressed by the "arg1" attribute value in "word" tag containing the word "activate." The attribute value is '1,' which is the id number of 'phrase' tag containing the word 'p53.' 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.

Consider another example:

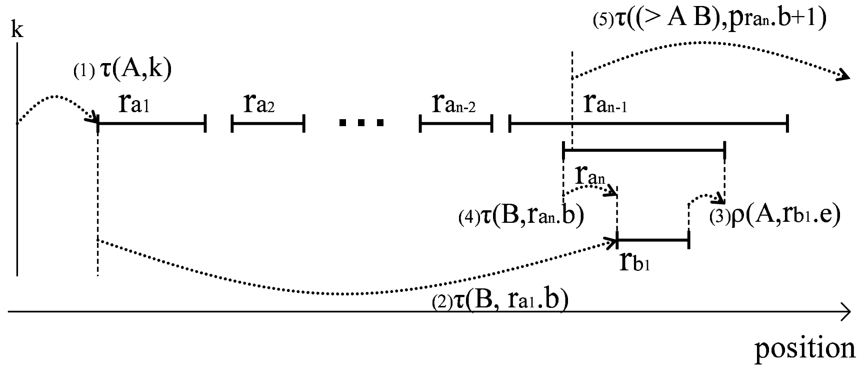


Figure 3.3: Evaluating the function '$\tau(> \text{A B},p)$'

5. The combination of p53 induction and IR cooperated
to activate CD25

To capture the relationship between 'P53,' and 'to activate', *proximity search based on constituent structures* must be used. That is, one has to express a query which says 'retrieve all sentences in which "P53" appears in the scope of a NP, which is linked in PAS with "to activate" as *arg1*, and in which "CD25" appears in the scope of another NP that fills in the role of *arg2* of the same "to activate" must be expressed.

Such a query can be formulated in terms of the relation between three tag-regions, the region or the scope of the NP in which "P53" appears, the region of the word, "to activate," and the tag-region in which the NP contains "CD25." In the following sections, we discuss how we have extended the original region algebra to construct an efficient and effective search system.

3.2 Retrieval of Nesting Annotations

The annotation of NLP results contains nesting structure in a set of annotations, such as the phrase structure in parsing results shown in Figure 3.2. In order to search over documents with XML-style annotations of NLP results, especially parsing result, the algorithm for region algebra must be able to search a set of nested-region, which means a set of regions in which the region contains another region. The original search algorithm for region algebra searches target regions in short time on the assumption that no nesting structure exists in a set of regions in search process, and the algorithm cannot be applied to nested regions. We extend the algorithm in order to search for a set of nested regions.

3.2.1 Operations for Nested Regions

Figure 3.3 depicts how the original algorithm for the 'containing' operation fails when applied to a set of nested tag regions. The cause of the failure is that, once it finds the first region of A (r_{a_n}) containing a region of B (r_{b_1}), it proceeds to the next A-region whose begin position is after the beginning position of r_{a_n}. This skipping is possible only when r_{a_n} is not contained in another larger region of A. In this example, it fails to recognize $r_{a_{n-1}}$ which properly contains r_{a_n} and r_{b_1}, but whose begin position is before r_{a_n}.

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 extended the algorithm to introduce the depth to handle nested regions efficiently.

Functions for region algebra with nested tag annotations

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

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 τ, ρ, τ', and ρ' with adding two arguments, a depth condition and a region condition as follows:
$\tau(A, p, d, r_r)$: Return the 'first' region of A, beginning at or after p, whose depth is d and contained in r_r.
$\rho(A, p, d, r_r)$: Return the 'first' region of A, ending at or after p, whose depth is d and contained in r_r.
$\tau'(A, p, d, r_r)$: Return the 'last' region of A, ending at or before p, whose depth is d and contained in r_r.
$\rho'(A, p, d, r_r)$: Return the 'last' region of A, beginning at or before p, whose depth is d and contained in r_r.
The order of the region used in the above definition is in the following:
τ, ρ': $r < r' \Leftrightarrow r.b < r'.b \vee (r.b = r'.b \wedge r.e > r'.e)$
ρ, τ': $r < r' \Leftrightarrow r.e < r'.e \vee (r.e = r'.e \wedge r.b > r'.b)$.
The first and second arguments, which represent the query expression and the begin position of search respectively, are the same with the corresponding functions in the original region algebra. The third argument is the depth which the output region must be, and the fourth argument is for the region restriction in which

```
Input: $p$: starting position, $d$: depth restriction, $r_r$: region restriction,
$(> AB)$: input query
Output: the first region satisfying the input conditions
Variables: $r_x$: region, $p_x$: position
$\tau((> AB), p, d, r_r) =$
  $r_{a_1} = \tau(A, p, d, r_r)$
  **if** $r_{a_1}$does not exist **then**
    **return** $(-,-)$
  $r_b = \rho(B, r_{a_1}.b, -, (r_{a_1}.b, r_r.e))$
  **if** $r_b$does not exist **then**
    **return** $(-,-)$
  **if** $r_{a_1}$ contains $r_b$ **then**
    **return** $r_{a_1}$
  **else**
    $r_{a_2} = \rho(A, r_b.e, d, (r_{a_1}.b, r_r.e))$
    $p_{next} = r_{a_2}.b$
    **if** $r_{a_2}$ contains $r_b$ **then**
      $r_{output} = r_{a_2}$
    **if** $d$ is not specified
      **for** $depth = r_{a_2}.depth - 1$ to 0
        $r_{a_3} = \rho'(A, r_{a_2}.b, depth, (r_{a_1}.b, r_r.e))$
        $p_{next} = r_{a_3}.b$
        **if** $r_{a_3}$ contains $r_b$ **then**
          $r_{output} = r_{a_3}$
  **if** $r_{output}$ exists **then**
    **return** $r_{output}$
  **else**
    **return** $\tau((> AB), p_{next}, d, r_r)$
```

Figure 3.4: The algorithm of the τ function for containing operator

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.

Algorithms for Nested Regions

We propose the extended algorithms which can search nested regions for operators of region algebra shown in Figure 3.1. Figure 3.4 and 3.5 show the algorithms for the two operators: containing and contained in, for the nested regions. The first step of the function $\tau((> AB), p, d, r_r)$ is similar to the original algorithm. The algorithm searches the first region r_{a_1} of A with τ function and the first ending region r_b of B from $r_{a_1}.b$. It outputs r_{a_1} if r_{a_1} contains r_b, since no region of A exists whose begin position is between p and $r_{a_1}.b$. When r_{a_1} does not contain r_b, the algorithm searches for the next candidate region r_{a_2}, which is the first ending region of A from $r_b.e$. The algorithm checks if r_{a_2} contains r_b, and stores r_{a_2} as a candidate of the output r_{output} if r_{a_2} contains r_b. Then, the algorithm searches

```
Input: $p$: starting position, $d$: depth restriction, $r_r$: region restriction,
$< AB$: input query
Output: the first region satisfying the input conditions
Variables: $r_x$: region, $p_x$: position
$\tau((< AB), p, d, r_r) =$
  $r_a = \tau(A, p, d, r_r)$
  **if** $r_a$ does not exist **then**
    **return** $(-, -)$
  $r_{b_1} = \rho(B, r_a.e, -, -)$
  **if** $r_{b_1}$ does not exist **then**
    **return** $(-, -)$
  $p_{next} = r_{b_1}.b$
  **if** $r_b$ contains $r_a$ **then**
    **return** $r_a$
  **for** $depth = r_b.depth - 1$ to 0
    $r_{b_2} = \rho'(B, r_b.b, depth, -)$
    **if** $r_{b_2}$ contains $r_a$**then**
      **return** $r_a$
    **if** $p_{next} < r_{b_2}.b$ **then**
      $p_{next} = r_{b_2}.b$
  $r_{b_3} = \rho(B, r_a.b, -, -)$
  **if** $r_{b_3}.e \ < \ r_a.e$ **then**
    $p_{next} = r_a.b + 1$
  **return** $\tau((< AB), p_{next}, d, r_r)$
```

Figure 3.5: The algorithm of the τ function for ‘contained in’ operator

Name(Expression)	Description
Containing ($> AB$)	‘A’ regions containing a ‘B’ region.
Contained In($< AB$)	‘A’ regions contained in a ’B’ region.
Followed By($-AB$)	Regions beginning with a ‘A’ region and ending with a ‘B’ region.
Both Of ($\&AB$)	Regions beginning with a ’A’ region and ending with a ’B’ region and the reverse.
One Of($\|AB$)	Regions of ’A’ regions or ’B’ regions.

Table 3.1: Operators of Region Algebra

a larger region of A (r_{a_3}), whose begin position is before $r_{a_2}.b$ and the nearest to r_{a_2}, with changing the depth restriction from the $r_{a_2}.depth - 1$ to 0, and stores r_{a_3} as r_{output} if r_{a_3} contains r_b. If r_{output} exists after the search of larger region r_{a_3} is finished, r_{output} is returned as an output. Otherwise, the algorithm searches the next region for ($> AB$) from the position p_{next}. In this case, no region r' for ($> AB$) such that $p < r'.b < p_{next}$ exists because no region r'_b of B such that $p < r'_b.e < r_b.e$ and no region of A contains r_b.

In this algorithm, we can skip some regions of A by using the depth restriction. Figure 3.6 shows an example of skipping. In this example, the algorithm searches

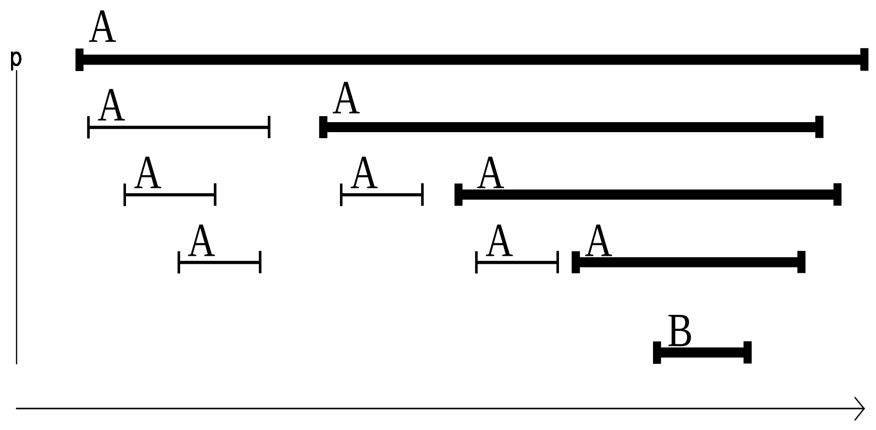


Figure 3.6: Skipping regions using depth

only regions of heavy line because the algorithm just has to search only one region in each depth. Because only one region can contain the region of B in the regions which have the same depth at most, the algorithm does not need to check other regions of A.

The first step of the algorithm for the function $\tau((< AB), p, d, r_r)$, which is show in Figure 3.5, is also similar for the original algorithm. The algorithm searches the first region r_a of A, and the first ending region r_{b_1} of B from r_a, and outputs r_a if r_a is contained in r_{b_1}. If r_a is not contained in r_{b_1}, it searches for a region r_{b_2} of B whose begin position is before and the nearest to $r_b.b$, with changing depth restriction and checks whether r_{b_2} contains r_a. When the region r_{b_2} that containing r_a exists, the algorithm outputs r_a. If no region of B contains r_a, the algorithm searches the next region for $(< AB)$. 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 $r'_b.b < p_{next}$, r'_a is contained in r_a or r'_b should be searched as r_b or r_{b_2} in the algorithm. Algorithms for the other functions is also constructed in a similar way.

We developed algorithms for the operators 'followed-by' (`-`) and 'both of' (`&`) to search innermost regions which satisfy the description in Table 3.1.

Because the above τ functions output the region whose beginning position is nearest to the search start 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 '`-`' or '`&`' contains other regions in the query.

3.3 Variables

3.3.1 Definition

In tag-annotation of sentential structure as in Figure 3.2, 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 $arg1$ in a word tag for the verb with the value of the attribute id in the phrase in Figure 3.2. When we search the relation, we need to express and confirm the equality of the two values. By using the variables, we can express the equality easily in queries such as:

```
(> [sentence]
   (& [word arg1=$x base="activate"]
      ([phrase id=$x] "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 $Q(x_1, ..., x_n)$, 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 $Q(x_1, ..., x_n)$ as

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

which means a set of the results retrieved by the query $Q(a_1, ..., a_n)$, which is the query $Q(x_1, ..., x_n)$ instantiated with an instantiation $\{a_1/x_1, ...a_n/x_n\}$.

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.

3.3.2 Algorithm

Main algorithm

The main algorithm to search all regions of $S(Q(x_1, ..., x_n))$ 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 $\{a_1/x_1, ...a_n/x_n\}$ 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 $S(Q(a_1, ..., a_n))$ 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.

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 $S(Q(x_1, ..., x_n))$ 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 Both Of(&) and One Of(|) with retaining the condition of words that a scope region should contain.

Generation of instantiations

We can compute $S(Q(x_1, ...x_n))$ by computing $S(Q(a_1, ..., a_n))$ for all possible instantiations of $\{a_1/x_1, ..., a_n/x_n\}$. 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, `[tag attr_1=$x_1,...,attr_m=$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 appears in at least one sub-query and calculating the variable value from the queries.

From a sub-query `[tag attr_1=$x_1,...,attr_m=$x_m]`, we can create an instantiation $\{a_1/x_1, ..., a_m/x_m\}$ by the following algorithm:
1. Create a query `[tag]` by removing attributes containing variable from the sub-query `[tag attr_1=$x_1,...,attr_m=$x_m]`.'
2. Calculate a region r from the query `[tag]` 3. Extract the value a_j for x_j from the corresponding text for r.

In the above, sub-queries are in the form of

```
[tag attr_1=$x_1,...,attr_m=$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. `[tag attr_1=$x_1,...,attr_m=$x_m]`
2. $(> Q_v\ Q)$
3. $(< Q_v\ Q)$

where Q_v is a value determining query, and Q is a query that contains no variable. 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.

Evaluation order of sub-queries

Consider the following query:

```
(> [sentence]
   (& [word arg1=$x1, arg2=$x2]
      (& (> ([phrase id=$x1]
             p53)
         (> [phrase id=$x2]
             cd25)))
```

This query retrieves all sentences in which p53 and cd25 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:

```
Input: $Q$: query
Output: the ordered list of query to calculate variable values
Variables: $SubQ$: sub-query, $X$: variable set, $Q_{min}$: query
function queryorder($Q$)
  $Qlist = ()$
  while no variable exist in $Q$
    $\{SubQ\}$ = value_determining_query($Q$)
    $Q_{min} = \text{argmin}_{SubQ_i \in \{SubQ\}}(efreq(SubQ_i))$
    $\{X\}$ = variables($Q_{min}$)
    add($Q_{min}$,$Qlist$) :add $Q_{min}$ at the end of $Qlist$
    $Q$ = assign_word($\{X\}$,$Q$,$efreq(Q_{min})$)
  return $Qlist$

value_determining_query($Q$): Output the sub-queries of $Q$ that satisfy
the condition of the value determining query.
assign_word($X$,$Q$,$f$): Output a query in which a word whose frequency
is $f$ is assigned temporarily into a variable of query $Q$ in the variable set
$X$.
```

Figure 3.7: The algorithm for the evaluation order

1. `[phrase id=$x1]`
2. `[phrase id=$x2]`
3. `[word arg1=$x1 arg2=$x2]`
4. `(> [phrase id=$x1] p53)`
5. `(> [phrase id=$x2] cd25)`

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.

```
Input: $Q$: query, $p$: start position of search
Output: the first region matched the query $Q$
Variables: $Q_s$: query, $Qlist$: list of queries, $r$: region
**function** retrieve($Q$, $p$)
  $Q_s$ = calcScopequery($Q$)
  $Qlist$ = queryorder($Q$)
  **while** $r = \tau(Q_s, p,\text{-},\text{-})$ exists
    $p = r.b + 1$
    determine($Q$,$Qlist$,$r$,$p$)

Input: $Q$: query, $Qlist$: query list, $r$: region, $p$: start position of search
Output: the first region matched the query $Q$
Variables: $Q_{first}, Q_l, Q'$: query, $p_{next}$: position, $r, r'$: region, $\{a_1/x_1, ..., a_n/x_n\}$: instantiation
**function** determine($Q$, $Qlist$, $r$, $p$)
  $p_{next} = r.b$
  $Q_{first} = Qlist.first$
  $Q'_{first}$ = remove_variable($Q_{first}$)
  **while** $r' = \tau(Q'_{first}, p_{next}, -, r)$ exists
    $p_{next} = r'.b + 1$
    $\{a_1/x_1, ..., a_n/x_n\}$ = calc_value($Q_{first}$,$r'$)
    $Q'$ = instantiate($Q$,$\{a_1/x_1, ..., a_n/x_n\}$)
    **if** variables do not exist in $Q'$
      **if** $r_{output} = \tau(Q',p,-,r)$ exists **then** output $r_{output}$
    **else**
      $Qlist'$ = remove_first($Qlist$)
      **foreach** $Q_l$ **in** $Qlist'$
        $Q_l$ = instantiate($Q_l$,$\{a_1/x_1, ..., a_n/x_n\}$)
      determine($Q'$,$Qlist'$,$r$, $p$)

remove_variable($Q$): Output a query $Q$ in which the attributes containing a variable are removed.
calc_value($Q$,$r$): Output a instantiation calculated in the region $r$.
instantiate($Q$,$\{a_1/x_1, ..., a_n/x_n\}$): Return a query instantiated with $\{a_1/x_1, ..., a_n/x_n\}$.
```

Figure 3.8: The algorithm for query with variables

However, since it takes time to determine the order for each instantiation, we estimate the frequency of the instantiated value for each variable as

$$efreq(SubQ_i) = \min_{w \in SubQ_i} freq(w)$$

, where $SubQ_i$ is a sub-query which is evaluated to determine the value for x and $freq(w)$ 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 3.7 shows the algorithm for determining the order of sub-queries.

The entire main algorithm is shown in Figure 3.8.

```
 1: <sentence>
 2: <phrase>, id=“0”, cat=“S”, head=“4”, lex_head=“6”
 3: <phrase>, id=“1”, cat=“NP”, head=“2”, lex_head=“3”
 4: <phrase>, id=“2”, cat=“NP”, head=“3”, lex_head=“3”
 5: <word>, id=“3”, pos=“NN”, cat=“NP”, base=“p53”
 6: p53
 7: </word>
 8: </phrase>
 9: </phrase>
10: <phrase>, id=“4”, cat=“VP”, head=“5”, lex_head=“6”
11: <phrase>, id=“5”, cat=“VP”, head=“6”, lex_head=“6”
12: <word>, id=“6”, pos=“VBZ”, cat=“VP”, base=“is”, arg1=“1”,
arg2=“7”
13: is
14: </word>
15: </phrase>
16: <phrase>, id=“7”, cat=“VP”, head=“8”, lex_head=“9”
17: <phrase>, id=“8”, cat=“VP”, head=“9”, lex_head=“9”
18: <word>, id=“9”, pos=“VBN”, cat=“VP”, base=“phosphorylate”,
    arg2=“1”, arg1=“-1”, arg3=“10”, rel_type=“phosphorylation”
19: phosphorylated
20: </word>
21: </phrase>
22: <phrase>, id=“10”, cat=“VP”, head=“11”, lex_head=“12”
23: <phrase>, id=“11”, cat=“VP”, head=“12”, lex_head=“12”
24: <word>, id=“12”, pos=“TO”, cat=“VP”, arg1=“1” arg2=“13”
25: to
26: </word>
27: </phrase>
28: <phrase>, id=“13”, cat=“VP”, head=“14”, lex_head=“15”
29: <phrase>, id=“14”, cat=“VP”, head=“15”, lex_head=“15”
30: <word>, id=“15”, pos=“VB”, cat=“VP”, base=“activate”,
arg1=“1”
    arg2=“16”
31: activate
32: </word>
...
```

Figure 3.9: Position number of words in Figure 3.2

3.4 Index Structure and Implementation

The algorithm we proposed requires mainly two types of data to search regions matching queries containing variables. One is a list of positions and depths in which words, tags and attributes appear, and the other is a set of attribute-values which appears in a position. The former data are used in execution of the functions τ, ρ, τ', ρ' to search regions, and the latter data are used in computation of a value for variables.

<sentence>	1-0, ...
<phrase>	2-0, 3-1, 4-2, 10-1,...
phrase:id="0"	2-0, ...
phrase:cat="S"	2-0, ...
phrase:head="4"	2-0, ...
phrase:lex_head="6"	2-0, 10-1, 11-2, ...
phrase:id="1"	3-1, ...
phrase:cat="NP"	3-1, 4-2, ...
...	...
<word>	4-0, ...
word:id="3"	4-0, ...
word:pos="NN"	4-0, ...
...	...
p53	6-0, ...
</word>	7-0, ...
</phrase>	8-2, 9-1, ...
...	...

Figure 3.10: Inverted position index

```
function makearray($T$)
  while $NodeQueue \neq \phi$
    $Node$ = $NodeQueue$.pop()
    for $i$ is 1 to # of position number in $Node$
      $PositionArray$.push($p_i$)
    for $j$ is 1 to # of child of $Node$
      $NodeQueue$.push($child_i$)
```

Figure 3.11: The algorithm to make array representation of B-tree

3.4.1 Depth and Position

A position number is attached for each appearance of words, tags and their attributes and the position list is constructed. In this implementation, we divide a tag into attributes and the tag without attributes to enable partial match of attributes. The position of the attributes is defined as the position of the tag which contains the attributes.

Figure 3.10 shows a list of tuple of a position and a depth. In the example, the depth of the tag region of 'phrase' $(4, 8)$(id="2") is 2 because regions $(3, 9)$(id="1") and $(2, 45)$ (id="0") contain it.

3.4.2 Implementation of Index Position Array

We implemented the list of positions for each word by use of an array representation of a B-tree structure, which the basic functions $(\tau, \rho, \tau', \rho')$ access in order to search a position of words. We can obtain the result for the functions in $O(logn)$ time, where n is the size of the list.

