December 5, 2005
RT0634
Computer Science; Life Sciences 10 pages

Research Report

SQL-based Aggregation for Text Mining

Akihiro Inokuchi and Kohichi Takeda

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

1 Introduction

The life science industry is an emerging market in which the application spaces, such as drug discovery and development in the pharmaceutical sector and clinical record management in health care, have become areas of significant interest [1]. Documents in the scientific literature play an important role in life science by serving as a potential source for underlying knowledge discovery. These documents are a rich repository of information on relationships among biomedical concepts such as genes, proteins, diseases, and a variety of other key topics. Text mining is a technology that makes it possible to discover patterns and trends semi-automatically from huge collections of unstructured text [2, 3, 4, 5]. It is based on technologies such as natural language processing, information retrieval, information extraction, and data mining. Early papers in this area mentioned the possibility of knowledge discovery from the biomedical literature. Considerable research has been done in the areas of biomedical concept extraction (named-entity extraction), relationship extraction, and network or pathway construction for protein-protein interactions.

We developed IBM TAKMI for Biomedical Documents to facilitate knowledge discovery from the very large text databases characteristic of life science and healthcare applications [6]. This set of tools, designated MedTAKMI, is an extension of the TAKMI (Text Analysis and Knowledge MIning) system originally developed for text mining in customer relationship management applications. MedTAKMI dynamically and interactively mines a collection of documents to obtain the characteristic features within them. By using multifaceted mining of these documents together with biomedically motivated categories for term extraction and a series of drill-down queries, users can obtain knowledge about a specific topic after seeing only a few key documents. In addition, the use of natural language techniques makes it possible to extract deeper relationships among the biomedical concepts. The MedTAKMI system is capable of mining the entire MEDLINE database of 11 million biomedical journal abstracts. It is currently running at a customer site.

Since MedTAKMI uses a proprietary index as a modified DTM (Document-Term Matrix) and a proprietary aggregate engine, and is implemented in C++, it can run each of its functions quickly. However, it is not easy to expand it to develop other functions and to integrate it with other systems. In this paper, we propose an SQL-based method for storing annotated words in a relational database and computing each function by SQL. Although the original MedTAKMI was implemented in a few thousands of lines of C++ code, the proposed method is implemented in a few lines of SQL and is comparable with the original MedTAKMI. The rest of this paper is organized as follows. We briefly explain our MedTAKMI system in Section 2 and the conventional method to aggregate a distribution of documents for keywords in Section 3. In Section 4, we introduce our data model to efficiently compute the distribution and its implementation. Section 5 presents experimental results using about 500,000 journal abstracts and Section 6 concludes this paper.

2 MedTAKMI

The MedTAKMI architecture consists of two main components: a preprocessing information extraction stage and a runtime search/mining server. In this section we briefly explain these main components.

2.1 Preprocess

In the preprocessing phase of the MedTAKMI system, document titles and abstracts are parsed by CCAT [7], a shallow syntactic parser. Because this is a general-purpose parser that has not been trained for biomedical documents, it is difficult to obtain optimized results by parsing documents from the medical domain. We solve this problem by first annotating the text with domain dictionaries. The annotations facilitate the parsing of medical-domain text even when the parser has not been specifically trained for this domain.

In the first step of the preprocessing, the term annotator finds words in the input text using the term dictionary and identifies these words using their canonical form. Most of the technical terms in the medical domain are compound words. Thus, biomedical terms tend to consist of a combination of numerals, symbols, and verbs, making it very difficult to find term boundaries. The annotations made by the technical term dictionary are based upon a part-of-speech analysis. In addition, there can be multiple expressions that are synonymous with a particular technical term. These can arise from abbreviations or acronyms as well as from spelling variations.

If these variations are recognized as different entities, it can often cause problems for text mining. For instance, "DNA" and "deoxyribonucleic acid" are synonyms. The dictionary contains spelling and abbreviation variants and their canonical forms. By reducing these variants to a single canonical form, we can treat them as the same entity.

In the second step, the text annotated with a technical term dictionary is passed to the syntactic parser. The parser outputs segments of phrases labeled with their syntactic roles, for example NP (noun phrase) or VG (verb group). In the third step, the category annotator assigns categories to the terms in these segments and phrases. The category dictionary consists of a set of canonical forms and their categories, which also indicates the node label in the hierarchy of categories. This information can include sets of keywords (e.g. protein names), predicate-argument binary relations (e.g. "activate-protein"), and ternary relations (e.g. subject-verb-object dependency triplets). All extracted information is finally encoded into an index file that is used by the runtime part of the system.