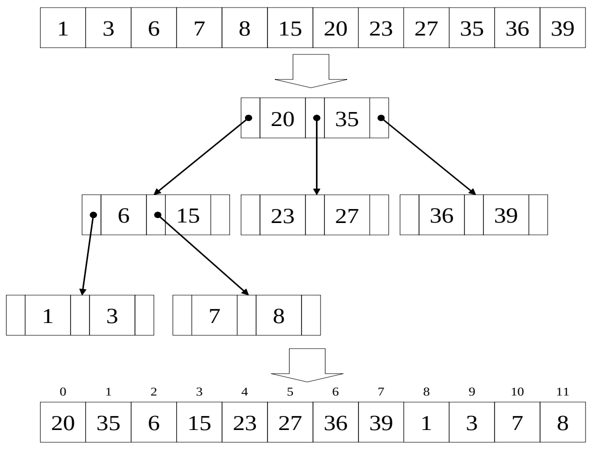


Figure 3.12: An example of B-tree structure and array representation

Array representation of B-tree structure

Figure 3.12 shows an example of the array representation of the B-tree structure. The B-tree structure we implemented have the following restriction.

- All nodes have n children except leaf nodes.
- All nodes have $n-1$ records, which are tuples of a position and a depth except the last node in breadth-first order.

By these restrictions, we can implement the B-tree structure in an array of records by placing records of each node in breadth-first order from the root node. The size of the array representation is only the size of records themselves. The list of the position is converted to the B-tree structure, and then the array representation of the B-tree structure is constructed by placing records in each node in breadth-first order from the root node. Figure 3.11 shows an algorithm to construct array representation for the B-tree structure. In the array representation, the move to a child node in the B-tree, is implemented by arithmetic operations of the suffix of the array. The formula to calculate a suffix number of the first element of the leftchild i_{c_l} and the right child i_{c_r} of the ith record is as follows:
$i_{c_l} = n \times i + n - r - 1$
$i_{c_r} = n \times i + n - r - 1 + n - 1$
where first suffix of the array representation is 0, i is a present suffix, n is a number of children each node has and r is remainder of $i \div (n-1)$.

Because the edges of the tree structure are expressed with relations of suffix number and the record data exist not only in leaf nodes but also in inner nodes, the size of the space that the array representation of tree structure requires is only the total size of record data.

In the case of Figure 3.12, since the root node contains the records 20 and 35, the algorithm first adds these two numbers to the position array. Next, it evaluates the child nodes of the root node and adds $6, 15, 23, 27, 36, 39$ to the position array. Finally, the algorithm evaluate the rest of the nodes and adds $1, 3, 7, 8$ to the array. Then the position array is $\{20, 35, 6, 15, 23, 27, 36, 39, 1, 3, 7, 8\}$ as shown in Figure 3.12.

Cache effect

When we operate a search system with this implementation, the retrieval speed will be improved by cache effect. The root node of the position list tree is, by necessity, accessed in each position search. When one user performs some query, and the system searches the regions, the data which are accessed while the search are loaded into a memory from the database. Further, another query containing the same words or tags from the previous query was performed; since the data in the root node corresponding to the words or tags in the previous query already exist in a memory, the retrieval time decreases. When the system is actually operated, the cache will have an effect to some degree, since the position list of tags will be accessed to an evaluation of many queries. In the following experiments, we show the performances of our system with cache and without cache. The one with cache is the one when a query is issued after the same query is processed. The one without cache means the one in which the system does not use cache at all. The performance in the actual environment, where different queries are issued in succession, would be between these two extremes.

3.5 Algorithm for Searching Stand-Off Annotations

```
<sentence>
 <phrase id="0" cat="S" head="4" lex_head="6">
  <phrase id="1" cat="NP" head="2" lex_head="3">
   <phrase id="2" cat="NP" head="3" lex_head="3">
    <word id="3" pos="NN" cat="NP" base="p53">p53</word>
   </phrase>
  </phrase>
 <phrase id="4" cat="VP" head="5" lex_head="6">
  <phrase id="5" cat="VP" head="6" lex_head="6">
   <word id="6" pos="VBZ" cat="VP" base="is" arg1="1" arg2="7">
   is
   </word>
  </phrase>
  <phrase id="7" cat="VP" head="8" lex_head="9">
   <phrase id="8" cat="VP" head="9" lex_head="9">
    <word id="9" pos="VBN" cat="VP" base="phosphorylate"
          arg2="1" arg1="-1" arg3="10"
          rel_type="phosphorylation">
    phosphorylated
    </word>
   </phrase>
   <phrase id="10" cat="VP" head="11" lex_head="12">
    <phrase id="11" cat="VP" head="12" lex_head="12">
     <word id="12" pos="TO" cat="VP" arg1="1" arg2="13">
     to
     </word>
    </phrase>
    <phrase id="13" cat="VP" head="14" lex_head="15">
     <phrase id="14" cat="VP" head="15" lex_head="15">
      <word id="15" pos="VB" cat="VP" base="activate" arg1="1"
           arg2="16">
       activate
      </word>
     </phrase>
     <phrase id="16" cat="NP" head="17" lex_head="18">
      <phrase id="17" cat="NP" head="18" lex_head="18">
       <word id="18" pos="NN" cat="NP" base="cd25" arg1="1">
        CD25
       </word>
      </phrase>
     </phrase>
    </phrase>
   </phrase>
  </phrase>
 </phrase>
</sentence>
```

Figure 3.13: Example of inline XML annotations

```
01234567890123456789012345678901234567 8
P53 is phosphorylated to acrivate CD25

 0 38 sentence
 0 38 phrase id="0" cat="S" head="4" lex_head="6"
 0  3 phrase id="1" cat="NP" head="2" lex_head="3"
 0  3 phrase id="2" cat="NP" head="3" lex_head="3"
 0  3 word id="3" pos="NN" cat="NP" base="p53"
 4 38 phrase id="4" cat="VP" head="5" lex_head="6"
 4  6 phrase id="5" cat="VP" head="6" lex_head="6"
 4  6 word id="6" pos="VBZ" cat="VP" base="is" arg1="1"
           arg2="7"
 7 38 phrase id="7" cat="VP" head="8" lex_head="9"
 7 21 phrase id="8" cat="VP" head="9" lex_head="9"
 7 21 word id="9" pos="VBN" cat="VP" base="phosphorylate"
           arg2="1" arg1="-1" arg3="10"
           rel_type="phosphorylation"
22 38 phrase id="10" cat="VP" head="11" lex_head="12"
22 24 phrase id="11" cat="VP" head="12" lex_head="12"
22 24 word id="12" pos="TO" cat="VP" arg1="1" arg2="13"
25 38 phrase id="13" cat="VP" head="14" lex_head="15"
25 33 phrase id="14" cat="VP" head="15" lex_head="15"
25 33 word id="15" pos="VB" cat="VP" base="activate"
           arg1="1" arg2="16"
34 38 phrase id="16" cat="NP" head="17" lex_head="18"
34 38 phrase id="17" cat="NP" head="18" lex_head="18"
34 38 word id="18" pos="NN" cat="NP" base="cd25" arg1="1"
```

Figure 3.14: Example of stand-off annotations for XML data in Figure 3.13

3.5.1 Stand-Off Annotations

The stand-off annotations is a kind of XML-style annotations. In a usual XML-style annotations, which is called *inline XML* or *inline annotation*, the XML-style tags of annotations are embedded into the target text. In contrast, in the stand-off annotations, the annotations are separated from the target text. Figure 3.14 shows an example of stand-off annotations for the example of inline XML annotations in Figure 3.13. Each line of stand-off annotations corresponds to a tag-annotation of inline XML. The first and second column expresses a begin position and end position of tag-annotated text region respectively. The third column is a tag name, and the fourth column expressed attribute values. The stand-off annotations can express all inline XML data, and also express annotations which intersect with each other.

The stand-off annotations have substantial advantage over in-line annotations in points of annotations and indexing for search systems. By using the stand-off annotations, the management of the annotations, such as addition and deletion of annotations, becomes easier than using in-line annotations because the target text and annotations are separated in stand-off annotations. In the search indices in region algebra, which consist of inverted indices, that is, list of positions in which words or tags appear, reconstruction of whole indices will be needed when new annotation is added in in-line annotations, since addition of new annotation will change the position of words in the target text and tags in existing other annotations. Since reconstruction of whole indices require very long time for a huge document collection, the in-line annotations cannot be applied to a huge document collection when addition of various types of annotations are supposed. By stand-off annotations, in which offset position in the target text is used as a position in search, when the new annotations are added to or deleted from the index databases, the system creates indices for only newly added annotations, and does not have to change the indices for the target text and other existing annotations. The target text, existing annotations and newly added annotations are incorporated in search phase.

3.5.2 Search Algorithm for Stand-Off Annotation

Although the stand-off style annotations makes it easier to manage the annotations, the algorithm cannot be applied to the stand-off annotations without change because a number of annotations will be placed in the same offset. Since the original algorithm uses positions of regions to search annotation regions, the algorithm cannot be applied to the stand-off annotations. In the improved algorithm, we extend the algorithm by incorporating "depth" information in order to apply the algorithm for a set of nested annotations. This improvement seems to be able to be applied to stand-off annotations. But since the "depth" information should be unique in a set of a type of annotations, the reconstruction of "depth" information will be needed when a new annotations are added to a search database.

We improve the algorithm in order to resolve this problem for stand-off annotations. First, we introduce an Iterator in order to store the search status, such as the region searched previously, the search condition and the iterator of sub-queries. For example, for a basic word query, such as "p53," the iterator for the query contains:

- search condition (search start position, depth and region limitation)
- the region previously retrieved,

and for a query `(> A B)`, the iterator for the query contains:

- search condition (search start position, depth and region limitation)
- the iterator for the query `A`
- the iterator for the query `B`
- the region previously retrieved.

Next, we extend the access functions τ, ρ, τ', ρ' to use the iterator. For example for the function τ, we introduce two types of function, τ_b and τ_n. The function τ_b searches a region from the position condition, and the function τ_n searches a next region of the region search previously in the same condition. The algorithm first searches a region for a query using τ_b functions with the begin position for search and a condition for search, the depth and region condition. When the algorithm searches the next region after the first search in the same condition, the function τ_n is called without any arguments because the search start position and other search condition is already stored in the iterator. For example, when the query is "[phrase]" and the list of position in which the tag "[phrase]" appears is {(0,38), (0,3), (4,38)}, the algorithm searches the first region by the function $itr_{[phrase]}.\tau_b(0, -, -)$, which means "search the first region of `[phrase]` whose begin position is after the position 0," and the result regions is (0,38). In the same time, this region is stored in the iterator $itr_{[phrase]}$ as a region previously searched, and the search conditions, although there is no restriction of depth and regions, are stored in the iterator. Then the algorithm searched the next region of (0,38) by the function $itr_{[phrase]}.\tau_n()$. In this case, the previous search result (0,38) is stored in the iterator, the algorithm searches "the next region of (0,38)" from the list of positions, and output the region (0,3). In the original algorithm, since the algorithm uses only the position of the regions and "the next region" means "the region begin/end at the next position," this region (0,3) is skipped in the algorithm.

Figure 3.15 shows an algorithm of τ_b and τ_n functions, and Figure 3.16 shows an algorithm of ρ_b and ρ_n functions for containing operator. The difference

Input: p: start position, d: depth restriction, r_r: region restriction
Output: the first region of $(>\ A\ B)$ satisfying the input conditions

```
$itr_{(>AB)}.\tau_b(p, d, r_r) =$
   $itr_{(>\ A\ B)}.itr_L.\tau_b(p, d, r_r)$
   **return**$\tau_s()$
```

Output: the next region of $(>\ A\ B)$ which is stored in $itr_{(>\ A\ B)}$ satisfying the input conditions

```
$itr_{(>\ A\ B)}.\tau_n() =$
   $itr_{(>\ A\ B)}.itr_L.\tau_n()$
   **return**$\tau_s()$

$itr_{(>\ A\ B)}.\tau_s() =$
   $r_A = itr_{(>\ A\ B)}.itr_L.r$
   **while** $r_A$ exists
      $r_B = itr_{(>\ A\ B)}.itr_R.\rho_b(r_A.b, -, (r_A.b, -))$
      **if** $r_B$ doesnotexists **then**
         **return**$(-, -)$
      **if** $r_A$ contains $r_B$ **then**
         **if** $r_A! = itr_{(>AB)}.itr_L.r$ **then**
         **return** $r_A$
      $r_{A2} = itr_{(>\ A\ B)}.itr_L.\rho_b(r_B.e, d, (r_A.b + 1, -))$
      **if** $\mathrm{r_{A2}}$does not$\exists$ **then**
         **return** $(-, -)$
      **if** $r_{A2}$ contains $r_B$ **then**
         $r_o = r_{A2}$
      **else**
         $r_n = r_{A2}$
      $r_{A3} = itr_{(>\ A\ B)}.itr_L.\rho'_b(r_{A2}.b, d, (r_A.b + 1, -))$
      $itr_{(>\ A\ B)}.\tau_{loop}(r_{A3}, r_b, r_o, r_n)$
      **if** $r_o isvalid$ **then**
         $itr_{(>\ A\ B)}.r = itr_{(>\ A\ B)}.itr_L.r$
         **return**$itr_{(>\ A\ B)}.itr_L.r$
      $r_A = r_n$
   $itr_{(>\ A\ B)}.r =$ invalid
   **return**$itr_{(>\ A\ B)}.r$

$itr_{(>\ A\ B)}.\tau_{loop}(r_A, r_B, r_o, r_n) =$
   **while** $r_A$ exists
      $itr_{(>\ A\ B)}.d = r_A.d - 1$
      $r_A = itr_{(>\ A\ B)}.itr_L.\rho'_n()$
      **if** $r_A$ exists **then**
         **if** $r_A$ contains $r_B$ **then**
            $r_o = r_A$
         **else**
            $r_n = r_A$
```

Figure 3.15: τ_b and τ_n function for query (> A B).

```
Input: $p$: start position, $d$: depth restriction, $r_r$: region restriction
Output: the first region of $(>\ A\ B)$ satisfying the input conditions
$itr_{(>\ A\ B)}.\rho_b(p, d, r_r) =$
  $itr_{(>\ A\ B)}.itr_L.\rho_b(p, d, r_r)$
  return $\rho_s()$

Output: the next region of $(>\ A\ B)$ which is stored in $itr_{(>\ A\ B)}$ satis-
fying the input conditions
$itr_{(>\ A\ B)}.\rho_n() =$
  $itr_{(>\ A\ B)}.itr_L.\rho_n()$
  return $\rho_s()$

$itr_{(>\ A\ B)}.\rho_s() =$
  $r_A = itr_{(>\ A\ B)}.itr_L.r$
  while $r_A$ exists
    $r_B = itr_{(>\ A\ B)}.itr_R.\rho_b(r_A.b, -, (r_A.b, -))$
    if $r_B$ does not exist then
      return $(-, -)$
    if $r_A$ contains $r_B$ then
      return $r_A$
    $itr_{(>\ A\ B)}.itr_L.r_r = (-, r_B.e)$
    $r_o = itr_{(>\ A\ B)}.\rho_{loop}(r_A, r_b, d, r_r)$
    if $r_o$ exists then
      return $r_o$
    $r_A = itr_{(>\ A\ B)}.itr_L.\rho_b(r_B.e, d, r_r)$


$itr_{(>\ A\ B)}.\rho_{loop}(r_A, r_B, d, r_r) =$
  while $r_A$ exists
    $itr_{(>\ A\ B)}.d = r_A.d - 1$
    $r_A = itr_{(>\ A\ B)}.itr_L.\rho_n()$
    if $r_A$ exists then
      if $r_B$ contains $r_A$ then
        return $r_A$
      $r_{B2} = itr_{(>\ A\ B)}.itr_R.\tau_b(r_A.b, -, r_A.b, r_A.e)$
      if $r_{B2}$ exists then
        return $r_A$
```

Figure 3.16: ρ_b and ρ_n function for query (> A B).

between the functions τ_b and τ_n for containing operator is the first step of search. For τ_b function, the algorithm searches the first region of A based on the position in the first step. On the other hand, the algorithm of τ_n function searches the first region of A by searching the next region of the region searched in previous search process. By using the τ_n function, the algorithm can search all regions of `(> A B)` which are indistinguishable by only the position information, that is, the regions start in the same position. Search algorithm for τ_b and τ_n functions after the first step, which searches the first region of $A(r_A)$, is as follows: The algorithm searches the first B region (r_B) by ρ_b function from the begin position of r_A, which searches the region whose end position is nearest to the the begin position of r_A. When the region r_A contains the region r_B, the algorithm output r_A region as the result region, since the is no region of A whose begin position is nearest to the search start position. When the region r_A does not contains the region r_B, the algorithm searches another region of A, which contains the region r_B. The algorithm searches the first region of A (r_{A2}) by ρ_b function from the end position of r_B, which searches the the region whose end position is nearest. In this search, the algorithm adds region condition that the begin position of result region is after the begin position of r_A, since there is no region of A whose begin position is before the begin position of r_A and contains the region r_B. When the region r_{A2} contains r_B, the region r_{A2} is a candidate of the output of this function. However, another region of A whose begin position is located between the begin position of r_A and r_{A2} can contains r_B, the algorithm searches another region of A with decreasing the depth condition. The τ_{loop} function in Figure 3.15 searches such regions of A by using ρ'_n function, with checking whether the searched region contains the region r_B. After the search in τ_{loop} is finished, the latest region which contains r_B and searched in τ_{loop} functions is output. When no such region exist, the algorithm searches the next region from the begin position of the last region searched in the τ_{loop} function. The algorithm for ρ_b and ρ_n function is similar to that for τ_b and tau_n functions. After the first step, which searched the first region of A (r_A), the algorithm searches the first region of B (r_B) by the ρ_b function from the begin position of r_A, which searches the region whose begin position is nearest. The algorithm output the region r_A when the searched region r_B is contained in r_A. When the region r_B does not contained in the region r_A, the algorithm searches another region of A by ρ_{loop} function in Figure 3.16, which searches regions of A with decreasing the depth condition. In the ρ_{loop} function, the algorithm checks the searched region of A contains the region r_B and also checks the region contains another region of B (r_{B2}). After the search in τ_{loop} is finished, the latest region which contains r_B and searched in ρ_{loop} functions is output. When no such region exists, the algorithm searches the next region from the begin position of the region of A that is searched by ρ_b function from the end position of r_B.

Figure 3.17 and Figure 3.18 shows an algorithm of functions for "contained in" operator. This algorithm is similar to the "containing" operator in that the difference of algorithm between τ_b and τ_n, and difference between ρ_b and ρ_n is the first step in the algorithm. The algorithm after the first step, which searches the first region of A (r_A), is as follows: The algorithm for τ_b and τ_n functions first searches the region of B by ρ_b function from the end position of r_A, which searched the first region of B (r_B) whose end position is nearest after the end position of r_A. When the region r_B contains the region r_A, the algorithm output

```
Input: $p$: start position, $d$: depth restriction, $r_r$: region restriction
Output: the first region of $(<\ A\ B)$ satisfying the input conditions
$itr_{(<\ A\ B)}.\tau_b(p, d, r_r) =$
  $itr_{(<\ A\ B)}.itr_L.\tau_b(p, d, r_r)$
  **return** $\tau_s()$

Output: the next region of $(<\ A\ B)$ which is stored in $itr_{(<\ A\ B)}$ satis-
fying the input conditions
$itr_{(>AB)}.\tau_n() =$
  $itr_{(<\ A\ B)}.itr_L.\tau_n()$
  **return** $\tau_s()$

$itr_{(<\ A\ B)}.\tau_s() =$
  $r_A = itr_{(<\ A\ B)}.itr_L.r$
  **while** $r_A$ exists
    $r_B = itr_{(>AB)}.itr_R.\rho_b(r_A.e, -, (-, -))$
    **if** $r_B$ does not exist **then**
      **return** $itr_{(<\ A\ B)}.\tau_n()$
    **if** $r_B$ contains $r_A$ **then**
      **return** $r_A$
    $p_{next} = r_B.b$
    **while**$r_B$ exists
      $itr_{(<\ A\ B)}.itr_R.d = r_B.d - 1$
      $r_B = itr_{(<\ A\ B)}.itr_R.\rho_n()$
      **if** $r_B$ exists
        **if** $r_B$ contains $r_A$
          **return** $r_A$
        **if** $r_B.b < p_{next}$
          $p_{next} = r_B.b$
    $r_{B2} = itr_{(<\ A\ B)}.itr_R.\rho_b(r_A.b, -, (-, -))$
    **if** $r_{B2}.e > r_A.e$
      $r_A = itr_{(<\ A\ B)}.itr_L.\tau_b(p_{next}, itr_{(<\ A\ B)}.d, itr_{(<\ A\ B)}.r_r)$
    **else**
      $r_A = itr_{(<\ A\ B)}.itr_L.\tau_n()$
```

Figure 3.17: τ_b and τ_n function for query (< A B).

the r_A as the region of the function since there is no region of A whose end position is before the begin position of r_A. When the region r_B does not contain the region r_A, the algorithm searched regions of B which contains the region r_A by using ρ_n functions, which searched the next region of B, with decreasing the depth condition. When at least one region of B searched in the loop contains the the region r_A, the algorithm output the region r_A as the result of the function. When there is no region of B contains r_A, the algorithm searches next output candidate region of A. In order to skip region of A which never contained in a region of B, the algorithm searches the region of B (r_{B2}) by ρ_b function from the begin position of r_A. When the end position of the region r_{B2} is after the end

```
Input: $p$: start position, $d$: depth restriction, $r_r$: region restriction
Output: the first region of $(<\ A\ B)$ satisfying the input conditions
$itr_{(<\ A\ B)}.\rho_b(p, d, r_r) =$
  $itr_{(<\ A\ B)}.itr_L.\rho_b(p, d, r_r)$
  return $\rho_s()$

Output: the next region of $(<\ A\ B)$ which is stored in $itr_{(<\ A\ B)}$ satis-
fying the input conditions
$itr_{(<\ A\ B)}.\rho_n() =$
  $itr_{(<\ A\ B)}.itr_L.\rho_n()$
  return $\rho_s()$

$itr_{(<\ A\ B)}.\rho_s() =$
  $r_A = itr_{(<\ A\ B)}.itr_L.r$
  while $r_A$ exists
    $r_B = itr_{(<\ A\ B)}.itr_R.\rho_b(r_A.e, -, (-, -))$
    if $r_B$ does not exist then
      return $(-, -)$
    if $r_B$ contains $r_A$ then
      return $r_A$
    $p_{next} = r_B.b$
    while $r_B$ exists
      $itr_{(>AB)}.itr_R.d = r_B.d - 1$
      $r_B = itr_{(>AB)}.\rho_n()$
      if $r_B$ is valid then
        if $r_B$ contains $r_A$ then
          return$r_A$
        if $r_B.b < p_{next}$ then
          $p_{next} = r_B.b$
    $r_A = itr_{(<\ A\ B)}.itr_L.\rho_b(max(r_A.e + 1, p_{next}),$
    $itr_{(<\ A\ B)}.d, itr_{(<\ A\ B)}.r_r)$
```

Figure 3.18: ρ_b and ρ_n function for query (< A B).

position of r_A, there is no region of B whose begin position is before the begin position of r_B since the region r_A is not contained in any region of B. In that case, the algorithm searches the next region of A from the smallest begin position of regions of B which are used to check whether the region r_A is contained in r_B. Otherwise, the algorithm searches the next region by using the τ_n function, because the regions of B whose begin position is before the begin position of r_A will contains the region. The algorithm for ρ_b and ρ_n functions is similar to that of τ_b and τ_n. After the first step which searches the first region of A (r_A), the algorithm searches the region of B (r_B) by the ρ_b function from the end position of r_A. When the region r_B contains the region r_A, the algorithm outputs r_A as the result of the function. When the region r_B does not contains the region r_A, the algorithm searched regions of B which contains the region r_A by using ρ_n

functions with decreasing the depth condition as the same with the algorithm for τ_b and τ_n functions. When at least one region of B contains the the region r_A, the algorithm output the region r_A. When there is no region of B contains r_A, the algorithm searches next output candidate region of A by using ρ_b function from the larger position in the next position of the end position of r_A and the smallest begin position of regions of B which are used to check whether the region r_A is contained in r_B. In the case of these ρ_b and ρ_n functions, the algorithm can skip regions of A which is not contained in any region of B by using the region of B which is searched in order to check whether the region r_A is contained in a region of B, that is, the next candidate region of A is searched from the smallest begin position in such regions of B (p_{next}). However, although the end position of the next candidate region of A should be placed after the end position of r_A, the begin position p_{next} can placed before the end position of r_A. In order to avoid the loop that the algorithm searched r_A as the next candidate region again, the candidate region is searched from the larger position of p_{next} and the next position of the end position of r_A.

Figure 3.19 and 3.20 show an algorithm of functions for the "one of" operator. The algorithm of τ_b and ρ_b first searches a region for each query of argument by using τ_b and ρ_b respectively. The the algorithm selects a region from two regions whose begin position is closer to the search start position in τ_b function, and whose end position is closer to the search start position in ρ_b function. On the other hand, the algorithm of τ_n and ρ_n searches the next region of an argument query from the argument from which the previous search result is derived, and the region stored in the iterator is selected as a candidate region from the argument from which the previous search result is not derived. Then the algorithm select a region from the two regions as the same with the algorithm in τ_b and ρ_b.

Figure 3.21, 3.22 and 3.23 show algorithm of access functions for the "both of" operator. Since the definition of a result region set for the "both of" operator is the inner-most regions in a set, that is, the result regions does not contain other regions, there is no region pair that the begin positions are the same. So the search algorithm for τ_n and ρ_n functions is the same with the algorithm for τ_b and ρ_b, and the search start position is the next position of the begin position of previously retrieved region. The algorithm for τ_b and τ_e functions first searches the first regions for A and B (r_A and r_B) by using the ρ_b function from the input search start position. The following algorithm branches by the positional relations between the end positions of regions r_A and r_B. When the end position of the region r_A is placed before that of r_B, the algorithm searches a new region of A (r_{A1}) from the end position of r_B by using ρ'_b function, which searches the region whose end position is nearest. The algorithm searches the region r_{A1} in order to construct the inner-most region by search the region of A nearest to the region r_b and construct a region whose begin position is the smaller position

```
Input: $p$: start position, $d$: depth restriction, $r_r$: region restriction
Output: the first region of $(|\ A\ B)$ satisfying the input conditions
$itr_{(|\ A\ B)}.\tau_b(p, d, r_r) =$
  $r_A = itr_{(|\ A\ B)}.itr_L.\tau_b(p, d, r_r)$
  $r_B = itr_{(|\ A\ B)}.itr_R.\tau_b(p, d, r_r)$
  **return** $itr_{(|\ A\ B)}.\tau_s(r_A, r_B)$

Output: the next region of $(|\ A\ B)$ which is stored in $itr_{(|\ A\ B)}$ satisfying
the conditions
$itr_{(|\ A\ B)}.\tau_n() =$
  **if** $itr_{(|\ A\ B)}.r == itr_{(|\ A\ B)}.itr_L.r$ **then**
    $r_A = itr_{(|\ A\ B)}.itr_L.\tau_n()$
  **else**
    $r_A = itr_{(|\ A\ B)}.itr_L.r$
  **if** $itr_{(|\ A\ B)}.r == itr_{(|\ A\ B)}.itr_R.r$ **then**
    $r_B = itr_{(|\ A\ B)}.itr_R.\tau_n()$
  **else**
    $r_B = itr_{(|\ A\ B)}.itr_R.r$
  **return** $itr_{(|\ A\ B)}.\tau_s(r_A, r_B)$

$itr_{(|\ A\ B)}.\tau_s(r_A, r_B) =$
  **if** $r_A$ does not exist **then**
    **return** $r_B$
  **if** $r_B$ dose not exist **then**
    **return** $r_A$
  **if** $r_A.b < r_B.b$ or $r_A.b == r_b.b$ and $r_A.e \geq r_B.e$ **then**
    **return** $r_A$
  **else**
    **return** $r_B$
```

Figure 3.19: τ_b and τ_n function for query (| A B).

in the begin position of r_{A1} and that of r_B as the start position and the end position is the end position of the region r_B. When the region r_A is the region of A nearest to the region r_b, r_{A1} is equal to r_A. In the contrary, when the end position of r_B is placed before that of r_A, the algorithm searches another region of B and construct a new region as an output in similar way of the previous case. In other cases, which mean that the end position of two regions, r_A and r_B, is placed in the same position, the algorithm output the region which contains the other region as an output.

The algorithm for ρ_b and ρ_e functions also searches the first regions for A and B (r_A and r_B) by using the ρ_b function, and the following algorithm branches by the positional relations between the regions r_A and r_B. When the end position of r_A and that of r_B are the same position, the calculation of the output region branches by the relation of the begin position of the regions r_A and r_B and the search start position. The begin position of the output region is basically the

```
Input: $p$: start position, $d$: depth restriction, $r_r$: region restriction
Output: the first region of $(|\ A\ B)$ satisfying the input conditions
$itr_{(|\ A\ B)}.\rho_b(p, d, r_r) =$
  $r_A = itr_{(|\ A\ B)}.itr_L.\rho_b(p, d, r_r)$
  $r_B = itr_{(|\ A\ B)}.itr_R.\rho_b(p, d, r_r)$
  **return**$itr_{(|\ A\ B)}.\rho_s(r_A, r_B)$

Output: the next region of $(|\ A\ B)$ which is stored in $itr_{(|\ A\ B)}$ satisfying
the conditions
$itr_{(|\ A\ B)}.\rho_n() =$
  **if** $itr_{(|\ A\ B)}.r == itr_{(|\ A\ B)}.itr_L.r$ **then**
    $r_A = itr_{(|\ A\ B)}.itr_L.\rho_n()$
  **else**
    $r_A = itr_{(|\ A\ B)}.itr_L.r$
  **if** $itr_{(|\ A\ B)}.r == itr_{(|\ A\ B)}.itr_R.r$ **then**
    $r_B = itr_{(|\ A\ B)}.itr_R.\rho_n()$
  **else**
    $r_B = itr_{(|\ A\ B)}.itr_R.r$
  **return**$itr_{(|\ A\ B)}.\rho_s(r_A, r_B)$

$itr_{(|\ A\ B)}.\rho_s(r_A, r_B) =$
  **if** $r_A$ does not exist **then**
    **return** $r_B$
  **if** $r_B$ does not exist **then**
    **return** $r_A$
  **if** $r_A.e < r_B.e$ or $r_A.e == r_b.e$ and $r_A.b \geq r_B.b$ **then**
    **return** $r_A$
  **else**
    **return** $r_B$
```

Figure 3.20: ρ_b and ρ_n function for query (| A B).

smaller position in both of the begin positions when both the begin position of r_A and that of r_B is placed after the search start position. However, when the begin position of r_A is placed before the search start position, the regions of A nearest to the region r_B can exists before the search start position. So the algorithm searches the first region of A from the search start position by ρ'_b functions, and construct an output region from the nearest regions. When the begin position of r_B is placed before the search start position, the algorithm searches a new region of B and construct an output region.

In the case the end position of r_A is larger than that of r_B, the algorithm searches the region of A (r_{A2}) nearest to the region r_B, and then searches the region of B (r_{B2}) nearest to the region r_{A2} and output a new region constructed with r_{A2} and r_{B2}. In the opposite case, which is the case that the end position of r_B is larger than that of r_A, the algorithm also construct a new region with the nearest two regions in similar way in the previous case.

```
Input: $p$: start position, $d$: depth restriction, $r_r$: region restriction
Output: the first region of (& $A$ $B$) satisfying the input conditions
$itr_{(\&\ A\ B)}.\tau_b(p, d, r_r) =$
  $r_A = itr_{(\&\ A\ B)}.itr_L.\rho_b(p, -, (p, r_r.e))$
  **if**$r_A$ does not exist **then**
    **return** $(-, -)$
  $r_B = itr_{(\&\ A\ B)}.itr_R.\rho_b(p, -, (p, r_r.e))$
  **if**$r_B$ does not exist**then**
    **return** $(-, -)$
  **if** $r_A.e < r_B.e$ **then**
    $r_{A1} = itr_{(\&\ A\ B)}.itr_L.\rho'_b(r_B.e, -, (p, r_B.e))$
    **return** $(min(r_{A1}.b, r_B.b), r_B.e)$
  **elsif** $r_A.e > r_B.e$ **then**
    $r_{B1} = itr_{(\&\ A\ B)}.itr_R.\rho'_b(r_A.e, -, (p, r_A.e))$
    **return** $(min(r_{B1}.b, r_A.b), r_A.e)$
  **else**
    **return** $(min(r_A.b, r_B.b), r_A.e)$
```

```
Output: the next region of (& $A$ $B$) which is stored in $itr_{(\&\ A\ B)}$ satisfying
the input conditions
$itr_{(\&\ A\ B)}.\tau_n() =$
  **return** $itr_{(\&\ A\ B)}.\tau_b(itr_{(\&\ A\ B)}.r.b + 1, itr_{(\&\ A\ B)}.d, itr_{(\&\ A\ B)}.r_r)$
```

Figure 3.21: τ_b and τ_n function for query (& A B).

The algorithm for access functions for the "followed by" operator, which is shown in Figure 3.24 and 3.25, is similar to the algorithm for the "both of" operator. The definition of a result region set for the "followed by" operator is also a set of inner-most regions like "both of" operator. The algorithm for τ_n and ρ_n functions is search the next region from the next position of the previous result by τ_b and ρ_b functions. The algorithm for τ_b and τ_n functions first searches the region of A (r_A) by using the ρ_b function from the search start position, and then searches the region of B (r_B) by using ρ_b function from the end position of r_A. In order to search the region of A nearest to r_B, the algorithm searches again the region of A (r_{A2}) from the begin position of r_B by using ρ'_b function. Then, the algorithm output the region from the begin position of r_{A2} to the end position of r_B.

The algorithm for ρ_b and ρ_n functions first searches a region of B (r_B) by using the function ρ_b. Then the algorithm check whether r_B contains another region of B, which is placed before the search start position, and the algorithm searches the next region of B when r_B contains another region. When the region r_B does not contains another region of B, the algorithm searches a region of A (r_A) nearest to the region r_B. When such region of A, r_A, does not exists, the algorithm searches another region of A (r_{A1}) from the begin position of B,

```
Input: $p$: start position, $d$: depth restriction, $r_r$: region restriction
Output: the first region of (& $A$ $B$) satisfying the input conditions
$itr_{(\&\ A\ B)}.\rho_b(p, d, r_r) =$
  $p_b = p$
  **while**$p_b$ exists
    $r_A = itr_{(\&\ A\ B)}.itr_L.\rho_b(p, -, r_r)$
    $r_B = itr_{(\&\ A\ B)}.itr_R.\rho_b(p, -, r_r)$
    **if** $r_A.e == r_B.e$ **then**
      **if** $r_A.b \geq p$ and $r_B.b \geq p$ **then**
        **return** $(min(r_A.b, r_B.b), r_A.e)$
      **elsif** $r_A.b \geq p$ and $r_B.b < p$ **then**
        $r_{B2} = itr_{(\&\ A\ B)}.itr_R.\rho'_b(p-1, -, (r_r.b, p-1))$
        **return** $(min(r_B.b, r_{B2}.b), r_A.e)$
      **elsif** $r_A.b < p$ and $r_B.b \geq p$ **then**
        $r_{A2} = itr_{(\&\ A\ B)}.itr_L.\rho'_b(p-1, -, (r_r.b, p-1))$
        **return** $(min(r_A.b, r_{A2}.b), r_B.e)$
      **else**
        $r_{A2} = itr_{(\&\ A\ B)}.itr_L.\rho'_b(p-1, -, (r_r.b, p-1))$
        $r_{B2} = itr_{(\&\ A\ B)}.itr_R.\rho'_b(p-1, -, (r_r.b, p-1))$
        **if** $r_{A2}.b > r_A.b$ and $r_{B2}.b > r_B$ **then**
          $p_b = r_A.e + 1$
        **elsif** $r_{A2}.b > r_A.b \geq p$ **then**
          **return** $(min(r_{A2}.b, r_B.b), r_A.e)$
        **elsif** $r_{A2}.b \geq p$ and $r_{B2}.b \geq p$ **then**
          **return** $(min(r_A.b, r_{B2}.b), r_A.e)$
        **elsif** $r_{A2}.b \geq p$ and $r_{B2}.b \geq p$ **then**
          **return** $(min(r_A.b, r_B.b), r_A.e)$
    **elsif** $r_A.e > r_B.e$ **then**
      $r_{A2} = itr_{(\&\ A\ B)}.itr_L.\rho'_b(p, -, (r_r.b, p))$
      **if** $r_{A2}$ exists **then**
        **if** $r_B$ contains $r_{A2}$ **then**
          **return** $r_B$
        **else**
          $r_{B2} = itr_{(\&\ A\ B)}.itr_R.\rho_b(r_{A2}.b, -, (r_{A2}.b, r_r.e))$
          **if** $r_{B2} == r_B$ **then**
            **return**$(min(r_{A2}.b, r_B.b), r_B.e)$
          **else**
            $p_b = r_A.e$
    **else**
      $r_{B2} = itr_{(\&\ A\ B)}.itr_R.\rho'_b(p, -, (r_r.b, p))$
      **if**$r_{B2}$ exists **then**
        **if**$r_A$ contains $r_{B2}$**then**
          **return** $r_A$
        **else**
          $r_{A2} = itr_{(\&\ A\ B)}.itr_L.\rho_b(r_{B2}.b, -, (r_{B2}.b, r_r.e))$
          **if** $r_{A2} == r_A$ **then**
            **return** $(min(r_A.b, r_{B2}.b), r_A.e)$
          **else**
            $p_b = r_A.e$
```

Figure 3.22: ρ_b function for query (& A B).

```
Output: the next region of $(\&\ A\ B)$ which is stored in $itr_{(\&\ A\ B)}$ satisfying
the input conditions
$itr_{(\&\ A\ B)}.\rho_n() =$
  **return** $itr_{(\&\ A\ B)}.f\rho_b(itr_{(\&\ A\ B)}.r.e + 1, itr_{(\&\ A\ B)}.d, itr_{(\&\ A\ B)}.r_r)$
```

Figure 3.23: ρ_n function for query (& A B).

```
Input: $p$: start position, $d$: depth restriction, $r_r$: region restriction
Output: the first region of $(-\ A\ B)$ satisfying the input conditions
$itr_{(-\ A\ B)}.\tau_b(p, d, r_r) =$
  $r_A = itr_{(-\ A\ B)}.itr_L.\rho_b(p, -, (p, r_r.e))$
  **if** $r_A$ does not exist **then**
    **return** $(-, -)$
  $r_B = itr_{(-\ A\ B)}.itr_R.\rho_b(p, -, (r_A.e, r_r.e))$
  **if** $r_B$ does not exist **then**
    **return** $(-, -)$
  $r_{A1} = itr_{(-\ A\ B)}.itr_L.\rho'_b(r_B.b, -, (r_r.b, r_B.b))$
  **return**$(r_{A1}.b, r_B.e)$
```

```
Output: the next region of $(-\ A\ B)$ which is stored in $itr_{(-\ A\ B)}$ satisfying
the input conditions
$itr_{(-\ A\ B)}.\tau_n() =$
  **return** $itr_{(-\ A\ B)}.\tau_b(itr_{(-\ A\ B)}.r.b + 1, itr_{(-\ A\ B)}.d, itr_{(-\ A\ B)}.r_r)$
```

Figure 3.24: τ_b and τ_n function for query (- A B).

and searches next region of $(-AB)$ from the begin position of r_{A1} with the τ_b function. When r_A exists, the algorithm searches the region of B (r_{B1}) from the end position of r_A, and output the result region from the begin position of r_A to the end position of r_{B1}.

3.5.3 Index for Stand-off Annotation

The list of position and depth value for search are created as the same with the inline annotations. The stand-off annotation contains begin position and end position of annotation regions, these positions are directly indexed, and the depth information is calculated as the same with the inline annotations. Moreover, in order to reduce the number of access times for index list, we constructed a list of position of regions instead of that of token.

Figure 3.26 shows an example of the position list for stand-off annotations in Figure 3.14. For each word, tag and attribute of tag, a list of the region, a pair of begin and end position, and depth value is created. In Figure 3.26, each value in triplex expresses the two position values and depth value respectively. For words, the depth value is always 0, and the position is the offset value for the

```
Input: $p$: start position, $d$: depth restriction, $r_r$: region restriction
Output: the first region of $(-\ A\ B)$ satisfying the input conditions
$itr_{(-\ A\ B)}.\rho_b(p, d, r_r) =$
  $p_b = p$
  while $p_b$ exists
    $r_B = itr_{(-\ A\ B)}.itr_R.\rho_b(p, -, r_r)$
    if $r_B.b < p_b.b$ then
      $r_{B1} = itr_{(-\ A\ B)}.itr_R.\rho_b(r_B.b, -, r_B)$
      if $r_{B1}$ exists then
        $p_b = r_B.e - 1$
    if $p_b != r_B.e - 1$
      $r_A = itr_{(-\ A\ B)}.itr_L.\rho'_b(r_B.b, -, (r_r.b, r_B.b))$
      if $r_A$ exists then
        $r_{A1} = itr_{(-\ A\ B)}.itr_L.\rho_b(r_B.b, -, (r_B.b, r_r.e))$
        return $itr_{(-\ A\ B)}.\tau_b(r_{A1}.b, d, r_r)$
      else
        $r_{B1} = itr_{(-\ A\ B)}.itr_R.\rho_b(r_A.e, -, (r_A.e, r_r.e))$
        if $r_{B1} == r_B$ then
          return $(r_A.b, r_{B1}.e)$
        else
          $p_b = r_B.e - 1$

Output: the next region of $(-\ A\ B)$ which is stored in $itr_{(-\ A\ B)}$ satisfying
the input conditions
$itr_{(-\ A\ B)}.\rho_n() =$
  return $itr_{(-\ A\ B)}.\rho_b(itr_{(-\ A\ B)}.r.e + 1, itr_{(-\ A\ B)}.d, itr_{(-\ A\ B)}.r_r)$
```

Figure 3.25: ρ_b and ρ_n function for query (- A B).

begin and end position of the words. For tags, the depth value is calculated in the same way with the case of in-line annotations, and the positions are begin and end position of tag regions, which are the value in stand-off annotations. Since a position list is created based on regions, the list for begin tag and that for end tag are merged in the position list of tag. For attributes of tags, the depth value and the positions are the same with the tag which the attribute appears.

Although the list in Figure 3.26 is sorted in the order of the begin position, or depth value in the case that the begin positions are the same, we constructed two types of lists of regions, one is the list in the order of the begin position, and the other is the list in the order of the end position. The list in the begin position order is accessed by the functions τ, τ' in the algorithm, and the list in the end position order is accessed by the functions ρ, ρ'.

p53	0-3-0, ...
is	4-6-0, ...
phosphorylated	7-21-0, ...
to	22-24-0, ...
activate	25-32-0, ...
cd25	34-38-0, ...
...	
<sentence>	0-38-0, ...
<phrase>	0-38-0, 0-3-1, 0-3-2, 4-38-1,...
phrase:id="0"	0-38-0, ...
phrase:cat="S"	0-38-0, ...
phrase:head="4"	0-38-0, ...
phrase:lex_head="6"	0-38-0, 4-38-1, 4-6-2, ...
phrase:id="1"	0-3-1, ...
phrase:cat="NP"	0-3-1, 0-3-2, ...
...	...
<word>	0-3-0, 4-6-0, ...
word:id="3"	0-3-0, ...
word:pos="NN"	0-3-0, ...
...	...

Figure 3.26: Inverted position index for stand-off annotation in Figure 3.14

3.6 Probabilistic Retrieval Model for Structured Document Retrieval

3.6.1 Query and Document Representation for Structured Document Retrieval

In traditional vector space model, queries and documents are represented by a vector of weight of keywords,

$$q = (w_1^q, w_2^q, ..., w_n^q)$$

$$d = (w_1^d, w_2^d, ..., w_n^d)$$

where w_i^q is a weight of a word w_i in the query q and w_i^d is a weight of a word w_i in the document d. Various value is used as a weight of a word w_i, such as binary value, i.e. $w_i^d = 1$ when w_i appears in the document d, pure frequency or tf*idf value, and so on. The similarity between a query and a document is calculated by the cosine value,

$$cos(q, d) = \frac{q \cdot d}{|q||d|}$$

where $|q|$ is the length of query vector, that is, $|q| = \sqrt{\Sigma_i w_o^q}$. The Binary Independent Model [75, 64, 65], which is one of the probabilistic model for retrieval, also models a query and a document with a vector, but the element of the vector is a binary value.