Fig. 1 shows an example of the preprocessing in the MedTAKMI system. When "Repetitive sequence-based polymerase chain reaction affects deoxyribonucleic acids" is given as input, an annotator assigns "DNA" as canonical and "proper noun" as part-of-speech to "deoxyribonucleic acids". After parsing the annotated text, categories are assigned to each word, and binary and ternary relations with their categories are extracted.

2.2 Aggregation and Mining Functions

The MedTAKMI system is a text-mining system that supports keyword-based search engines, and the mining process can be applied to the whole database or to a subset document collection obtained by a series of searches. Users can submit a query and receive a document collection in which each document contains the query keywords or their synonyms. Mining functions can then be applied to the collection in order to discover underlying information, such as protein-protein relationships. Alternatively, users can continue the search process by using the results of previous searching and mining operations. Interactivity is a very important MedTAKMI feature, because it allows users to switch between searching and mining in a flexible manner.

MedTAKMI provides various mining functions for large document collections: keyword-based and full-text searching, a hierarchical category viewer, a two-dimensional viewer (term-associations), and other analytical tools. The hierarchical category viewer provides a distribution of documents for keywords in a data collection over a predefined hierarchy. For example, the viewer returns the distribution of documents for each keyword belonging to the "Disease" node and its child nodes in the MeSH hierarchy. The two-dimensional viewer allows a user to visualize the strength of association between keywords. For example, the viewer returns protein-protein associations in a document collection related to mouse. Most of these viewers can function interactively. This interactivity is an important requirement for users in the lifescience domain. For example, users may wish to use a fact discovered in a previous mining result to structure a new query from a different point of view. Indeed, a user often cannot define a complete query beforehand but rather must iteratively refine the query based on

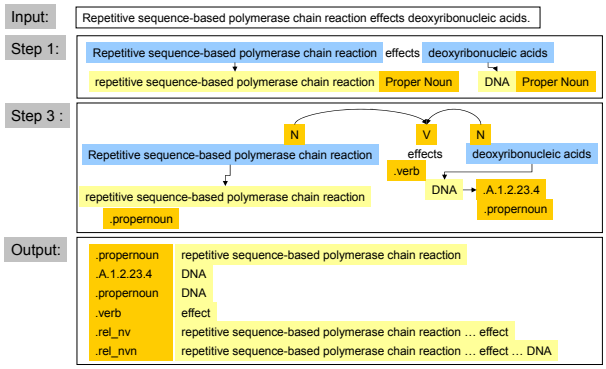

Figure 1: Example of the Preprocessing in the MedTAKMI System

previous results. Furthermore, when a document collection is very large, a single query may not be able to sufficiently narrow the collection, then requiring the user to submit a sequence of queries to obtain the desired subset of documents. MedTAKMI's viewers help users to navigate toward such a goal interactively.

Usage Scenario: The MedTAKMI system can be used to aid in drug discovery and clinical information retrieval. For example, defects in the AML1 gene are thought to cause acute leukemia. Suppose we are interested in developing a treatment for leukemia and are looking for potential drug targets. After narrowing down to a document collection containing "AML1", a category viewer based on a category of the NCBI LocusLink44 phenotype information indicates that "leukemia" is indeed the term most frequently associated with AML1. Furthermore, the term leukemia appears 54.41 times more often in the collection than it does in the entire sample database.

3 Conventional Method

In this paper, we focus on the hierarchical category viewer using a simple example, because it is most often used among MedTAKMI ' s functions and its fundamental methods are similar to those of the other functions. The original MedTAKMI uses a Document-Term Matrix (DTM) to quickly compute the results for the category viewer. Let the sets of terms and documents be $T = \{t_1, t_2, \ldots, t_n\}$ and $D = \{d_1, d_2, \ldots, d_m\}$, respectively. A TDM is a matrix $M = (m_{ij})$ of $m \times n$, and an elements m_{ij} represents how many times the term t_j appears in the document d_i. Although storing the whole matrix requires a lot of memory, it can be compressed by storing a pair for each element that is not zero and its index in each row or column, because the matrix is very sparse.