We extended the vector models to apply to the structured document retrieval. When the user would like to search about the relation "p53 activates CD25," the query for structured document retrieval in the form of region algebra is,

```
(> [sentence]
   (& [tok arg1=$s arg2=$o base="activate"]
      (> [cons id=$s] "p53")
      (> [cons id=$o] "cd25")))
```

The documents contains whole the relation “p53 activates CD25,” which means that the documents containing the region matches to the above query of region algebra, are relevant to the query of course, but the documents containing the relation “p53 activates” or “activates CD25” can be useful for the users, and the documents containing these relations will be more relevant to the users' queries than the documents containing only the keywords “p53,” “activates” and “CD25.” So we extended the element of vectors from simple keywords to the structured queries. In this example, a set of queries will be as followings:

```
"p53"
"cd25"
[tok base="activate"]
(& [tok arg1=$s base="activate"] (> [cons id=$s] "p53"))
(& [tok arg2=$o base="activate"] (> [cons id=$o] "cd25"))
```

Although we can simply define the similarity between a structured query and a document by the cosine value, we have to consider the dependency between the sub-queries. We derive the similarity function by extending Binary Independent Model for structured retrieval with considering the dependency between sub-queries.

3.6.2 Extend BIM Model for Structured Document Retrieval

The Binary Independent Model assumed the independency of keywords in a query. We extend the model for the structured queries and documents.

We will estimate the probability function $P(R = 1|d, q)$, where d is a document, q is a query and R is a relevance The query q is represented by a vector of sub-queries, $Q = (q_1, q_2, ..., q_n)$ where q_i is a word or a structured query. When the sub-queries are only keywords, we can suppose that the sub-queries re independent. But when some sub-queries have a structure, we cannot ignore the dependency among the sub-queries. For example, when the given query q is

```
(& [tok arg1=$s arg2=$o base="activate"]
   (> [cons id=$s] "p53")
   (> [cons id=$o] "cd25"))
```

the sub-queries q_i are as followings:

```
"p53"
"cd25"
```

```
[tok base="activate"]
(& [tok arg1=$s base="activate"] (> [cons id=$s] "p53"))
(& [tok arg2=$o base="activate"] (> [cons id=$o] "cd25"))
```

There is obvious dependency in these queries. When a document contains regions matching the query

```
(& [tok arg1=$s base="activate"]
   (> [cons id=$s] "p53")),
```

the region contains regions matching the query `[tok base="activate"]` and `"p53"` by necessity, that is, the document contains the relation " 'p53' is the subject of 'activate' " also contain the word "p53" and "activate" inevitably. Since there is dependency among the sub-queries, the score calculation for some queries are duplicated when the scoring model for traditional keyword-based retrieval are applied to the structured document retrieval.

The document d is also represented by a vector of appearance of regions, $X_d = (x_1, x_2, ..., x_n)$ where $x_i = 1$ if a region matched the query q_1 exists in the document d. We assume here that the relevance of a document is independent of the relevance of the other documents.

As the binary independent model, we will derive a ranking function with calculating odds of relevance, that is

$$O(R|X,Q) = \frac{P(R=1|X,Q)}{P(R=0|X,Q)}$$

for making things easier. By using Bayes rule, we can have

$$P(R=1|X,Q) = \frac{P(X|R=1,Q)P(R=1|Q)}{P(X|Q)}$$

$$P(R=0|X,Q) = \frac{P(X|R=0,Q)P(R=0|Q)}{P(X|Q)}.$$

Then the above formula is derived as

$$\begin{aligned} O(R|X,Q) &= \frac{P(R=1|X,Q)}{P(R=0|X,Q)} \\ &= \frac{\frac{P(X|R=1,Q)P(R=1|Q)}{P(X|Q)}}{\frac{P(X|R=0,Q)P(R=0|Q)}{P(X|Q)}} \\ &= \frac{P(R=1|Q)}{P(R=0|Q)} \frac{P(X|R=1,Q)}{P(X|R=0,Q)} \end{aligned}$$

The left term of the last expression, $\frac{P(R=1|Q)}{P(R=0|Q)}$, is a constant for a given query. In the Binary Independent Model for keyword-based retrieval, we can suppose the assumption that appearance of a word is independent from the other words. But in the structured document retrieval, we cannot suppose the assumption because of the dependency among the queries.

We incorporate the dependency into the model. Here, we suppose a simple example, $Q = (q_1, q_2, q_3)$ and $q_3 = (\&q_1q_2)$. For these queries, when a document contains regions matching the query q_3, the document inevitably contains regions matching q_1 and q_2. By considering the dependency, the probability of the document can be expressed as following:

$$\begin{aligned} P(X) &= P(x_1) \cdot P(x_2) \cdot P(x_3|x_1, x_2) \\ &= P(x_1) \cdot P(x_2) \cdot \frac{P(x_1, x_2, x_3)}{P(x_1, x_2)} \end{aligned}$$

By incorporating the dependency among the sub-queries, we can transform the formula for this example as follows.

$$\begin{aligned} \frac{P(X|R=1,Q)}{P(X|R=0,Q)} &= \frac{P(x_1|R=1,Q)}{P(x_1|R=0,Q)} \cdot \frac{P(x_2|R=1,Q)}{P(x_2|R=0,Q)} \cdot \frac{P(x_3|x_1,x_2,R=1,Q)}{P(x_3|x_1,x_2,R=0,Q)} \\ &= \frac{P(x_i|X_i,R=1,Q)}{P(x_i|X_i,R=0,Q)} \end{aligned}$$

where X_i is a set of variables $\{x_{j_1^i}, ..., x_{j_n^i}\}$ which have a dependency to variable x_i. In the above example, $X_1 = \{\}$, $X_2 = \{\}$ and $X_3 = \{x_1, x_2\}$. As is the case of the original Binary Independent Model, since x_i is a binary value, the terms can be separated.

$$\frac{P(X|R=1,Q)}{P(X|R=0,Q)} = \prod_i \frac{P(x_i=1|X_i,R=1,Q)}{P(x_i=1|X_i,R=0,Q)} \cdot \prod_i \frac{P(x_i=0|X_i,R=1,Q)}{P(x_i=0|X_i,R=0,Q)}$$

Here, let $p_i = P(x_i = 1|X_i, R = 1, Q)$ be the probability of a term appearing in a document relevant to the query, and $u_i = P(x_i = 1|X_i, R = 0, Q)$ be the probability of a term appearing in a nonrelevant document. With the simplifying assumption that the terms not appearing in the query appear in relevant and non-relevant documents in the same probability, that is, $p_i = u_i$ when $x_i = 0$, the above formula can be transformed,

$$\begin{aligned} & \prod_i \frac{P(x_i=1|X_i,R=1,Q)}{P(x_i=1|X_i,R=0,Q)} \cdot \prod_i \frac{P(x_i=0|X_i,R=1,Q)}{P(x_i=0|X_i,R=0,Q)} \\ = & \prod_{i:x_i=q_i=1} \frac{p_i}{u_i} \cdot \prod_{i:x_i=0,q_i=1} \frac{1-p_i}{1-u_i} \\ = & \prod_{i:x_i=q_i=1} \frac{p_i(1-u_i)}{u_i(1-p_i)} \cdot \prod_{i:q_i=1} \frac{1-p_i}{1-u_i} \end{aligned}$$

Since the right product term is a constant when a query is given, only the left product term should be estimated in order to ranking the documents. *Retrieval status value* (RSV) is defined by the logarithm of the term,

$$RSV_d = \log \prod_{i:x_i=q_i=1} \frac{p_i(1-u_i)}{u_i(1-p_i)} = \sum_{i:x_i=q_i=1} \log \frac{p_i(1-u_i)}{u_i(1-p_i)}$$

.

Here, we define the value c_i for a query q_i as follows:

$$c_i = \log \frac{p_i(1-u_i)}{u_i(1-p_i)} = \log \frac{p_i}{1-p_i} + \log \frac{1-u_i}{u_i}.$$

In Binary Independent Model, the idf value can be derived from the right term of the above formula, $\log \frac{1-u_i}{u_i}$. We would like to corresponding value in the model including dependency between the queries. From the definition, the value u_i for the sub-query q_3 is derived as follows:

$$\begin{aligned}
u_3 &= P(x_3 = 1|x_1, x_2, R = 0, q) \\
&= P(x_3 = 1|x_1 = 1, x_2 = 1, R = 0, q) \\
&\quad + P(x_3 = 1|x_1 = 1, x_2 = 0, R = 0, q) \\
&\quad + P(x_3 = 1|x_1 = 0, x_2 = 1, R = 0, q) \\
&\quad + P(x_3 = 1|x_1 = 0, x_2 = 0, R = 0, q)
\end{aligned}$$

By the definition of query q_3, $x_3 = 1$ only if $x_1 = 1 \wedge x_2 = 1$, that is,

$$\begin{aligned}
&P(x_3 = 1|x_1 = 1, x_2 = 0, R = 0, q) = 0, \\
&P(x_3 = 1|x_1 = 0, x_2 = 1, R = 0, q) = 0 \text{ and} \\
&P(x_3 = 1|x_1 = 0, x_2 = 0, R = 0, q) = 0.
\end{aligned}$$

Then, the u_3 is,

$$\begin{aligned}
u_3 &= P(x_3 = 1|x_1 = 1, x_2 = 1, R = 0, q) \\
&\quad + P(x_3 = 1|x_1 = 1, x_2 = 0, R = 0, q) \\
&\quad + P(x_3 = 1|x_1 = 0, x_2 = 1, R = 0, q) \\
&\quad + P(x_3 = 1|x_1 = 0, x_2 = 0, R = 0, q) \\
&= P(x_3 = 1|x_1 = 1, x_2 = 1, R = 0, q) \\
&= \frac{P(x_1 = 1, x_2 = 1, x_3 = 1|R = 0, q)}{P(x_1 = 1, x_2 = 1|R = 0, q)}
\end{aligned}$$

As the same with Binary Independent Model for keywords, the statistics for non-relevant documents can be estimated by statistics for the whole document collections since the relevant documents account for only a small percentage in a document collection. Let N is the number of documents and df_{q_i} is the number of documents containing the region matching q_i. The probability u_3 is derived as follows:

$$\begin{aligned}
u_3 &= \frac{P(x_1 = 1, x_2 = 1, x_3 = 1|R = 0, q)}{P(x_1 = 1, x_2 = 1|R = 0, q)} \\
&= \frac{\frac{df_{q_3}}{N}}{\frac{df_{(\&q_1q_2)}}{N}} \\
&= \frac{df_{q_3}}{df_{(\&q_1q_2)}}.
\end{aligned}$$

Then, the weighting value corresponds to IDF in keyword-based search is,

$$\begin{aligned}\log\frac{1-u_3}{u_3} &= \log\frac{1-\frac{df_{q_3}}{df_{(\&q_1q_2)}}}{\frac{df_{q_3}}{df_{(\&q_1q_2)}}}\\ &= \log\frac{df_{(\&q_1q_2)}-df_{q_3}}{df_{q_3}}.\end{aligned}$$

Generalizing this formula, when there is a query set $\{q_i|i=1,...,n\}$ and a query q_j contains a $q_{j_1},...,q_{j_n}$ as sub-queries, the weighting factor for q query q_j corresponds to IDF value in Binary Independent Model is $\log\frac{df_{(\&q_{j_1}\cdots q_{j_n})}-df_{q_j}}{df_{q_j}}$. We called this value as *relative IDF* value.

3.6.3 Ranking Algorithm

The algorithm to construct a ranking list for each query is as follows:
The inputs of the algorithm are two types of queries, *Filtering Query* and a list of *Scoring Query*. The Filtering Query is a keyword-based query, which is used to decreasing the number of documents in which relevance score is calculated in the next step. The Scoring Query is used to calculate relevance score. First, the algorithm searches documents with the Filtering Query. Then, a score is calculated for each document based on the frequency of regions matching to the Scoring Query in the document. Finally, a ranking list is constructed based on the scores. Given a list of Scoring Queries $q_1,...,q_n$, the score of a document D is calculated with Okapi BM25 [66, 31] with replacing the IDF value into relative IDF and replacing the frequency of terms into the frequency of regions matching to the query, that is,

$$S(D,q_1,...,q_n)=\sum_{i=1}^{n} RIDF(q_i)\cdot\frac{rf_{q_i}\cdot(k_1+1)}{rf_{q_i}+k_1\cdot(1-b+b\cdot\frac{|D|}{DL_{ave}})}$$

where rf_{q_i} is the number of regions matching to the query q_i in the document D, k_1 and b are parameters, (we used $k_1=2.0$ and $b=0.75$ in the experiments), $|D|$ is the length of the document D, DL_{ave} is the average length of documents in the collection and $RIDF(q_i)$ is the relative IDF value for the query q_i, which is defined as

$$RIDF(q_i)=\log\frac{(df_{(\&q_{i_1}\cdots q_{i_n})}-df_{q_i})+0.5}{df_{q_i}+0.5}.$$

where df_{q_i} is the number of documents which contain at least one region matching the query q_i in the document set, and $q_{i_1}...q_{i_n}$ are the sub-queries of q_i in Scoring Queries.

Chapter 4

MEDIE: Semantic Retrieval System for MEDLINE

The number of journal articles in the biomedical science area is numerous and rapidly increasing. The articles in MEDLINE, around 20 million of references in biomedical area, are mutually related and the research fields of articles are wide-ranging. Although researchers would like to follow articles in their field, it is hard to find articles or knowledges written in articles concerning their field without any support for accessing. Computer-assisted access for the articles are needed to acquire target articles or knowledges from the huge textbase.

The main-stream technology used to assist researchers for accessing the huge textbase has been classical keyword-based search. Although the keyword-based search is simple and useful for simply searching or filtering articles, researchers demand to search with more complex queries, which can specify relational concepts. These relational concepts have a wide variety. Well-established concepts, like protein-protein interaction, can be extracted from articles in advance as structured data and searched by querying in, e.g., a relational database. More vaguely defined concepts, which are difficult to capture in advance, can be extracted by searching the texts by specifying conditions on BioNLP results annotated to the texts in advance.

We constructed a search system, *MEDIE*, which is a real-time search system over richly NLP-annotated textbase. The target data of the system is MEDLINE, which contains approximately 19 million references to journal articles in biomedical sciences. This system is based on a search for semi-structured text, which can search document fragments in textbases with queries specifying relational concepts extracted from articles in advance or expressed with the integration of several types of NLP annotations.

4.1 MEDIE system

MEDIE system is a semantic retrieval system for MEDLINE, a database of paper abstracts in biomedical area. MEDLINE contains around 20 million of paper abstracts, and the number of them is rapidly increasing. Although researchers

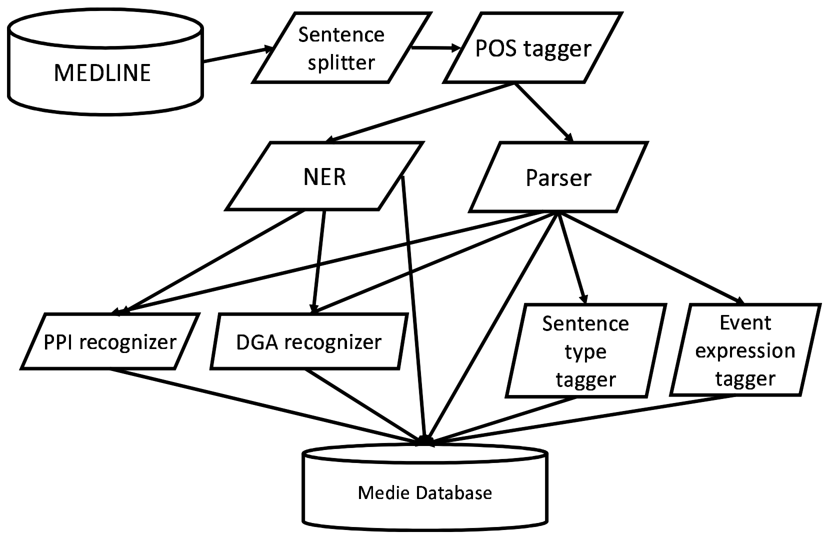


Figure 4.1: Overview of the pre-processing system of MEDIE

would like to follow articles in their own field, it is hard to find articles or knowledges written in articles concerning their field without any support for accessing. Computer-assisted access for the articles are needed to acquire target articles or knowledges from the huge textbase.

In order to enable a search system in which user can query the advanced search request beyond a set of keywords or can retrieve not a whole document but a requested information itself, we constructed semantic retrieval system for MEDLINE by using our framework and NLP modules. We processed documents by various kind of NLP modules, annotate the target documents with the result from NLP modules and indexed the documents with annotations in our search framework of tag-annotated text search. Users can write an advanced search condition by specifying the annotations by NLP modules in the documents. For example, the relation among two substances can be specified by using the results from a parser, that is, two substances are combined by a verb in subject-verb-object relation. Some sorts of terms are recognized as a term with a type and the terms are connected to the actual object by using dictionaries. In the search process, the system regards the terms which are connected to the same object, that is the synonyms, as the same term, and the system can treat words in conceptual level, such as “gene,” “disease,” and so on, by using the annotations from named entity recognizer.

We first explain the NLP modules used in constructing annotations for target data, and then we explain about the whole MEDIE system.

4.2 Natural Language Processing Module

We pre-processed MEDLINE articles by many NLP modules. Figure 4.1 shows an overview of the pre-preprocessing system, which consisting of three layers. In the first layer, the biomedical text in MEDLINE articles is split into sentences, and then the sentences are tokenized and POS-tagged. as well as the high-level tools, were tuned for the biomedical text genre.

In the second layer, syntactic analysis of the text is done by a parser, and technical terms, such as gene/protein names and disease names, are recognized by a named entity recognizer (NER). The syntactic and terminological analyses made in this stage are especially important because they are directly queried in the search system and also utilized in the NLP modules in the next layer.

The NLP modules in the third layer do specialized processing on the biomedical text; namely, recognition of protein-protein interactions (PPI) and gene-disease associations (GDA) mentioned in the text, recognition of the rhetorical roles of the sentences (e.g., object, method, conclusion etc.), and recognition of biological "events" described in the text.

In the followings, we will introduce the NLP modules pre-processed in MEDIE systems.

Enju HPSG Parser and its Adaptation to Biology and Medical Text Domains

A parser receives a sentence and returns some kinds of analyzed structures for it such as phrase structures. Syntactic parsers are fundamental tools for various NLP tasks including information extraction and text mining. A parser receives a sentence and returns its syntactic structure represented in a certain formalism. Phrase structures and dependency structures have been widely used as the representational formalism for many NLP applications. Many types of parsers have been used in various NLP tasks including IE and text mining, and Recently, parsers that compute deeper analyses have become available for the processing of real-world text, and our parser Enju, [52, 55], is such a parser.

Our parser, Enju [52, 55], is a deep parser that produces the syntactic and semantic structure of a sentence in one shot The parser is based on the head-driven phrase structure grammar (HPSG) formalism [59] and outputs predicate argument structures (PAS) that explicitly express the semantic relations among words. Such a deep representation is beneficial for the purpose of the present study because this representation represents relational concepts in an abstract manner, while ignoring the differences in the surface level, such as differences in voices (active/passive).

An important sub-component of a syntactic parser is the disambiguation model, which is used to find a correct interpretation of an input sentence. Since most of natural language sentences have many grammatically admissible, but often strange, interpretations, the disambiguation model is crucial for accurate syntactic/semantic analysis of the text. The disambiguation model of Enju was trained using the Wall Street Journal portion of the Penn Treebank (PTB) corpus

	Text genre (corpus)	
Model	News article (PTB)	Biomedical (Genia)
Enju (original)	89.99/89.63/89.81	86.71/86.08/86.39
Enju (re-trained)	-	90.23/90.08/**90.15**

Table 4.1: Improvement of parsing accuracy by re-training (LP/LR/F_1)

[42], whereas our target text consists of biomedical articles.

Due to the difference in the text genres, the original disambiguation model of Enju provided numerous incorrect analyses of the MEDLINE text. This kind of problems is called domain portability and is known to damage the performance of NLP applications severely. Therefore, there has been much research on adapting parsers or other NLP tools to a specific domain. This kind of problems is called domain portability and is known to damage the performance of NLP applications severely. We developed an effective method for re-training the model in another domain [23, 24]. By re-training the model, the parser could maintain high performance in the target domain.

We evaluated the impact of re-training the model of Enju. The original parser, Enju, was trained using Sections 02-21 of PTB [55]. For re-training in the target domain, we used the GENIA treebank [33], which consists of 1,200 abstracts extracted from MEDLINE. We divided the GENIA treebank into three sets of 900, 150, and 150 abstracts (resp. 8,127, 1,361, and 1,360 sentences), and used these sets as training, development, and final evaluation data. For evaluating the parsing performance, we measured the labeled precision (LP), recall (LR), and the harmonic-mean (F_1-score) on the predicate-argument dependencies produced by the parser. The experimental results revealed that the re-trained model could improve the parsing accuracy for the biomedical domain by 3.84 F_1-score points (Table 4.2.

Named Entity Recognition

The named-entity recognition module, referred to hereinafter as NERsuite, is a machine-learning-based NER toolkit that uses a conditional random field machine learning toolkit referred to as CRFsuite [57]. The NERsuite exploits diverse features proposed in previous studies [38, 40, 54]. We pre-processed the biomedical text with the GENIA sentence splitter[1], a fine-grained tokenizer, the GENIA tagger [73] and dictionaries, EntrezGene[41], and the UMLS Metathesaurus[71], in order to extract feature values. For recognizing entities, the system uses the IOBES chunk label set because this set provides better discrimination results, as mentioned in a previous study [60]. We evaluated the NERsuite in the BioCreative2 Gene Mention Recognition task. The performance was ranked between the second and third systems in the original shared task, as shown in Table 4.2.

Rank	Precision	Recall	F1-score	Add. tech.
1	88.48%	85.97%	87.21%	S, G, P
2	89.30%	84.49%	86.83%	E, G, P
NERsuite	90.12%	83.41%	86.64%	G
3	84.93%	88.28%	86.57%	E
4	87.27%	85.41%	86.33%	E, P
5	85.77%	86.80%	86.28%	G
6	82.71%	89.32%	85.89%	G, P
7	86.97%	82.55%	84.70%	G, A

Table 4.2: Performance comparison with other systems of BioCreative 2 competition. The Add. tech. column lists additional techniques used for these systems (S: semi-supervised method, E: ensemble classifier, G: gazetteer, P: post-processing, and A: abbreviation resolution).

Protein-Protein Interaction (PPI) recognition

The PPI module brings together all other components from the NLP pipeline to perform PPI extraction. This includes sentence splitting, part-of-speech (POS) tagging, protein name recognition, protein database-identifier mapping, syntactic parsing with different language models, and machine learning with tree and graph kernels.