As mentioned in Section 2.1, since each word is assigned some categories, MedTAKMI stores the following modified DTM. Rows in our DTM correspond to a set of documents $D = \{d_1, d_2, \ldots, d_m\}$ similar to the conventional DTM, and columns correspond to a set of pairs of categories and keywords $T = \{(c_1, k_1), (c_2, k_2), \ldots, (c_n, k_n)\}$. For example in Fig. 1, c_j=.A.1.2.23.4 and k_j=DNA. It may be either $c_i = c_j$ or $k_i = k_j$ for $i \neq j$. To facilitate counting the number of documents for each subcategory in the category viewer, when an element m_{ij} for d_i and (c_j, k_j) in DTM is incremented by 1, the element $m_{ij'}$ for d_i and $(c'_j, NULL)$ is also incremented by 1, where c'_j is an ancestor category of c_j. For example in Fig. 1, $c_{j'}$=.A.1.2.23 is an ancestor category of c_j=.A.1.2.23.4.

Fig. 2 shows how MedTAKMI computes the results for the category viewer after narrowing down to the documents containing a word k_0 whose category is c_0, when the user specified the category c_3. First, MedTAKMI narrows the search down to the document set $\{d_1, d_5, d_9\}$ containing (c_0, k_0) (1). In parallel with this Process 1, a set of keywords $\{k_3, k_4, k_5, k_6, k_7\}$ whose categories are equal to c_3 is output(2). After Processes 1 and 2, the distribution of documents for keywords appearing in the documents $\{d_1, d_5, d_9\}$ is returned as $\{(c_3, k_3) : 2, (c_7, k_7) : 1\}$ (3). MedTAKMI also computes the distribution of the documents for each subcategory of a category specified by the user. This means that it returns the number of documents containing keywords whose category or its ancestor is equal to c_3.1, the number of documents containing keywords whose category or its ancestor is equal to c_3.2, and so on. In this case, a set of categories $(c_3.*, NULL)$ are returned in Process 2. Since Process 3 requires much more computation time than Processes 1 and 2, Process 3 is calculated for a subset of the documents sampled from the document collection. The cardinality of such a subset can be specified by the user.

Since MedTAKMI uses a proprietary index as a modified DTM and a proprietary aggregate engine and is implemented in C++, it can run each function quickly. However, it is not easy to expand it to develop other functions and to integrate it with other systems.

4 Proposed Method

We define two tables, `CATEGORY` and `KEYWORD`, to store a hierarchical category tree and annotated keywords as

```
CATEGORY (PATH VARCHAR, DESCRIPTION VARCHAR, PARENT VARCHAR), and
KEYWORD (DOCID INT, KEYWORD VARCHAR, PATH VARCHAR).
```

Each record in `CATEGORY` and `KEYWORD` corresponds to a node in the category tree and an annotated keyword, respectively. `PATH`, `DESCRIPTION` and `PARENT` in `CATEGORY` are a category path for a node in the category tree, a description for the node, and a category path for its parent node, respectively, and `DOCID`, `KEYWORD` and `PATH` in

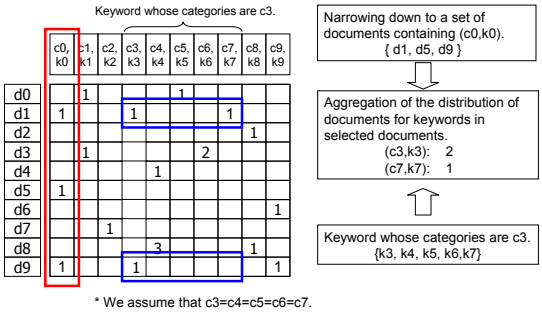

Figure 2: Example of Hierarchical Category Viewer

KEYWORD are a document ID and an annotated keyword, and the path for its category, respectively. An SQL query to aggregate the distribution of documents containing any keywords which belong to a category *path* specified by a user can be represented as

```
SELECT KEYWORD, COUNT(DISTINCT DOCID) AS COUNT
  FROM KEYWORD WHERE PATH='path' GROUP BY KEYWORD,          (1)
```

and an SQL query to aggregate for each subcategory can be represented as

```
SELECT DESCRIPTION, COUNT(DISTINCT ID) AS COUNT
  FROM CATEGORY AS A, KEYWORD AS B
  WHERE A.PATH='path' AND (B.PATH='path' OR B.PATH LIKE 'path.%')
  GROUP BY DESCRIPTION
UNION ALL
SELECT DESCRIPTION, COUNT(DISTINCT ID)
  FROM CATEGORY AS A, KEYWORD AS B
  WHERE PARENT='path' AND B.PATH LIKE 'path.%' AND (A.PATH=B.PATH
     OR RIGHT(B.PATH,LENGTH(B.PATH)-LENGTH(A.PATH)) like '.%'
     AND LOCATE(A.PATH,B.PATH)=1 )
  GROUP BY DESCRIPTION,                                     (2)
```

where *path* corresponds to c_3 in Fig. 2.