The PPI module reads stand-off annotations that include outputs from the above-mentioned modules, together with the original text file, and performs two tasks. The PPI module first extracts features from the parsers and then runs the support vector machine with kernels (SVM-TK [30, 53]). We trained the SVM on 225 PPI-annotated MEDLINE abstracts in the AIMed corpus [8]. The AIMed corpus and the syntactic features used for machine learning are described in greater detail in [68].

A prototype of the proposed PPI system was used for the BioCreative2 PPI challenge. This prototype did not include the SVM machine learning functionality. However, even without this extra module, the prototype still achieved an F_1-score of 18.7% on the SwissProt IPS test set, making it the 6th best system among 16 participants. We also evaluated the PPI extraction module by 10-fold cross-validation on the AIMed corpus, and the results are shown in Table 4.2. The comparison of the different PPI systems is based on the precision and recall of the predicted PPIs and the harmonic mean (F_1-score). Even though Giuliano et al.'s system [21] appears to be 10%-points better than the proposed system, an attempt at reproducing their results did not provide results that were as good as those that were reported. The reason for this is a subject for future research.

Recognition of gene-disease associations

Gene and disease name pairs co-occurring in a sentence have some potential important associations. However, these co-occurring pairs also have numerous

	Precision(%)	Recall(%)	F_1(%)
Our system	61.3	40.7	48.6
[9]	**73.9**	35.2	47.7
[49]	54.2	42.6	47.7
[21]	60.9	**57.2**	**59.0**

Table 4.3: Comparison of different PPI systems evaluated using AIMed F

	Precision(%)	Recall(%)	F_1-score(%)
Baseline	84.6	97.6	90.6
GDA without NER	88.0	95.1	91.4
GDA with NER	95.8	99.8	97.8

Table 4.4: Performance of GDA recognition

incorrect associations. In order to filter out these incorrect associations, we used a machine learning technique. Various context features have been used to train the filter, including candidate gene and disease names, one or two adjacent words of candidate gene and disease names, the order of candidate gene and disease names, the distance between candidate gene and disease names, and bags of words in sentences.

In order to evaluate the GDA recognition system, a GDA annotated corpus has been constructed. A total of 3,999 co-occurrences have been selected from among 2,939 MEDLINE abstracts by biologists. Six biologists annotated the existence of associations with reference to the original abstracts that include co-occurrences. The inter-annotator agreement among the six annotators had an F_1-score of 0.930, and 3,099 (77.5%) co-occurrences were found to contain correct associations.

We performed 10-fold cross validation and measured the precision, recall, and F_1-score of the system for all experiments. Table 4.2 describes the experimental results for baseline, GDA without NER, and GDA with NER. In a baseline experiment, we assumed that the co-occurrence of gene and disease names by NER in a sentence indicates an association. These results clarify the effectiveness of machine learning-based named entity recognition (NER) and GDA recognition methods.

Event expression recognition

Biological molecular events (bio-events) are important in understanding the dynamics of biological systems. In order to detect in text locations in which bio-events are discussed, we developed a dictionary of event expression patterns. The patterns are extracted from the GENIA event annotations, which were made to

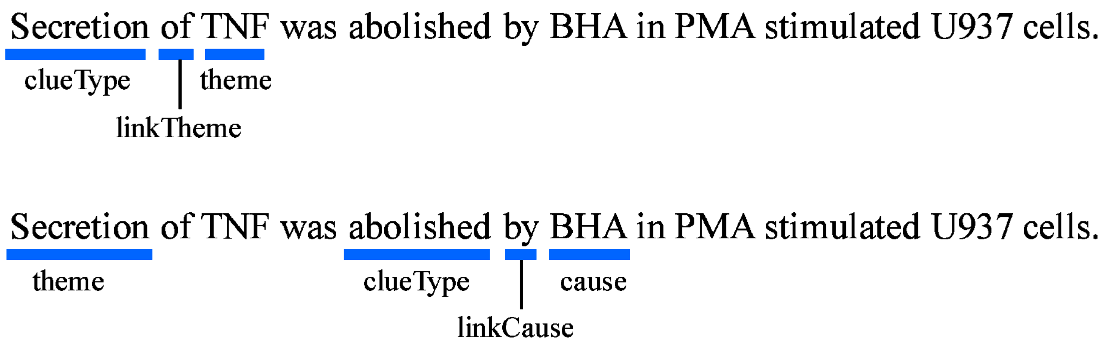

Secretion of TNF was abolished by BHA in PMA stimulated U937 cells.
clueType
theme
linkTheme
Secretion of TNF was abolished by BHA in PMA stimulated U937 cells.
theme
clueType
cause
linkCause

Figure 4.2: Event annotation

1,000 Medline abstracts [34]. Since the annotations are based on the GENIA event ontology, which is a simplified version of the gene ontology, the dictionary also contains patterns of event expressions based on event classification in the ontology.

Figure 4.2 shows two instances of GENIA event annotation of an example sentence. The first identifies the bio-event "secretion of TNF", which is classified as a protein catabolism event, according to the GENIA ontology. The second identifies a negative regulation of the secretion event. The textual expressions, which take part in the event expressions, are marked up in the GENIA event annotation. In this example, "secretion" and "abolished" are textual expressions that indicate a localization and a negative regulation event, respectively (clueType). The terms "TNF" and "secretion" are the themes of the two respective events. Note that "secretion" is simultaneously the clueType of the first event and theme of the second event. The term "BHA" is the cause of the second event. The term "of" links the theme and the clueType in the first event (linkTheme), and the term "by" links the cause and the clueType in the second event (linkCause).

The dictionary of event expression patterns is a collection of these event annotation instances. Based on the above example, the following two patterns are extracted and stored in the dictionary:

Localization:	*secretion*/NN (*of*/IN THEME)
Negative_regulation:	*abolish*/VBN (THEME) (*by*/IN CAUSE)

In order to raise the specificity of the event patterns, the POS label predicted by the GENIA tagger is attached to each word.

Sentence Role Classification

Scientific abstracts tend to share a similar rhetorical structure. Namely, an abstract usually begins with a description of background information, followed by the target problem, a proposed solution to the problem, an evaluation of the solution, and, finally, the conclusion of the paper. Prior knowledge of the rhetorical structure of abstracts is useful for improving the performance of information

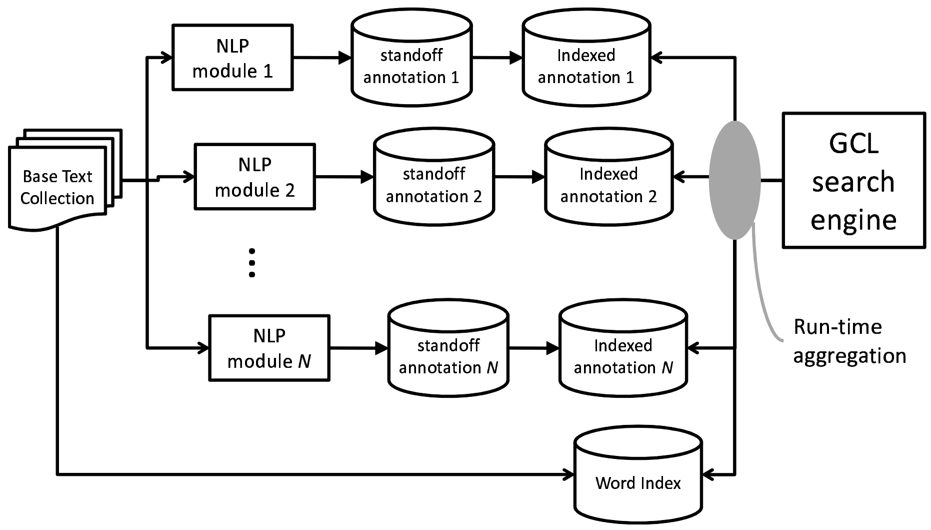


Figure 4.3: Overview of the back-end database architecture of MEDIE

retrieval systems by, for example, assigning more weight to sentences that refer to a purpose or conclusion. We developed a system called Abstruct [27], which categorizes sentences in scientific abstracts into four categories: objective, methods, results, and conclusions. Formalizing the task as a sequence labeling problem, we modeled the conditional probability of a sequence of section names $\mathbf{y} = (y_1, \ldots, y_n)$, given an abstract consisting of sentences $\mathbf{x} = (x_1, \ldots, x_n)$. The conditional probability distribution was modeled by conditional random fields (CRFs). Trained on 50,000 MEDLINE abstracts with section names annotated, the system achieved a per-sentence accuracy of 95.5% and a per-abstract accuracy of 68.8%. The system showed a more than 10% improvement in per-abstract accuracy, as compared with conventional approaches.

4.3 Databases for Retrieval

Figure 4.3 shows an overview of the back-end database architecture of MEDIE. The NLP modules are applied to texts, and the results are converted to the form of stand-off annotations. The inverted index for the regions of stand-off annotations is constructed in order of the begin and end positions. The inverted indexes are constructed independently for the results of each NLP module, and the evaluation of queries in which various types of NLP modules are integrated is implemented by run-time aggregation of the inverted index in the search phase. The independent construction of inverted indices for the annotations enables easy management of annotations in the database, the addition of annotations from new NLP modules, or the deletion of a type of annotations from the database, without having to reconstruct the entire database. This mechanism is essential for the ever-extending nature of the annotated textbase, to which we continue to add new annotations produced by novel NLP technologies.

```
0  38  sentence id="s1"
0  38  cons   id="0" cat="S"  head="4"
0   3  cons   id="1" cat="NP" head="2"
0   3  cons   id="2" cat="NP" head="3"
0   3  tok    id="3" cat="N"  base="p53"
4  37  cons   id="4" cat="VP" head="5"
4   6  cons   id="5" cat="VP" head="6"
4   6  tok    id="6" cat="V"  base="be" arg1="1" arg2="7"
7  37  cons   id="7" cat="VP" head="8"
7  20  cons   id="8" cat="VP" head="9"
7  20  tok     ...
```

Figure 4.4: Example of stand-off annotation

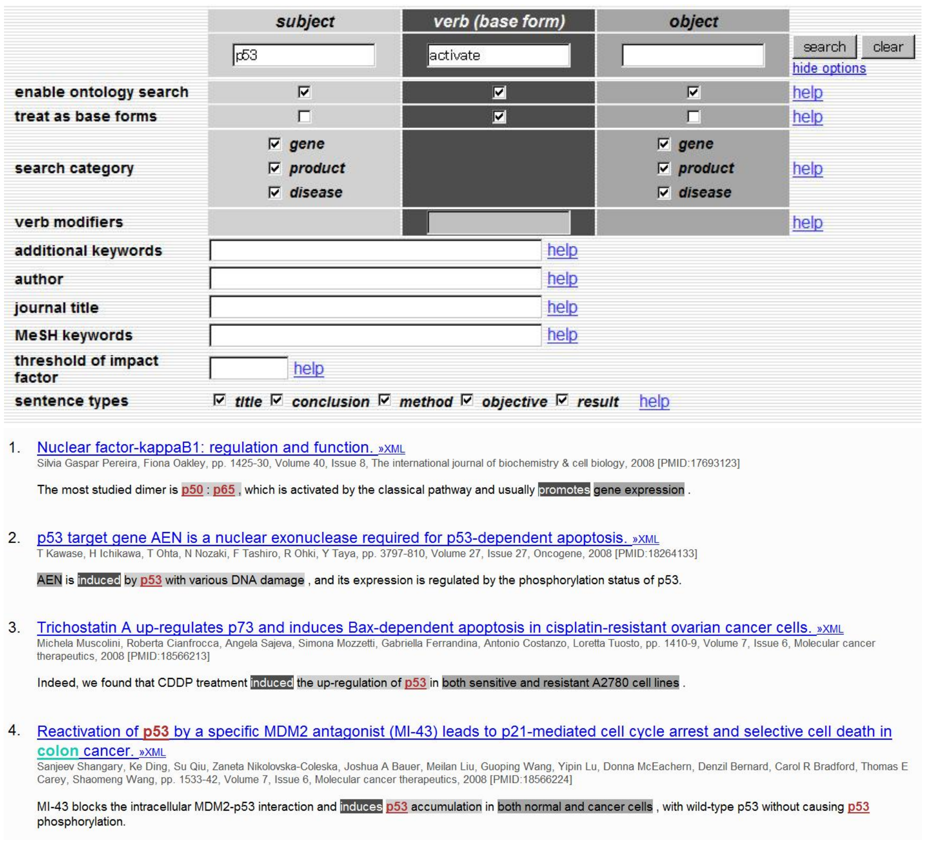
The results given by the NLP modules are associated to the text through a stand-off annotation scheme. Figure 4.4 shows a snippet of the stand-off annotations. The first and second columns are respectively the begin and end position of the tag-annotated text span. The third is the name of the tag, and the rest is a list of attributes. Stand-off annotations enable easy management of text and annotations because the text and corresponding annotations are stored separately. The text and annotations are stored in the MEDIE database, with an index data structure that facilitates fast structural search in the annotated text. The rest of this section gives a more detailed description of the NLP tools.

4.4 Search and Browsing User Interface

MEDIE [51] is an intelligent search system for the MEDLINE database. It gives easy access to the very flexible search functionality based on the structure search engine and the extensive off-line processing of the whole MEDLINE database by the various NLP tools described in the previous sections. The user-interface is implemented as a WEB-application that works in web-browsers.

Snapshots of the MEDIE user-interface is shown in Figure 4.5. In the example, the user inputs ‘p53’ and ‘activate’ respectively in the subject and the verb field. Note that the subject/object field is for specifying not a syntactic subject/object but a *semantic* subject/object; hence, sentences like “*protein X is activated by p53*” or even “*p53 affects X by activating protein Y*” will be retrieved for the query. The search results are shown as a list of articles where each item includes the title and other bibliographic information of the article, and a snippets from the abstract in which the queried relation is found.

The user-interface of the MEDIE system creates a search query expressed in the region algebra formula from a few input fields given by the user, and then it shows the retrieved results. In the example session shown in Figure 4.5, the user input `subject=`“p53” and `verb=`“activate” and this was translated to the following query:



	subject	verb (base form)	object	
	p53	activate		search clear hide options
enable ontology search	✓	✓	✓	help
treat as base forms	☐	✓	☐	help
search category	✓ gene ✓ product ✓ disease		✓ gene ✓ product ✓ disease	help
verb modifiers				help
additional keywords				help
author				help
journal title				help
MeSH keywords				help
threshold of impact factor				help
sentence types	✓ title ✓ conclusion ✓ method ✓ objective ✓ result			help

1. Nuclear factor-kappaB1: regulation and function. »XML
   Silvia Gaspar Pereira, Fiona Oakley, pp. 1425-30, Volume 40, Issue 8, The international journal of biochemistry & cell biology, 2008 [PMID:17693123]

   The most studied dimer is p50 : p65 , which is activated by the classical pathway and usually promotes gene expression .

2. p53 target gene AEN is a nuclear exonuclease required for p53-dependent apoptosis. »XML
   T Kawase, H Ichikawa, T Ohta, N Nozaki, F Tashiro, R Ohki, Y Taya, pp. 3797-810, Volume 27, Issue 27, Oncogene, 2008 [PMID:18264133]

   AEN is induced by p53 with various DNA damage , and its expression is regulated by the phosphorylation status of p53.

3. Trichostatin A up-regulates p73 and induces Bax-dependent apoptosis in cisplatin-resistant ovarian cancer cells. »XML
   Michela Muscolini, Roberta Cianfrocca, Angela Sajeva, Simona Mozzetti, Gabriella Ferrandina, Antonio Costanzo, Loretta Tuosto, pp. 1410-9, Volume 7, Issue 6, Molecular cancer therapeutics, 2008 [PMID:18566213]

   Indeed, we found that CDDP treatment induced the up-regulation of p53 in both sensitive and resistant A2780 cell lines .

4. Reactivation of p53 by a specific MDM2 antagonist (MI-43) leads to p21-mediated cell cycle arrest and selective cell death in colon cancer. »XML
   Sanjeev Shangary, Ke Ding, Su Qiu, Zaneta Nikolovska-Coleska, Joshua A Bauer, Meilan Liu, Guoping Wang, Yipin Lu, Donna McEachern, Denzil Bernard, Carol R Bradford, Thomas E Carey, Shaomeng Wang, pp. 1533-42, Volume 7, Issue 6, Molecular cancer therapeutics, 2008 [PMID:18566224]

   MI-43 blocks the intracellular MDM2-p53 interaction and induces p53 accumulation in both normal and cancer cells , with wild-type p53 without causing p53 phosphorylation.

Figure 4.5: MEDIE user-interface. top: query input form, bottom: search results

```
[article] >> (
  ([sentence sentence_id="$sentence"] >> (
    [event_expression
       event_type="Positive_regulation"
       arg1=$subject arg2=$object id=$verb]
    & ([phrase id="$subject"] > (
         [entity_name gena_id="GDM017078"] |
         [entity_name gena_id="GMM053612"] |
         [entity_name gena_id="GRN004619"])))))
```

In the query, the subject "p53" is normalized (and expanded) to three `[entity_name]` tags with different protein IDs (`gena_id` attributes), connected with OR-operators ('`|`'). This OR-ed expression will match many different expressions, such as "p53", "P53" or "P-53," given that the expressions in the text have been recognized as referring to one of the three protein concepts "`GDM017078`", "`GMM053612`", or "`GRN004619`." Any synonyms of the protein name "p53" can thus be matched as the subject of the relation, given that the named entity recognizer have annotated the synonym as an `entity_name` with It means that the protein name "p53" is represented as a set , each of which is a possible concept of the named protein "p53." Note also that the verb "activate" is translated to a type of biological event, `Positive_regulation`, in the query. The subject-verb relation is represented by a shared variable `$subject` in the `arg1` attribute of the `[event_expression]` tag and the `id` attribute of the `[phrase]` tag, the latter tag specified to contain the text region mentioning the OR-ed `[entity_name]` tags. By using the automatic query-creation mechanism, the users can enjoy the above complex functionalities without knowing the data structure in the annotated textbase or the syntax of the query language, while an expert user can devise and submit his/her own query directly to the search engine. In certain cases, one may wish to search for only those sentences with a specific syntactic structure, in addition to the semantic structure. One may also wish to combine a query specifying the semantic/syntactic structure with a condition on the role of the sentence in the abstract text (e.g., objective, method, or result). The flexibility of the extended region algebra allows a query that simultaneously represents several layers of structures, e.g., semantic structure and syntactic structure, by combining sub-queries for each layer with &-operator and tying them together by using variables

The search functionality offered by the MEDIE system is summarized as follows:

- Semantic search: users can search for a *semantic relations* described in MEDLINE. Since a biological fact can be represented by various natural language expressions, more accurate and high-recall search results can be obtained by the semantic search, compared to the traditional keyword-based search.

- Terminology normalization: the names of biological entities (protein, gene, disease) in the queried semantic relations are automatically treated as *biological concepts*, not merely the name given by the user.

- Event expression normalization: similarly to the terminology normalization, some verbs and nouns commonly used for expressing biological event (e.g, "activate" or "bind") are automatically treated as *biological relational concepts*. Since a single biological concept appears in many different forms in the literature, the normalization from expressions to concepts enhances the coverage of the retrieved results.

- Complex querying with additional conditions: users can narrow down the retrieved results by combining their semantic query with filters on bibliographic information (author name, journal title etc.) or on the information added by the NLP tools (e.g, whether the specified semantic relation is described as a conclusion of a paper).

The semantic search, the terminology normalization, and the event expression normalization are based on the annotations given by the parser, the named entity recognizer, and the event expression recognizer, respectively. Although users can switch on/off these functionalities freely, the default behavior of the UI system is to use a combination of the first three functionalities.

Chapter 5

Experiments

We developed a semantic search system and evaluated from two point of view, effectiveness of using NLP in search, that is, accuracy of search, and efficiency of our algorithm, which is expressed by search time. In the experiments for effectiveness, we evaluated two types of data. First, we evaluated exact match retrieval in our semantic search system with the queries and judgment of relevance created by us. Next, in order to evaluate the raking retrieval and show applicability of our semantic search system, we evaluated our system on the existing test collections in biomedical area, the test collection of TREC Genomic Track. In the experiments for the efficiency of our algorithm, we compared our system with existing XML Databases in the point of search time, and show the efficiency of our algorithm in detail. Target data of the experiments are the documents used in MEDIE system, which are MEDLINE abstracts annotated with the results of various NLP modules.

5.1 Evaluation for Effectiveness of Incorporating NLP into Search

In order to show the effectiveness by incorporating NLP into search in our semantic search system, we evaluated our system with two types of data, our original data and publically usable test collection. The target documents are the MEDLINE abstract in both data. In the former data, the queries are created by a biologist, and searched documents are judged whether the documents are relevant to the documents by the biologist. On the other hand, in the latter data, the queries and the judgment of relevant document are created by the participants of conference, Text REtrieval Conference (TREC).

5.1.1 Effectiveness of Specifying Parsing Results

We evaluated the effectiveness of specifying the predicate argument structure from the parser, which means the "subject-verb-object" relations by comparing the accuracy of "keyword search" with that of "semantic search," which executes queries specifying predicate argument structure. Table 5.1 shows the queries used in the experiments. These queries were created by biologists, and these queries express relational concepts that biologists want to find. "*something*"

Query No.	User input
1	*something* inhibit ERK2
2	*something* trigger diabetes
3	adiponectin increase *something*
4	macrophage induce *something*
5	*something* suppress MAP phosphorylation
6	*something* enhance p53 (negative)

Table 5.1: Queries for evaluating accuracy

Query	Keyword search			Semantic search		
	# results	time (first/all)	precision	# results	time (first/all)	precision
1	252	0.00s/ 1.5s	74/100 (74%)	143	0.01s/ 2.5s	96/100 (96%)
2	125	0.00s/ 1.8s	45/100 (45%)	27	0.02s/ 2.9s	23/ 27 (85%)
3	287	0.00s/ 1.5s	20/100 (20%)	30	0.05s/ 2.4s	23/ 30 (80%)
4	10698	0.00s/ 42.8s	14/100 (14%)	1559	0.01s/3014.5s	65/100 (65%)
5	87	0.04s/ 2.7s	34/ 87 (39%)	15	0.05s/ 4.2s	10/ 15 (67%)
6	1812	0.01s/ 7.6s	19/100 (19%)	84	0.20s/ 29.2s	73/ 84 (87%)

Table 5.2: Number of retrieved sentences, search time, and accuracy

indicates that any word can appear. The system output 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

```
(> [sentence]
   (& [word arg1=$subject arg2=$object base="verb"]
      (& (> [phrase cat="np" id=$subject]
            (> [word] subject))
         (> [phrase cat="np" id=$object]
            (> [word] object)))
```

in the same way as converting the queries in Table 5.1. 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 judgment. Table 5.2 shows the precision attained by semantic and keyword search. The results 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.

5.1.2 Effectiveness of Incorporating Various NLP Techniques in Search

Our search system is evaluated with respect to accuracy and speed on documents which are annotated with various types of NLP modules described in the previous chapter. Speed is indispensable for a real-time search system and accuracy the

Query No.	User input
1	*something* inhibit ERK2
2	adiponectin increase *something*
3	TNF activate IL6
4	dystrophin cause *disease*
5	macrophage induce *something*
6	*something* suppress MAP phosphorylation

Table 5.3: Queries used in the experiment

```
(> [sentence]
  (& (| [entity_name gena_id="GHS034354"]
        [entity_name gena_id="GMM092596"] )
     [event_expression
         event_type="Positive_regulation"] ) )

(> [sentence]
   (& [event_expression
          event_type="Positive_regulation"
          arg1="$subject"]
      (> [phrase id=$subject]
         (| [entity_name gena_id="GHS034354"]
            [entity_name gena_id="GMM092596"] ) ) ) )
```

Figure 5.1: Queries of extended region algebra for Query 2-TE (top: keyword search, bottom: semantic search)

need for more accurate search is the motivation of semantic search. That is, our motivation for employing semantic search was to provide a device for the accurate identification of relational concepts. In particular, high accuracy is desired in text search from huge texts because users want to extract only relevance informations.

We varied two parameters in this experiments. One is specification of semantic relations, and the other is the use of ontological identification by term normalization. The effect by specifying semantic relations is evaluated by comparing the results of search without specification of semantic relations ("Keyword Search") and with the specification ("Semantic Search"). The former is a traditional search technique, in which the query is given to the framework as a set of keywords and the results are searched by matching of the keywords in the query and in documents. The latter is a new feature of our system, in which the query contains not only a set of keywords but the relations among the keywords, and the results are searched using the relations with which the documents are annotated. The effect of using ontological identification is evaluated by comparing the results of search with queries using keywords themselves and queries in which the keywords are replaced with corresponding ontological identifiers in both keyword search and semantic search. When we use the term ontology, the nominal keywords in queries are replaced with ontological identifiers. When we use the event

Query	Keyword search		
No.	# ans.	time (first/all)	precision
1-1	338	0.01/ 3.0	75/100 (75%)
1-2	576	0.02/ 3.8	60/100 (60%)
1-3	4965	0.01/ 8.6	16/100 (16%)
2-1	1527	0.01/ 3.6	33/100 (33%)
2-2	1355	0.01/ 3.6	20/100 (20%)
2-3	2226	0.01/ 4.0	31/100 (31%)
3-1	4	0.89/ 2.3	0/ 4 (0%)
3-2	485	0.01/ 8.2	6/100 (6%)
3-3	9376	0.11/ 39.9	10/100 (10%)
4-1	460	0.02/ 2.8	90/100 (90%)
4-2	714	0.03/ 4.6	61/100 (61%)
4-3	4839	0.19/ 23.4	12/100 (12%)
5-1	4070	0.05/ 20.9	12/100 (12%)
5-3	24492	0.40/185.1	17/100 (17%)
6-1	76	0.05/ 5.7	45/ 76 (59%)
6-3	1010	0.04/ 8.9	48/100 (48%)

Table 5.4: Number of retrieved sentences, retrieval time, and accuracy for Keyword Search

expression ontology, verbal keywords in queries are replaced with event types.

Table 5.1.2 lists the queries used in the experiments. Words in italic indicates a class of words: "*something*" indicates that any words can appear, and "*disease*" indicates that any disease names can appear. These queries were selected by a biologist, and express typical relational concepts that a biologist may wish to find.

For example, Query 3 attempts to search sentences that mention the protein-protein interaction "TNF activates IL6." This query is converted into queries of the region algebra given in Figure 5.1. The top query is for keyword search and the bottom query is for semantic search. The query for keyword search only specifies the appearance of the three keywords, and the keywords are translated into the ontological identifiers, "GHS034354," "GMM092596" (for "adiponectin") and "Positive_regulation" (for "increase"). The query for semantic search specifies the appearance of the three keywords and the relation of the keywords is indicated by the variables "$subject," which means "GHS034354" or "GMM092596" is the subject of "Positive_regulation." The class of words, such as *disease* in Table 5.1.2, is translated into the class name in the similar way to the ontological identifier, like `[entity_name type="disease"]`.

Table 5.1.2 and Table 5.1.2 shows the results of the experiments. The postfixes of query numbers denote whether ontological identifiers are used. X-1 used no ontologies, X-2 used only the term ontology and X-3 used both the term and event expression ontology. Comparison of the results of X-1 and X-2 shows the effect of using the term ontology, and comparison of the results of X-2 and X-

Query	Semantic search			
No.	# ans.	time (first/all)	precision	relative recall
1-1	189	0.05/ 3.0	98/100 (98%)	61/75 (81%)
1-2	290	0.09/ 6.4	86/100 (86%)	40/60 (66%)
1-3	1404	0.01/ 15.0	56/100 (56%)	10/16 (62%)
2-1	23	0.09/ 12.8	18/ 23 (78%)	2/33 (6%)
2-2	38	0.16/ 11.0	24/ 38 (63%)	1/20 (5%)
2-3	295	0.01/ 16.6	49/100 (49%)	6/31 (19%)
3-1	0	2.79/ 2.9	0/ 0 (-)	0/ 0 (-)
3-2	7	7.82/ 29.1	7/ 7 (100%)	1/ 6 (16%)
3-3	311	1.79/110.9	78/100 (78%)	5/10 (50%)
4-1	78	0.13/ 11.6	78/ 78 (100%)	16/90 (17%)
4-2	118	0.22/ 10.1	96/100 (96%)	22/61 (36%)
4-3	233	0.31/ 34.0	69/100 (69%)	4/12 (33%)
5-1	611	0.15/ 13.2	30/100 (30%)	4/12 (33%)
5-3	1691	0.36/288.0	16/100 (16%)	2/17 (11%)
6-1	34	0.09/ 14.6	30/ 34 (88%)	29/45 (64%)
6-3	389	0.06/ 16.7	76/100 (76%)	30/48 (62%)

Table 5.5: Number of retrieved sentences, retrieval time, and accuracy for Semantic Search

3 shows the effect of using the event expression ontology. Since no ontological expansion for terms for the query 5 and 6, the query 5-2 and 6-2 does not exist.

Accuracy is calculated by judgements of biologists. At most 100 results were searched for each query, and the results of 6 queries, varied with keyword search or semantic search and whether terms and event ontologies are used to expand the queries, were shuffled to hide which query searches the results for biologists who judged the results. The criterion of the judgements is whether the intended information in the query are expressed in the results, that is, whether the results contains the relation expressed in the query. The modality of the sentences was not distinguished in the judgements, that is, the negation or certainty of the relation is ignored. These criteria may be disadvantageous for the semantic search because its ability to recognize the participants of relational concepts is not evaluated.

We evaluated our framework with respect to accuracy with two metrics, precision and relative recall calculated from the judgements of a biologist. Precision is calculated as ratio of the results judged relevant to the query in all results. Relative recall is fraction of the correct search results returned by the semantic search which are also found by the keyword search, against all the correct result returned by the keyword search: $|S_{key} \cup S_{sem}|/|S_{key}|$, where S_{key} is a set of correct results searched in keyword search and S_{sem} is a set of correct results searched in semantic search. Although relative recall is no the true recall, it suffices to compare between keyword search and semantic search.

The results show that the semantic search exhibited impressive improvements in precision, from 37% to 72%, 35 percentage points up on average, by the specification of semantic relations. The relative recalls are varied by queries, from 10 % to 70%, and the average is 38%. This results show that the precision was increased significantly by specification of semantic relations with some recall decreases by the specification. This indicates that specification of semantic relation is effective for searching relational concepts precisely.

Comparison among the results of queries X-1, X-2 and X-3 shows that the ontology expansion exhibited impressive improvements in recall, that is, the expansion increased the number of search results with little deterioration in the precision. For example, the keyword search without ontology expansion in Query 3 could not search correct results. Ontology expansion enable us to search more results corresponds to the search aims by considering synonyms of "TNF" and "IL6" and the relation of "activate" described by other expressions. Time to search the first or all results shows the system is sufficiently fast as a human UI.

5.1.3 Evaluation on Public Test Collection

We evaluated our semantic search on the test collection constructed in TREC Genomics Track, in order to show how our semantic search system can be applied into general search task, in which the queries are written in natural language and the search results are the documents relevant to the queries. We employed the test collection in TREC Genomis Track for the experiments.

TREC Genomics Track

TREC Genomics Track, which ran from 2003 to 2007, is one of the tracks in Text Retrieval Conference(TREC), which providing the infrastructure for large-scale of text retrieval methodologies. The target area of the track is biomedical area.

We evaluated our framework on the test collection constructed from Ad Hoc Retrieval Task on TREC 2004 Genomic Track [26] and TREC 2005 Genomic Track [25]. The target document collection for the task is a 10-year subset of the MEDLINE, from 1994 to 2003, which contains 4,591,008 abstracts in biomedical area. The test collection contains 50 topics, which indicate "queries" in both TREC 2004 and 2005, which derived by biologists as real information needs. In TREC 2004, the topics consists 3 parts of text written in natural sentences, *title* (abbreviated statement of information need), *information need* (full statement of information need), and *context* (background information to place information need in context). Figure 5.6 and 5.7 show a part of topics for TREC 2004. In TREC 2005, the topics are categorized into 5 templates:

1. Articles describing standard methods or protocols for doing some sort of experiment or procedure.

2. Articles describing the role(s) of a gene invoked in a disease.

Topic No.	Type	Query sentences
002	T	Generating transgenic mice
	N	Find protocols for generating transgenic mice.
	C	Determine protocols to generate transgenic mice having a single copy of the gene of interest at a specific location.
003	T	Time course for gene expression in mouse kidney
	N	What is the time course of gene expression in the murine developing kidney?
	C	Relevant articles describe genes involved in kidney development.
004	T	Gene expression profiles for kidney in mice
	N	What mouse genes are specific to the kidney?
	C	What genes are expressed only in the mouse kidney and not in other tissues?
005	T	Protocols for isolating cell nuclei
	N	Articles are relevant if they describe methods for subcellular fractionation of nuclei.
	C	Laboratory preparations can be enriched for certain kinds of proteins if the cellular compartment in which they reside is purified away from the rest of the cell contents.
007	T	DNA repair and oxidative stress
	N	Find correlation between DNA repair pathways and oxidative stress.
	C	Researcher is interested in how oxidative stress effects DNA repair.
011	T	Carcinogenesis and hairless mice
	N	Find articles regarding carcinogenesis induced in hairless mice.
	C	Researching genes and proteins (pathways) common to DNA repair, oxidative diseases, skin-carcinogenesis, and UV-carcinogenesis.
012	T	Genes regulated by Smad4
	N	Find articles describing genes that are regulated by the signal transducing molecule Smad4.
	C	Project is to characterize Smad4 knockout mouse in skin (specifically skin) to establish signaling network. Identify all Smad4 targets to compare gene expression patterns of the knockout mouse to the normal mouse.
014	T	Expression or Regulation of TGFB in HNSCC cancers
	N	Documents regarding TGFB expression or regulation in HNSCC cancers.
	C	The laboratory wants to identify components of the the TGFB signaling pathway in HNSCC, and determine new targets to study HNSCC.
021	T	Role of p63 and p73 in relation to DNA damage
	N	Do p63 and p73 cause cell cycle arrest or apoptosis related to DNA damage?
	C	DNA damage may cause cell cycle arrest or apoptosis. p63 and p73 may play a role in mediating these sequelae of DNA damage.
025	T	Cause of scleroderma
	N	Identify studies that include genome-wide scans and microarray analysis in the investigation of scleroderma.
	C	New information about experiments and genes involved in scleroderma.
030	T	Regulatory targets of the Nkx gene family members
	N	Documents identifying genes regulated by Nkx gene family members.
	C	The laboratory needs markers to follow Nkx family-member expression and activity.

Table 5.6: Topics in TREC Genomic Track 2004, Title (T), Need (N) and Context (C)

Topic No.	Type	Query sentences
031	T	TOR signaling in neurofibromatosis
	N	Reports that provide possible links between neurofibromatosis and TOR signaling.
	C	TOR is a serine-threonine kinase in a pathway involved in the control of cell growth and proliferation, and it is the target of the signaling inhibitor rapamycin.
033	T	Mice, mutant strains, and Histoplasmosis
	N	Identify research on mutant mouse strains and factors which increase susceptibility to infection by Histoplasma capsulatum.
	C	The ultimate goal of this initial research study, is to identify mouse genes that will influence the outcome of blood borne pathogen infections.
034	T	Gene products of Cryptococcus important to fungal survival
	N	Articles reporting experiments allowing annotation of gene products of Cryptococcus.
	C	Information needed to contribute to the development of a standardized annotated database of Cryptococcus neoformans genome.
035	T	WD40 repeat-containing proteins
	N	What is the function of proteins containing WD40 repeats?
	C	Need to understand the variety of functions that involve this domain.
038	T	Risk factors for stroke
	N	Information concerning genetic loci that are associated with increased risk of stroke, such as apolipoprotein E4 or factor V mutations.
	C	Candidate gene testing within a large Scottish case-control study of genetic risk factors for stroke. Future research includes investigations into other ethnically distinct populations.
039	T	Hypertension
	N	Identify genes as potential genetic risk factors candidates for causing hypertension.
	C	A relevant document is one which discusses genes that could be considered as candidates to test in a randomized controlled trial which studies the genetic risk factors for stroke.
040	T	Antigens expressed by lung epithelial cells
	N	To identify the antigens expressed by lung epithelial cells and the antibodies available.
	C	Information gathering to design assays to determine the nature of donor cells in tissues of chimaeric animals.
042	T	Genes altered by chromosome translocations
	N	What genes show altered behavior due to chromosomal rearrangements?
	C	Information is required on the disruption of functions from genomic DNA rearrangements.
050	T	Low temperature protein expression in E. coli
	N	Find research on improving protein expressions at low temperature in Escherichia coli bacteria.
	C	The researcher is not satisfied with the yield of expressing a protein in E. coli when grown at low temperature and is searching for a better solution. The researcher is willing to try a different organism and/or method.

Table 5.7: Topics in TREC Genomic Track 2004 (cont.)

Query ID	Gene	Disease
110	Interferon-beta	Multiple Sclerosis
111	PRNP	Mad Cow Disease
112	IDE gene	Alzheimer's Disease
113	MMS2	Cancer
114	APC(adenomatous polyposis coli)	Colon Cancer
115	Nurr-77	Perkinson's Disease
116	Insulin receptor gene	Cancer
117	Apolipoprotein E (ApoE)	Alzheimer's Disease
118	Transforming growth factor-beta 1 (TGF-beta1)	Cerebral Amyloid Angiopathy (CAA)
119	GSTM1	Breast Cancer

Table 5.8: Queries for Topic Type 2

3. Articles describing the role of a gene in a specific biological process.
4. Articles describing interactions (e.g., promote, suppress, inhibit, etc.) between two or more genes in the function of an organ or in a disease.
5. Articles describing one or more mutations of a given gene and its biological impact or role.

For each topics, 10 sets of specific words for underlined words in template are given. Figure 5.8 shows the list of specific words for Query Type 2. For example, for the topic ID 110, "Interferon-beta" and "Multiple Sclerosis" are given as a gene and a disease respectively. So the query becomes "Articles describing the role(s) of a Interferon-beta invoked in a Multiple Sclerosis."

For each topic, around 1,000 documents are judged whether the document is relevant to the topic or not. The relevance has three levels, *definitely relevance*, *possibly relevant* and *not relevant*. These relevance are annotated by the participants of the conference.

Query Transformation

Because the topics in the TREC Data are written as natural language sentences, we need to transform the topics into a set of queries in the form of region algebra in order to evaluate our system on the TREC Genomics Track test collection. We transformed these topics by hands, based on the results of NLP modules. We first process the sentences by the same NLP modules used in the preprocess for the target documents, and we constructed queries in the region algebra manually. The results of NLP modules mainly used to create queries are results from the syntactic/semantic parser and the named entity recognizer. The queries are created with focusing mainly on predicate-argument structure. The most basic type of queries are words themselves and the results of named entity recognizer, and the more structured queries contains on predicate-argument structure.

Filtering query

```
(> [MedlineCitation]
   (& [entity_name facta_id="UMLS:C0002395"]
      (| [entity_name uniprot_id="Q24K02"]
         [entity_name uniprot_id="P14735"])))
```

Scoring queries
Query 1

```
[entity_name facta_id="UMLS:C0002395"]
```

Query 2

```
(| [entity_name uniprot_id="Q24K02"]
   [entity_name uniprot_id="P14735"])
```

Query 3

```
(> [setence]
   (& [GDA entity1=$gene entity2=$disease]
      (& [entity_name id=$disease
                      facta_id="UMLS:C0002395"]
         (| [entity_name id=$gene uniprot_id="Q24K02"]
            [entity_name id=$gene uniprot_id="P14735"]))))
```

Figure 5.2: Queries for Topic 112

Evaluation Measure

We evaluated the result of ranking retrieval with MAP (Mean Average Precision), which is used as a standard evaluation measure in the TREC community. The average precision is calculated in the following formula.

$$AP(q) = \frac{1}{n}\sum_{k=1}^{n}\frac{k}{r_k}$$

where q is a query, n is the number of relevant documents for the query q, This value expresses the average of precision at the rank where the relevant documents appears. The MAP value takes an average of the average precision, that is,

$$MAP(Q) = \frac{1}{|Q|}\sum_{i=1}^{|Q|}AP(q_i)$$

where Q is a set of queries $q_1, q_2, ..., q_{|Q|}$.

Experiments on TREC 2005 Test Collection

First, we evaluated our system in topics of TREC 2005. Figure 5.2 shows an example of queries for Topic 112, 'Articles describing the role of 'IDE gene' involved in 'Alzheimer's Disease'. " Since the articles relevant to the topic should mention to 'IDE gene' and 'Alzheimer's Disease,' we filtered out the documents not including them by the Filtering Query in Figure 5.2. Thus the Filtering

Query for this topic type expresses the condition "Documents contains both the target gene and the target disease." Note that in the Filtering Query for topic 112, 'IDE gene' and 'Alzheimer's Disease' are converted to the query on annotations `[entity_name]` with the UniProt IDs for 'IDE gene' and UMLS IDs for 'Alzheimer's Disease,' 'Q24K02' or 'P14735' and 'C0002395' respectively by using the dictionary constructed in advance. We used three queries to calculate a score for this topic. First and second queries are the expansion of keywords 'IDE gene' and 'Alzheimer's Disease' respectively. The last query expresses the gene-disease associations (GDA) between 'IDE gene' and 'Alzheimer's Disease,' which are recognized in advance and annotated to target documents as `[GDA]` annotations. In the score calculation, Scoring Query 1, 2 and 3 are evaluated in each document searched with Filtering Query, and the number of searched regions for the queries are counted. The number of searched regions expresses the number of appearances of synonymous expressions for 'IDE gene' and 'Alzheimer's Disease' for Scoring Query 1 and 2, and expresses the number of sentences including the expression of gene-disease association between 'IDE gene' and 'Alzheimer's Disease' for Scoring Query 3. The score for each document are calculated by the formula,

$$S(D, q_1, ..., q_n) = \sum_{i=1}^{n} RIDF(q_i) \cdot \frac{rf_{q_i} \cdot (k_1 + 1)}{rf_{q_i} + k_1 \cdot (1 - b + b \cdot \frac{|D|}{DL_{ave}})}$$

where rf_{q_i} is the number of regions matching to the query q_i in the document D, k_1 and b are parameters, (we used $k_1 = 2.0$ and $b = 0.75$ in the experiments), $|D|$ is the length of the document D, DL_{ave} is the average length of documents in the collection and $RIDF(q_i)$ is the relative IDF value for the query q_i, which is defined as

$$RIDF(q_i) = \log \frac{(df_{(\&q_{i_1} \cdots q_{i_n})} - df_{q_i}) + 0.5}{df_{q_i} + 0.5}.$$

where df_{q_i} is the number of documents which contain at least one region matching the query q_i in the document set, and $q_{i_1}...q_{i_n}$ are the sub-queries of q_i in Scoring Queries. The queries for other topics of topic type 2 were constructed by the same way. with the queries for Topic 112 in the above examples except for the target gene and disease expanded with the the UniProt IDs and UMLS IDs. In this type of topics, we narrowed down the documents by the Filtering Query expressing the condition and we used 3 Scoring Queries, 1) a query for the target gene, 2) a query for the target disease and 3) a query for gene-disease association.

For other topic types, we used queries described in the following: For topic type 1, "Articles describing standard methods or protocols," we used subject-verb-object relations from the parsing results and keywords expansions with NER result when the topic include a gene name or a disease name. Figure 5.3 shows an example of queries for Topic 107, "Articles describing about 'normalization procedures that are used for microarray data'. " Scoring Queries 1, 2 and 3 expresses

Filtering query

```
(> [MedlineCitation]
   (& (| [word base="normalization"]
         [word base="normalize"])
      [word base="microarray"])))
```

Scoring queries
Query 1

```
(- [word base="microarray"] [word base="datum"])
```

Query 2

```
(| [word base="normalization"] [word base="normalize"])
```

Query 3

```
(| [word base="method"]
   (| [word base="procedure"]
      (| [word base="technique"]
         (| [word base="protocol"] [word base="process"]))))
```

Query 4

```
(> [sentence]
   (& (| [word cat="V" base="normalize" arg2=$data]
         (& (> [phrase id=$n]
               [word base="normalization"])
            [word cat="P" base="of" arg1=$n arg2=$data]))
      (> [phrase id=$data cat="NP"]
         (& [word base="microarray"] [word base="datum"]))))
```

Figure 5.3: Queries for Topic 107

the keywords, and Query 4 specifies the expression such as 'normalize microarray data' or 'normalization of microarray data' using the parsing results. The relation between 'normalize' or 'normalization' and 'microarray data' is expressed with the variable `$data`. For each topic in topic type 3, "Articles describing the role of gene in a specific biological process," because currently there is no annotations in the database that directly expressed the notion of 'biological process,' we used various types of annotations such as the results of parsing, NER, GDA and event recognizer to specify a biological process. For example, Figure 5.4 shows an example of queries for Topic 120, "Articles describing the role of 'NM23' in a 'tumor progression'." Filtering Query show the condition that "Documents contains both 'NM23' and 'tumor'." The relation 'NM23' and 'tumor' are expressed in two types of annotations, `[GDA]` (Scoring Query 3) and `[event_expression]` (Scoring Query 4). For topic type 4, " Articles describing interactions between two genes in the function of an organ or in a disease," we constructed a query expressing the interactions between the target genes with results of event recognizer with additional keywords of 'function of organ' and 'disease' expanded with NER

Filtering query

```
(> [MedlineCitation]
   (& (| [entity_name uniprot_id="P15531"]
         [entity_name uniprot_id="P15532"])
      (| [word base="tumor"]
         [entity_name facta_id="UMLS:C0006826"])
```

Scoring queries

Query 1

```
(| [entity_name uniprot_id="P15531"]
   [entity_name uniprot_id="P15532"])
```

Query 2

```
(| [word base="tumor"]
   [entity_name facta_id="UMLS:C0006826"])
```

Query 3

```
(> [sentence]
   (& [GDA entity1=$gene entity2=$disease]
      (& [entity_name id=$disease
                      facta_id="UMLS:C0006826"]
   (| [entity_name id=$gene uniprot_id="P15531"]
      [entity_name id=$gene uniprot_id="P15532"]))))
```

Query 4

```
(> [sentence]
   (& [event_expression event_type="Positive_regulation"
                        arg1=$1 arg2=$2]
      (& (> [phrase id=$1]
            (| [entity_name uniprot_id="P15531"]
               [entity_name uniprot_id="P15532"]))
         (> [phrase id=$2]
            (| [word base="tumor"]
               [entity_name facta_id="UMLS:C0006826"])))))
```

Figure 5.4: Queries for Topic 120

results. For topic type 5, " Articles describing mutations of a given gene and its biological impact," we used the results of event recognizer expressing 'mutation', and added expanded keywords in 'biological impact.'

Table 5.9 shows the mean average precision (MAP), precision in top 10 results and recall averaged over each topic type. MAP and precision are calculated ignoring the documents not judged in the test collection. Recall is calculated in documents searched by Filtering Query. When the number of articles searched by Filtering Query is less than 10, the total precision is used as P10. The results show that MAP of the proposed framework is higher than average MAP of the runs in TREC 2005 Genomics Track in topic type 1, 2 and 3, but lower in topic

	Topic type 1	Topic type 2	Topic type 3	Topic type 4	Topic type 5
MAP	0.190	0.301	0.295	0.138	0.179
P10	0.480	0.530	0.470	0.400	0.300
Recall	0.405	0.434	0.418	0.313	0.323
# searched documents	1287	291	738	467	463
# judged documents	145	162	165	83	147
Judged percentage	0.445	0.834	0.510	0.748	0.630
Ave. MAP in TREC 2005	0.160	0.236	0.202	0.193	0.192
Ave. P10 in TREC 2005	0.368	0.428	0.377	0.295	0.315

Table 5.9: Average MAP, precision and recall for types of topics

type 4 and 5. The precision in top 10 results is significantly higher than that of runs in TREC 2005. One of reasons for low MAP is the Filtering Query. As shown in Table 5.9, the recall of the Filtering Queries is lower than 0.5 for all topic types, that is, more than half of relevant documents are filtered out with the Filtering Queries. For example, in Topic 111 "Articles describing the role of 'PRNP' involved in 'Mad Cow Disease', " the recall for search with Filtering Query "Documents containing both 'PRNP' and 'Mad Cow Disease' " is only 0.163 despite a query expansion with the result of named entity recognition. Some of the relevant results contain other 'prion diseases' such as "Creutzfeldt-Jakob disease," which is similar disease with "Mad Cow Disease," or does not contain the corresponding disease name. Our current system have annotations which enables the system to regard different expressions of a disease or a gene as the same object, but does not contain annotations for 'knowledge' such that these different diseases can be considered as the same disease. In order to satisfy both of the search speed and the accuracy, improvement of Filtering Query with addition of other types of annotations are required to search more relevant documents in the filtering step in order to improve the accuracy.

Table 5.10 shows MAP for topics of topic type 2 with different types of queries used in scoring. The queries Q_{gene}, $Q_{disease}$ and Q_{GDA} correspond to Scoring Query 1, 2 and 3 in Figure 5.2 respectively. This results show that calculating score considering not only the expanded keywords but the structured relation of keywords, GDA in this case, is effective to improve the accuracy of retrieval. However, the MAP scores of the results by using all Scoring Queries are lower than that in scoring without GDA query in some queries. We suspect this is caused by the fact that the implicit assumption made in BM25 scoring, namely the independence of the keyword occurrence, does not hold in our case. For example, when a document includes a gene name and a disease name in a topic definition, the probability that the document also includes a [GDA] annotation is generally higher than one under the independence assumption.

Q_{gene}	√	√	√		√		
$Q_{disease}$	√	√		√		√	
Q_{GDA}	√		√	√			√
110	0.008	0.012	0.008	0.008	0.015	0.015	0.008
111	0.031	0.034	0.032	0.033	0.033	0.031	0.032
112	0.388	0.278	0.356	0.388	0.139	0.248	0.373
113	0.460	0.460	0.460	0.375	0.466	0.375	0.416
114	0.295	0.292	0.292	0.290	0.310	0.288	0.310
115	0.200	0.200	0.200	0.200	0.200	0.200	0.200
116	0.150	0.158	0.158	0.145	0.121	0.155	0.112
117	0.695	0.711	0.711	0.669	0.701	0.655	0.648
118	0.000	0.000	0.000	0.000	0.000	0.000	0.009
119	0.778	0.775	0.775	0.782	0.719	0.779	0.778
AVR	0.301	0.292	0.297	0.297	0.270	0.275	0.287

Table 5.10: MAP for topics of topic type 2 with different types of Scoring Queries

Experiments on TREC 2004 Test Collection

In order to capture the effectiveness of specification of NLP annotations in detail, we conducted the experiments of the TREC 2004 test collection. We selected 20 topics from the whole 50 topics in TREC 2004 test collection by the criteria that specification of annotations seems to be effective for accuracy, that is, the relation which can be captured by using the parsing result, such as the subject-verb-object relation or modification relation of nouns, is written in the topic sentences directly. For example, in the topic 002, the relation 'the object of the verb "generate" is "mice" ' and ' "transgenic" modify the noun "mice",' is directly written in TITLE and NEED field.

In order to see the effectiveness of specifying each type of annotations, we compared five types of query set, *All*, *Keyword + NE*, *Keyword*, *Structure* and *Structure+NE.* The query set *All* is a base set of queries created by hand using the topic and the results from NLP modules. The query set *Keyword* is a set of basic form of keywords. The query set *NE* is a set of results from the named entity recognizer. *Keyword + NE* uses both the queries in the *Keyword* query set and the *NE* query set. The query set *Structure* is a set of queries containing the structure, which is mainly the predicate-argument structure from the syntactic/semantic parser. *Structure + NE* uses both the queries in the *Structure* query set and the *NE* query set. In this definition, the query set *All* is the same with the query set *Keyword + NE + Structure.*

Figure 5.5 shows an example of queries for the topic 4. The original sentences in topic 4 are followings:

TITLE Gene expression profiles for kidney in mice

NEED What mouse genes are specific to the kidney?

Query 1
`[word base="gene"]` Query 2
`[word base="expression"]` Query
3
`[word base="profile"]` Query 4
`[word base="kidney"]` Query 5
`[word base="mouse"]` Query 6

```
(| [entity_name facta_id="UMLS:C0496892"]
   [entity_name facta_id="UMLS:C0496927"]
   [entity_name facta_id="UMLS:C0812426"])
```

Query 7

```
(| [entity_name type="species" ncbi_id="10095"]
   [entity_name type="species" ncbi_id="10090"])
```

Query 8

```
(> [sentence]
   (& [word base="gene" arg1=$a1]
      [word base="expression" arg1=$a2]
      (> [phrase id=$a1]
         (> [phrase id=$a2] [word base="profile"]))))
```

Query 9

```
(> [sentence]
   (& [word base="in" arg2=$arg2 arg1=$arg1]
      (> [phrase id=$arg2] [word base="mouse"])
      (> [phrase id=$arg1]
         (| [word base="kidney"]
            [entity_name facta_id="UMLS:C0496892"]
            [entity_name facta_id="UMLS:C0496927"]
            [entity_name facta_id="UMLS:C0812426"]))))
```

Query 10

```
(> [sentence]
   (& [word base="for" arg2=$arg2 arg1=$arg1]
      (> [phrase id=$arg2]
         (| [word base="kidney"]
            [entity_name facta_id="UMLS:C0496892"]
            [entity_name facta_id="UMLS:C0496927"]
            [entity_name facta_id="UMLS:C0812426"]))
      [word base="gene" arg1=$a1]
      [word base="expression" arg1=$a2]
      (> [phrase id=$arg1]
         (> [phrase id=$a1]
            (> [phrase id=$a2] [word base="profile"])))))
```

Query 11

```
(> [sentence] (& [word base="mouse" arg1=$a] (
                 > [phrase id=$a] [word base="gene"])))
```

Figure 5.5: Queries for Topic 004

Topic No.	All	Keyword +NE	Keyword	Structure	Structure +NE
002	0.1584	0.155	0.1718	0.1463	0.1478
003	0.1884	0.1817	0.123	0.1717	0.2521
004	0.0448	0.044	0.0566	0.0251	0.0409
005	0.1883	0.1761	0.1761	0.073	0.073
007	0.4023	0.4117	0.3461	0.243	0.2662
011	0.4603	0.4579	0.2906	0.3446	0.459
012	0.7647	0.7637	0.3002	0.349	0.8334
014	0.0283	0.0284	0.0232	0.0222	0.0281
021	0.5409	0.5305	0.1083	0.2238	0.5632
025	0.0411	0.0421	0.0256	0.0279	0.0407
030	0.156	0.1591	0.1591	0.1526	0.1526
031	0.2216	0.2193	0.2475	0.1798	0.1768
033	0.3824	0.385	0.0603	0.205	0.3219
034	0.1066	0.1083	0.0665	0.0398	0.0468
035	0.7346	0.7337	0.7258	0.5479	0.7249
038	0.3593	0.3712	0.4563	0.3386	0.3059
039	0.5987	0.5902	0.4146	0.2956	0.3634
040	0.3769	0.3957	0.3957	0.2303	0.2303
042	0.7487	0.7513	0.7336	0.6849	0.7059
050	0.3399	0.325	0.335	0.488	0.4685
AVR	0.3421	0.3415	0.2608	0.2395	0.3101

Table 5.11: MAP for topics in TREC 2004

CONTEXT What genes are expressed only in the mouse kidney and not in other tissues?

Queries from the Query 1 to Query 5 are keyword-based queries. These keywords are selected from the words in original sentences by using the stopword list. The stopword list consist of the function words, such as prepositions, auxiliary verbs and pronouns, which are regarded as stopwords in general, and words used to describe search requests, such as "search," "find," "relevant," "describe," "article," and "document." Query 6 and Query 7 is a query created from the results from named entity recognizer. Query 6 is a expansion of "kidney" by IDs in UMLS, and Query 7 is a expansion of "mouse." The other queries are created by using the result from the parser. Query 8 is a query searching sentences containing "gene expression profile." Since the query expresses the modification relation that "expression" modifies "profile" and "gene" modifies "profile" modified by "expression", the search results by this query is not necessarily exact string "gene expression profile", but other words can be appear between the words if the modification relation in the query is retained. Query 9 expresses sentences containing the expression"kidney in mouse." Synonyms of "kidney" and "mouse" can be retrieved by this query since these words are expanded with the results of named entity recognizer. The results also are not necessarily exact string “kidney in mouse” as the same with the case of Query 8. Query 10 searches sentences containing “gene expression profiles for kidney.” “kidney” is expanded with synonyms, and the result also are not necessarily exact string as the same with the above queries. Query 11 expresses sentences containing the relation “mouse” modifies “gene”, the simplest result of this queries is the sentence containing the string “mouse gene.” In the case of these queries, five types of query set in the experiments is defined as follows:

All 1,2,3,4,5,6,7,8,9,10,11

Keyword 1,2,3,4,5

Keyword+NE 1,2,3,4,5,6,7

Structure 8,9,10,11

Structure+NE 6,7,8,9,10,11

Table 5.11 shows the result of experiments, MAP value for each selected topic, on the TREC 2004 test collection. The results show that specification of results from NLP modules in retrieval is effective, especially the result from named entity recognizer is effective. However, the accuracy decreased by incorporating annotations in ranking retrieval in some queries.

5.2 Evaluation of Search Algorithm

In the previous section, we evaluated our system as the whole semantic search system, that is, in the point of improvement of accuracy by using NLP technologies. In this section, we evaluated in the point of efficiency of algorithm in our search system, that is, the speed of search by using our algorithm. We compared our system with the existing XML Databases, and the effectiveness of our algorithm for searching regions for the query containing variables.

5.2.1 Search Speed Compared with XML Database

We evaluated our framework in point of efficiency. The efficiency search time is compared with that of other XML databases, MonetDB [7] and eXist [46]. Since these XML databases cannot be applied to the data used in the MEDIE system, which is not a well-formed XML, that is, the annotations from NLP modules intersect one another in the documents, we used the documents annotated with only the paring results in the experiments.

Table 5.12 show the search times for three queries in our system, and existing XML databases, MonetDB/XQuery [7] and eXists [46]. *Our(Cache)* denotes that search time of our system when the data concerning to the query is cached on memory. We indexed 150,000 MEDLINE articles in all systems for this experiment. We converted the queries into a form of region algebra like

	Search time in each # abstracts				
System	30,000	60,000	90,000	120,000	150,000
Query subject:p53 verb:activate(b)					
Our(Cache)	0.018s	0.036s	0.047s	0.065s	0.082s
Our	2.085s	4.233s	6.680s	8.978s	10.711s
MonetDB	6.409s	21.728s	32.334s	62.431s	83.044s
eXist	10.996s	24.841s	39.398s	53.137s	68.417s
Query verb:activate(b) object:p53					
Our(Cache)	0.016s	0.030s	0.042s	0.059s	0.073s
Our	2.164s	4.182s	6.561s	8.903s	10.449s
MonetDB	6.431s	20.354s	30.601s	60.011s	83.349s
eXist	12.729s	25.537s	39.118s	52.758s	68.989s
Query verb:cause(b) object:cancer					
Our(Cache)	0.058s	0.114s	0.164s	0.219s	0.278s
Our	8.175s	11.794s	21.087s	27.083s	40.152s
MonetDB	7.871s	26.063s	36.396s	56.050s	90.315s
eXist	17.171s	35.631s	61.673s	84.562s	119.122s

Table 5.12: Search time of three systems

Query	XQuery
verb:activate(b), subject:p53	for $s in //sentence[.//word = “p53”] [.//word[@base=“activate”]], $w in $s//word[@base=“activate”], $p in $s//phrase[.//word = “p53”] [@id=$w/@arg1] return $s
verb=“cause” (b), object=“cancer”	for $s in //sentence[.//word = “cancer”] [.//word[@base=“cause”]], $w in $s//word[@base=“cause”], $p in $s//phrase[.//word = “cancer”][@id=$w/@arg2] return $s

Table 5.13: A Query converted to XQuery form

```
(> [sentence]
   (& [word arg1=$subject arg2=$object base="verb"]
      (& (> [phrase cat="np" id=$subject]
            (> [word] subject))
         (> [phrase cat="np" id=$object]
            (> [word] object)))
```

for the experiments in our framework, and also converted the query into XQuery form as shown in Table 5.13 for two XML Databases. (In the experiments on eXist, we used the “contain” function to express that tag regions contain a word.) The results showed that the search time of all systems was nearly linear to the number of abstracts, and that our system could search documents in a shorter

Query	descending	ascending
verb:cause(b), object:cancer	6.418s	6.182s
verb:cause(b), object:cancer +modifier:not	16.363s	11.433s
subject:stress, object:cancer	10.594s	8.426s
verb:interact with(b), object:CD4	2.494s	1.859s

Table 5.14: Search time by changing order of queries

Topic No.	Our	Baseline
002	48.675s	120.241s
003	25.317s	151.619s
004	24.952s	116.457s
005	24.350s	134.712s
007	18.551s	101.974s
011	20.087s	103.500s
012	18.276s	48.644s
014	20.021s	64.642s
016	23.050s	106.542s
021	14.998s	81.388s

Topic No.	Our	Baseline
025	22.079s	93.054s
030	24.968s	153.753s
031	17.556s	69.831s
034	15.545s	77.966s
035	19.021s	71.394s
038	23.873s	115.908s
039	31.125s	37.294s
040	27.714s	142.869s
042	21.630s	116.678s
050	21.990s	142.704s

Table 5.15: Search Time by using Our Algorithm and Baseline Algorithm Searching All Combinations

amount of time than existing XML databases in our tasks.

5.2.2 Effectiveness of Algorithm

To see the effectiveness of our algorithm of searching regions, especially for a query containing variables, we compared search time by using our algorithm with that by a simplest baseline algorithm, which searches all regions of sub-queries and creates combination of all regions. For example, when the query is

```
(> [sentence]
   (& [word arg1=$subject base="activate"]
      (& (> [phrase id=$subject]
            (> [word] "p53"))
```

the algorithm searches all regions of the following sub-queries in order to create all combination of regions.

```
[word arg1=$subject base="activate"]
[phrase id=$subject]
[word]
"p53"
```

After the algorithm creates all combinations, it retrieves combinations in which values for the same variables are idential as result regions. In the example of the above query, the algorithm retrieves combinations in which values for the variable `$subject` in the first and second queries are identical. In the experiments, we used the queries in the TREC 2004 test collection in Table 5.6 and Table 5.7, converted to the expression of region algebra. Target documents are the set of documents which have relevance judgement in the TREC 2004 test collection. Table 5.15 shows search time by using our algorithm and that by searching all combinations. The result show that our algorithm is very efficient compared with the algorithm creating all combinations. In the case of this experiment, since the number of the annotations `[phrase]` from the parser was very huge, our algorithm, which skips annotations which are not related to the target query, was very effective.

5.2.3 Effectiveness of Ordering Sub-Queries

To see the effectiveness of ordering value determining queries, we compared our system with the system in which the value determining queries are evaluated in the reverse order, or in the descending order of estimated frequency. Table 5.14 illustrates the search time of the two systems. When all queries for the tag regions containing variables are conditioned with some words in the query, there is no large difference between the search time of two systems. However, when some queries for the tag region containing variables are not conditioned with any words e.g. `[phrase id=$vp0 lex\_head=$verb]`, such queries are evaluated first. Then, the search time becomes longer because the algorithm has to search almost all of the tag regions matching the query.

5.3 Discussion

5.3.1 Annotations

In our semantic search system, we annotated the documents with large amount of annotations for NLP modules, but it seems to be too many annotations to retrieve documents for querying only "subject-verb-object" relation. In the example of data in Figure 5.6, the "subject-verb-object" search system do not need the phrase tag of 'id="2",' because this phrase does not correspond to the "subject-verb-object" relation. In the same reason, the phrase tags whose id number is 2, 4, 5, 7, 8, 11, 13, 14 and 17 can be eliminated. 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 'lex_head' 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 our system supports not only the "subject-verb-object" retrieval but also other type of queries, and some users will use these annotation in some queries and

```
<sentence>
 <phrase id="0" cat="S" head="4" lex_head="6">
  <phrase id="1" cat="NP" head="2" lex_head="3">
   <phrase id="2" cat="NP" head="3" lex_head="3">
    <word id="3" pos="NN" cat="NP" base="p53">p53</word>
   </phrase>
  </phrase>
 <phrase id="4" cat="VP" head="5" lex_head="6">
  <phrase id="5" cat="VP" head="6" lex_head="6">
   <word id="6" pos="VBZ" cat="VP" base="is" arg1="1" arg2="7">
   is
   </word>
  </phrase>
  <phrase id="7" cat="VP" head="8" lex_head="9">
   <phrase id="8" cat="VP" head="9" lex_head="9">
    <word id="9" pos="VBN" cat="VP" base="phosphorylate"
          arg2="1" arg1="-1" arg3="10"
          rel_type="phosphorylation">
    phosphorylated
    </word>
   </phrase>
   <phrase id="10" cat="VP" head="11" lex_head="12">
    <phrase id="11" cat="VP" head="12" lex_head="12">
     <word id="12" pos="TO" cat="VP" arg1="1" arg2="13">
     to
     </word>
    </phrase>
    <phrase id="13" cat="VP" head="14" lex_head="15">
     <phrase id="14" cat="VP" head="15" lex_head="15">
      <word id="15" pos="VB" cat="VP" base="activate" arg1="1"
            arg2="16">
       activate
      </word>
     </phrase>
     <phrase id="16" cat="NP" head="17" lex_head="18">
      <phrase id="17" cat="NP" head="18" lex_head="18">
       <word id="18" pos="NN" cat="NP" base="cd25" arg1="1">
        CD25
       </word>
      </phrase>
     </phrase>
    </phrase>
   </phrase>
  </phrase>
 </phrase>
</sentence>
```

Figure 5.6: Example of XML data

```
<phrase id="50" cat="NP" head="53" lex_head="54">
  <phrase id="51" cat="NP" head="52" lex_head="52">
    <word id="52" pos="NN" cat="NP" base="colon"
          mod="53">
      colon
    </word>
  </phrase>
  <phrase id="53" cat="NP" head="54" lex_head="54">
    <word id="54" pos="NN" cat="NP" base="cancer">
      cancer
    </word>
  </phrase>
</phrase>
```

Figure 5.7: Example of data which express “modification” in in-line XML

new valuable queries will be constructed by combining two types of information when other type of information is annotated to the same document. For example, when users want to search sentences containing “colon cancer”, users can send a query

```
(> [sentence] (& colon cancer))
```

which expresses sentences containing both “colon” and “cancer.” But users can also specify the relation ‘ “colon” modifies “cancer” ’ by a query like

```
(> [sentence]
   (& [word base="colon" mod=$mod]
      (> [phrase id=$mod] cancer))
```

because such relation is expressed by “mod” attribute in the data as shown in Figure 5.7.

5.3.2 Scalability

As shown in the experiments, search time in our system is linear to the number of documents and complexity of query, that is, the number of operators in the query. In the case of the exact match retrieval, since the system filters the documents by using the keyword-based queries, the number of documents in which a structured query is evaluate is the minimum document frequency in the words appeared in the structured query. In the case of the ranking retrieval, since the idf value for structured queries is calculated, the system need to search all documents in which the words in the query appear. By calculating the df values in advance or estimating the idf values for the structured queries based on the values which can be calculated preliminary, such as df or tf values of words, the system will calculate ranking of document efficiently.

Search time will be reduced by parallelization or decentralization of algorithm and indices, which is generally used in many search systems. For the exact match retrieval, the search process can be parallelized easily by separating regions in which each parallelized process searches. The search algorithm can be separated into two steps, the document search step, in which the algorithm searches documents by using the keyword-based queries, and the structured search step, in which the algorithm searches regions matching to the structured queries. Since the former step is only a keyword-based search, the search process can be parallelized by existing method for keyword-based search. The latter step also can be parallelized easily since the process to search matching regions can be executed independently in each document searched in the former step. Moreover, since the latter step can be executed as long as the target documents searched, the former step and the latter step will be also parallelized. For the ranking retrieval, the calculation of idf value for structure queries is needed in the current algorithm, that is, the algorithm have to search whole database to count the document frequency of the regions matching to the structure query. Pallarelization of search process is very effective for the current algorithm of the ranking retrieval. Decentralization of search process also can be applied to our framework easily. The index structure of our search framework is very simple, which is the inverted index of positions for words, tags and their attributes on the assumption that the documents are expressed as the stream of characters. However since the search process can be evaluated independently in each document, the indices, the list of positions, can be separated unless the unit of documents is broken.

The index size is reduced in comparison with original documents size. In this experiments, as is explained above, we create indices for all annotated information in the results from NLP modules. However the index size can be reduced by eliminating the annotated information in advance which is never used in search process.

5.3.3 Evaluation of Our Framework on TREC Test Collection

Currently, there is no test collection for semantic retrieval which directly matches to the scope of our framework. Although the INEX provides the test collections for XML retrieval, these test collections are different from the scope of our frameworks because the text is structured by document structure, such as chapter, section, title and author and so on. The target of our system is a set of documents containing more detailed structure annotated with results of NLP modules, such as syntactic/semantic parser and named entity recognizer and so on. In order to evaluate the our framework in existing test collections, we used test collection of TREC Genomic track since the target data of the test collection is the same with the target data of our MEDIE system and some of the search topic is the same with the search topic in our system, such as search of the relation between a gene and a disease.

The TREC test collection is constructed by using keyword-based search system. First, the participants of TREC competition retrieved documents from a document set by these own retrieval system, and the result documents are collected. The collected documents are distributed to the participants and then they assign a relevance judgement to documents whether a document is relevant to the topic or not.

For the topics in the TREC test collection, which are written as natural language sentences, we needed to convert the sentences in topics into the query for our search framework. We applied NLP modules which is also applied to target documents to the topic sentences, and construct structured queries using the results from NLP modules by hand. Since the queries are constructed by hand, comparison of accuracy in this experiments to that of search systems which join the TREC competition means little. However, automatic construction of queries effective to accuracy of search from the natural language sentences is very difficult. Structured queries can be constructed straightforwardly by using the NLP results, for example, words in the query are expanded by using the results from the named entity recognizer or the structure of queries is constructed from the parsing result, especially predicate-argument structure. But the queries constructed by such process contains many meaningless queries for searching documents which contains the information which users would like to search. Determination of the importance of queries based on the analysis of sentence or elimination of such meaningless queries is very advanced task. As shown in the experimental results on the test collection in TREC 2004 (Table 5.11), even the queries created by hands decrease accuracy in some queries. However, when topic sentences are converted to queries automatically in high accuracy, we will be able to evaluate our framework not only in the TREC test collection, but in other task such as Question Answering.

In our current system including NLP modules used to annotate text, it is difficult to specifying the relation over sentence boundary, such as anaphoric relation, since the annotation of the corresponding NLP module does not exist. When the information need written in topics is expressed in multiple sentences, the specification of structure such as the result from the parser is not effective. However, the experimental results show a certain amount of possibility of accuracy improvement by using the result of NLP modules.

5.3.4 Search Accuracy and Results from NLP modules

The experimental results on the test collection in TREC 2004 (Table 5.11) shows that the results of named entity recognition largely improve the accuracy of retrieval when compared to keyword search. This is the same with the traditional query expansion in keyword based search. The documents not containing original words in topics but containing the different expression of the object in topic are retrieved by the expansion with the results of named entity recognition. The difference between traditional query expansion and our expansion it that the traditional methods expand only queries, but our framework expands not only queries but also documents by annotating them with the results from name entity recognizer. By expanding the documents, the matching of expanded queries and documents becomes easy by matching the system only matches the expanded form of words in queries and documents.

The specification of predicate argument structure is effective in some queries. Especially, specification of modification relation between the words is effective. These queries increase the score of documents containing the words and the modification relation among the words, which cannot be captured in the traditional keyword-based retrieval.

In the case of specification of more complex structure, such as the subject-verb-object relation by predicate argument structure, when the document contains the relation the document is relevant the original topics in high probability. But the number documents matching to such queries is very small or zero. For example, in the case of the subject-verb-object relation, documents do not match to the query because the verb is expressed in different verbs whose meaning is similar to the query verb, or the relation itself is expressed in different expression to predicate argument structure, such that the relation is described by verbal noun or the relation is described across multiple sentences. Although some expressions can be resolved by the result of the event recognizer, these description variation of relation is beyond the current system since the expansion for such variation cannot be produced by results of NLP modules in current system. In some queries, the specification of the structure decreases accuracy. When the queries used in search is not related to the central theme of topics, the system may wrongly increase score of irrelevant documents. Therefore, in order to search documents in high accuracy for the information need written in natural language sentences, the system has to recognize the important part of the information need, and determine the importance of queries in the input sentences separately from the importance calculated based on the frequency in target documents. This types of recognition is beyond the current framework.

Chapter 6

Conclusion

This thesis proposed a semantic search system incorporating various NLP systems. The semantic search system is constructed based on a search framework for tag-annotated textbases. We constructed a search framework for documents in which text are annotated with various types of annotations, such as results from NLP modules or manual annotations. In order to search documents using these annotations, we extended the framework of region algebra and construct an algorithm to search over annotations. In order to apply region algebra into documents with annotations from NLP systems, we extended the region algebra and its search algorithm to deal with the nested annotations properly. Moreover, we incorporated variables into the region algebra and the efficient algorithm to calculate value matching, in order to search relations that annotations refer another annotations, which is used to represent the relationships among words, such as "subject-verb-object" relations from a syntactic/semantic parser. We also proposed a method for raking retrieval using annotated informations. We incorporated dependency among queries into Binary Independent Model, which is a traditional probabilistic model for keyword-base search on the assumption that no dependency exists among words.

We constructed a semantic search system, MEDIE, for the a set of paper abstracts in biomedical area, the MEDLINE databases. The target documents of the MEDIE system are annotated with the annotations from several NLP modules, such as paring, named entity recognition, event recognition and GDA recognition, and construct indices for search across these annotations. In the experiments, we evaluated our system on our own data and publically usable test collection, which is created in TREC Genomic Tracks. We showed the effectiveness of incorporating NLP systems into search systems. The expansion of terms with the result from named entity recognizer is especially effective in the current system, but the specification of NLP results in ranking retrieval decreases the accuracy for some queries.

However, several functionalities are lacking in the current framework. One of the future works is an estimation of weighting for ranking retrieval. In the current framework, the IDF values for structured query is calculated on the fly. Since the DF values for all variation of structured queries cannot be calculated in advance, unlike the DF values for words, we will need to estimate the DF values for structured queries by using values for words. Another is the automated construction of queries from natural language sentences. In the experiments, we constructed queries from natural language sentence manually, and we showed a certain amount of effectiveness of using annotations of NLP results in retrieval. However, the system should recognize users' search intention from the input sentences to the system and create effective queries automatically for searching information which matches the users' request.

References

[1] Genia sentence splitter. http://www-tsujii.is.s.u-tokyo.ac.jp/ y-matsu/geniass/.

[2] Nasreen Abdul-jaleel, James Allan, W. Bruce Croft, O Diaz, Leah Larkey, Xiaoyan Li, Donald Metzler, Mark D. Smucker, Trevor Strohman, Howard Turtle, and Courtney Wade. Umass at trec 2004: Notebook. In *In TREC 2004*, pages 657–670, 2004.

[3] Alex S Ade, Zachary C Wright, Aaron V Bookvich, Brian D Athey, et al. Misearch adaptive pubmed search tool. *Bioinformatics*, 25(7):974–976, 2009.

[4] Sean Bechhofer. Owl: Web ontology language. In *Encyclopedia of Database Systems*, pages 2008–2009. Springer, 2009.

[5] P. Bellot, T. Chappell, A. Doucet, S. Geva, S. Gurajada, J. Kamps, G. Kazai, M. Koolen, M. Landoni, M. Marx, A. Mishra, V. Moriceau, J. Mothe, M. Preminger, G. Ramírez, M. Sanderson, E. Sanjuan, F. Scholer, A. Schuh, X. Tannier, M. Theobald, M. Trappett, A. Trotman, and Q. Wang. Report on inex 2012. *SIGIR Forum*, 46(2):50–59, December 2012.

[6] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[7] Peter Boncz, Torsten Grust, Maurice van Keulen, Stefan Manegold, Jan Rittinger, and Jens Teubner. MonetDB/XQuery: A Fast XQuery Processor Powered by a Relational Engine. In *Proceedings of the 2006 ACM SIGMOD international conference*, pages 479–490, June 2006.

[8] Razvan Bunescu, Ruifang Ge, Rohit J. Kate, Edward M. Marcotte, Raymond J. Mooney, Arun Kumar Ramani, and Yuk Wah Wong. Comparative experiments on learning information extractors for proteins and their interactions, 2005.

[9] Razvan C. Bunescu and Raymond J. Mooney. Subsequence kernels for relation extraction. In *NIPS*, 2005.

[10] Forbes J. Burkowski. An algebra for hierarchically organized text-dominated databases. *Information Processing and Management*, 28(3):333–348, 1992.

[11] Don Chamberlin. XQuery: An XML query language. *IBM Systems Journal*, pages 597–615, 2002.

[12] Rashmi Chauhan, Rayan Goudar, Robin Sharma, and Atul Chauhan. Domain ontology based semantic search for efficient information retrieval through automatic query expansion. In *Intelligent Systems and Signal Processing (ISSP), 2013 International Conference on*, pages 397–402. IEEE, 2013.

[13] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0, 1999.

[14] Charles L. A. Clarke, Gordon V. Cormack, and Forbes J. Burkowski. An algebra for structured text search and a framework for its implementation. *The computer Journal*, 38(1):43–56, 1995.

[15] Tim Converse, Ronald M. Kaplan, Barney Pell, Scott Prevost, Lorenzo Thione, and Chad Walters. Powerset's natural language Wikipedia search engine. In *Wikipedia and Artificial Intelligence: An Evolving Synergy, Papers from the 2008 AAAI Workshop*, 2008.

[16] David Dubin. The most influential paper gerard salton never wrote. *Library Trends*, page 2004.

[17] Sébastien Ferré and Alice Hermann. Semantic search: Reconciling expressive querying and exploratory search. In *The Semantic Web–ISWC 2011*, pages 177–192. Springer, 2011.

[18] David Ferrucci and Adam Lally. UIMA: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348, 2004.

[19] Jean-Fred Fontaine, Adriano Barbosa-Silva, Martin Schaefer, Matthew R Huska, Enrique M Muro, and Miguel A Andrade-Navarro. Medlineranker: flexible ranking of biomedical literature. *Nucleic acids research*, 37(suppl 2):W141–W146, 2009.

[20] Norbert Fuhr. Probabilistic models in information retrieval. *The Computer Journal*, 35(3):243–255, 1992.

[21] Claudio Giuliano, Alberto Lavelli, and Lorenza Romano. Exploiting shallow linguistic information for relation extraction from biomedical literature. In *EACL*, 2006.

[22] Brian Goetz. The Lucene search engine: Powerful, flexible, and free. In *JavaWorld*, 2000.

[23] Tadayoshi Hara, Yusuke Miyao, and Jun'ichi Tsujii. Adapting a probabilistic disambiguation model of an HPSG parser to a new domain. In Robert Dale, Kam-Fai Wong, Jian Su, and Oi Yee Kwong, editors, *IJCNLP 2005*, volume 3651 of *LNAI*, pages 199–210, Jeju Island, Korea, October 2005. Springer-Verlag. ISSN 0302-9743.

[24] Tadayoshi Hara, Yusuke Miyao, and Jun'ichi Tsujii. Evaluating impact of re-training a lexical disambiguation model on domain adaptation of an hpsg parser. In *Proceedings of IWPT 2007*, Prague, Czech Republic, June 2007.

[25] William Hersh, Aaron Cohen, Jianji Yang, Ravi Teja Bhupatiraju, Phoebe Roberts, and Marti Hearst. Trec 2005 genomics track overview. In *In TREC 2005 notebook*, pages 14–25, 2005.

[26] William R. Hersh, Ravi Teja Bhuptiraju, Laura Ross, Phoebe Johnson, Aaron M. Cohen, and Dale F. Kraemer. Trec 2004 genomics track overview. In *In Proc. of the 13th Text REtrieval Conference*, 2004.

[27] Kenji Hirohata, Naoaki Okazaki, Sophia Ananiadou, and Mitsuru Ishizuka. Identifying sections in scientific abstracts using conditional random fields. In *Proceedings of the Third International Joint Conference on Natural Language Processing (IJCNLP 2008)*, pages 381–388, 2008.

[28] Nancy Ide and C. M. Sperberg-McQueen. The text encoding initiative – its history, goals and future development. *Computers and the Humanities*, (29):5–16, 1995.

[29] Jani Jaakkola and Pekka Kilpelainen. Nested text-region algebra. Technical Report C-1999-2, University of Helsinki, 1999.

[30] Thorsten Joachims. *Advances in Kernel Methods: Support Vector Learning*, chapter 11 - Making Large-scale SVM Learning Practical. MIT Press, 1999.

[31] K. Sparck Jones, S. Walker, and S. E. Robertson. A probabilistic model of information retrieval: development and comparative experiments. In *Information Processing and Management*, pages 779–840, 2000.

[32] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[33] Jin-Dong Kim, Tomoko Ohta, Yuka Teteisi, and Jun'ichi Tsujii. GENIA corpus - a semantically annotated corpus for bio-textmining. *Bioinformatics*, 19(suppl. 1):i180–i182, 2003.

[34] Jin-Dong Kim, Tomoko Ohta, and Jun'ichi Tsujii. Corpus annotation for mining biomedical events from literature. *BMC Bioinformatics*, 9(1):10, 2008. ISSN 1471-2105.

[35] Jung-jae Kim, Piotr Pezik, and Dietrich Rebholz-Schuhmann. Medevi: retrieving textual evidence of relations between biomedical concepts from medline. *Bioinformatics*, 24(11):1410–1412, 2008.

[36] Graham Klyne and Jeremy J Carroll. Resource description framework (rdf): Concepts and abstract syntax. 2006.

[37] Sadao Kurohashi and Makoto Nagao. A syntactic analysis method of long japanese sentences based on the detection of conjunctive structures. *Comput. Linguist.*, 20(4):507–534, December 1994.

[38] Robert Leaman and Graciela Gonzalez. Banner: an executable survey of advances in biomedical named entity recognition. *Pacific Symposium on Biocomputing*, pages 652–663, 2008.

[39] Changki Lee and Gary Geunbae Lee. Probabilistic information retrieval model for a dependency structured indexing system. *Inf. Process. Manage.*, 41(2):161–175, March 2005.

[40] Ki-Joong Lee, Young-Sook Hwang, Seonho Kim, and Hae-Chang Rim. Biomedical named entity recognition using two-phase model based on svms. *J. of Biomedical Informatics*, 37(6):436–447, 2004.

[41] D Maglott, J Ostell, K D Pruitt, and T Tatusova. Entrez gene: gene-centered information at ncbi. *Nucleic Acids Res*, 33(Database issue):54–58, January 2005.

[42] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. The Penn Treebank: Annotating predicate argument structure. In *ARPA HLT Workshop*, 1994.

[43] M. E. Maron and J. L. Kuhns. On relevance, probabilistic indexing and information retrieval. *J. ACM*, 7(3):216–244, July 1960.

[44] Katsuya Masuda, Takuya Matsuzaki, and Jun'ichi Tsujii. Semantic search based on the online integration of nlp techniques. *Procedia-Social and Behavioral Sciences*, 27:281–290, 2011.

[45] Katsuya Masuda and Jun'ichi Tsujii. Tag-annotated text search using extended region algebra. Number 12, pages 2369–2377, 2009.

[46] Wolfgang Meier. exist: An open source native xml database. In *Web-Services, and Database Systems, NODe 2002 Web and Database-Related Workshops*, pages 169–183. Springer, 2002.

[47] Donald Metzler and W. Bruce Croft. Combining the language model and inference network approaches to retrieval. *Inf. Process. Manage.*, 40(5):735–750, September 2004.

[48] H. Mima, S. Ananiadou, G. Nenadic, and J. Tsujii. A Methodology for Terminology-based Knowledge Acquisition and Integration. In *Proceedings of Coling 2002*, pages 667–673, 2002.

[49] Tomohiro Mitsumori, Masaki Murata, Yasushi Fukuda, Kouichi Doi, and Hirohumi Doi. Extracting protein-protein interaction information from biomedical text with svm. *IEICE - Trans. Inf. Syst.*, E89-D(8):2464–2466, 2006.

[50] Y. Miyao and J. Tsujii. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of ACL 2005*, pages 83–90, 2005.

[51] Yusuke Miyao, Tomoko Ohta, Katsuya Masuda, Yoshimasa Tsuruoka, Kazuhiro Yoshida, Takashi Ninomiya, and Jun'ichi Tsujii. Semantic retrieval for the accurate identification of relational concepts in massive textbases. In *Proceedings of COLING-ACL 2006*, pages 1017–1024, Sydney, Australia, July 2006.

[52] Yusuke Miyao and Jun'ichi Tsujii. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of ACL 2005*, pages 83–90, 2005.

[53] Alessandro Moschitti. Making tree kernels practical for natural language learning. In *EACL*, 2006.

[54] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30:3–26, 2007.

[55] Takashi Ninomiya, Takuya Matsuzaki, Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Tsujii. Extremely lexicalized models for accurate and fast HPSG parsing. In *Proceedings of EMNLP 2006*, pages 155–163, 2006.

[56] Tomoko Ohta, Yusuke Miyao, Takashi Ninomiya, Yoshimasa Tsuruoka, Akane Yakushiji, Katsuya Masuda, Jumpei Takeuchi, Kazuhiro Yoshida, Tadayoshi Hara, Jin-Dong Kim, Yuka Tateisi, and Jun'ichi Tsujii. An intelligent search engine and GUI-based efficient MEDLINE search tool based on deep syntactic parsing. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 17–20, Sydney, Australia, July 2006.

[57] Naoaki Okazaki. Crfsuite: a fast implementation of conditional random fields (crfs), 2007.

[58] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[59] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications, 1994.

[60] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the 13th Conference on CoNLL*, pages 147–155, 2009.

[61] Dietrich Rebholz-Schuhmann, Harald Kirsch, Miguel Arregui, Sylvain Gaudan, Mark Riethoven, and Peter Stoehr. Ebimed窶付ext crunching to gather facts for proteins from medline. *Bioinformatics*, 23(2):e237–e244, 2007.

[62] S. E. Robertson. The probability ranking principle in ir. *Journal of Documentation*, 33(4):294–304, 1977.

[63] S. E. Robertson. Readings in information retrieval. chapter The probability ranking principle in IR, pages 281–286. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[64] S. E. Robertson and Sparck K. Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976.

[65] Stephen E. Robertson and Karen Sparck Jones. Document retrieval systems. chapter Relevance weighting of search terms, pages 143–160. Taylor Graham Publishing, London, UK, UK, 1988.

[66] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. Okapi at trec-3. In *Overview of the Third Text REtrieval Conference (TREC-3)*, pages 109–126, 1996.

[67] Thomas Roelleke and Jun Wang. Probabilistic logical modelling of the binary independence retrieval model. In *Proceedings of the 1st International Conference on Theory of Information Retrieval (ICTIR 07) - Studies in Theory of Information Retrieval*, 2007.

[68] Rune Sætre, Kenji Sagae, and Jun'ichi Tsujii. Syntactic features for protein-protein interaction extraction. In *The 2nd International Symposium on Languages in Biology and Medicine (LBM) 2007 - SHORT PAPERS*, volume 319 of *ISSN 1613-0073*, pages 6.1–6.14. CEUR Workshop Proceedings, January 2008.

[69] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975.

[70] Gerald Salton and Chris Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[71] P. L. Schuyler, W. T. Hole, M. S. Tuttle, and D. D. Sherertz. The UMLS Metathesaurus: representing different views of biomedical concepts. *Bull Med Libr Assoc*, 81(2):217–222, April 1993.

[72] Keiji Shinzato, Tomohide Shibata, Daisuke Kawahara, and Sadao Kurohashi. Tsubaki: An open search engine infrastructure for developing information access methodology. *Journal of Information Processing*, 20(1):216–227, 2012.

[73] Yoshimasa Tsuruoka and Jun'ichi Tsujii. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of the conference on HLT and EMNLP*, pages 467–474, 2005.

[74] H. Turtle and W. B. Croft. Inference networks for document retrieval. In *Proceedings of the 13th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '90, pages 1–24, New York, NY, USA, 1990. ACM.

[75] C. T. Yu and G. Salton. Precision weighting – an effective automatic indexing method. *J. ACM*, 23(1):76–88, January 1976.

[76] Hwanjo Yu, Taehoon Kim, Jinoh Oh, Ilhwan Ko, Sungchul Kim, and Wook-Shin Han. Enabling multi-level relevance feedback on pubmed by integrating rank learning into dbms. *BMC bioinformatics*, 11(Suppl 2):S6, 2010.