Because the SQL query (2) contains a string function, it requires a lot of computation time to aggregate the distribution. Therefore, we use the preorder-postorder method handling ancestor-descendent containment in a tree [8]. The method is a method for checking the containment by assigning a preorder and a postorder to each node in a tree as shown in Fig. 3 and comparing the numbers assigned to the two nodes. If a node A is an ancestor of a node B, the preorder of A must be less than one of B's and a postorder of A must be greater than one of B's. Since the string function is replaced in the relationship of preorder and postorder, the proposed method can quickly aggregate the distribution.

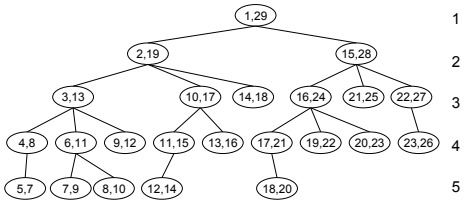

Figure 3: Preorders, Postorders, and Depth in a Tree

We modify the tables to

```
CATEGORY (PATH        VARCHAR,
          DESCRIPTION VARCHAR,
          PREORDER    INT,
          POSTORDER   INT,
          DEPTH       INT,
          PARENT      INT), and
KEYWORD (DOCID     INT,
         KEYWORD   VARCHAR,
         PREORDER  INT
         POSTORDER INT)
```

respectively, where PREORDER, POSTORDER, and DEPTH in CATEGORY are a preorder, a postorder and a depth of the corresponding node in the category tree, and PARENT is a preorder of the parent node of the node, respectively, and PREORDER and POSTORDER in KEYWORD are a preorder and a postorder for a category of the corresponding keyword, respectively.

Let *pre* and *post* be a preorder and a postorder of the node *path*. In accordance with the new tables, the SQL queries (1) and (2) are represented as

```
SELECT KEYWORD, COUNT(DISTINCT DOCID) AS COUNT
  FROM KEYWORD WHERE PREORDER=pre GROUP BY KEYWORD, and        (3)
SELECT DESCRIPTION, COUNT(DISTINCT DOCID) AS COUNT
  FROM CATEGORY AS A, KEYWORD AS B
  WHERE (A.PREORDER=pre OR A.PARENT=pre)
    AND B.PREORDER>=pre AND B.POSTORDER<=post
    AND B.PREORDER>=A.PREORDER AND B.POSTORDER<=A.POSTORDER
  GROUPY BY DESCRIPTION,                                       (4)
```

respectively. When a subset document collection is obtained by a series of searches, the SQL query (3) is modified as

```
SELECT KEYWORD, COUNT(DISTINCT DOCID) AS COUNT
  FROM KEYWORD WHERE PREORDER=pre AND DOCID IN (sql) GROUP BY KEYWORD,
```

where *sql* is a query which returns document IDs of the collection and corresponds to $\{d_1, d_5, d_9\}$ in Fig. 2. In the actual implementation, we use PREORDER<*post*+*dep* instead of POSTORDER<=*post* to take advantage of the use of indexes.

5 Experiments

The proposed method was implemented in Java to compare with the conventional method implemented in C++. The proposed method accesses a relational database via JDBC (Java Database Connectivity). We used 503,989 abstracts from Medline which contain structured information such as authors and Mesh Terms and unstructured information such as titles and abstracts. After preprocessing, the numbers of annotated keywords, categories, and types of (c_j, k_j) were 193185919, 340154, and 14331595, respectively.

Fig. 4 shows the results for all of the documents. The DTM and DB in the figure correspond to the conventional method and the proposed method, respectively. KW and SUB correspond to a category viewer to aggregate a distribution for documents for each keyword and subcategory, respectively. "DTM SUB 1k" shows the computation time for 1,000 documents sampled from all of the documents. Each point (x, y) in the figure means that the method returns the result within x seconds for y% of all of the categories. The ideal method is at the upper left corner. Although the proposed method can return the result for about 89% of the categories within 0.1 second, the conventional method can return for about 60% of the categories for 1,000 sampled documents, and for about 0.01% of categories for 10,000 sampled documents for KW. The result for 11,914 documents containing "cancer" was similar to Fig. 4. As shown in Fig. 4, the proposed method is superior to the conventional method for most of the categories.

According to the empirical 8-second rule saying that a webpage should be loaded within 8 seconds of a request, we can conclude that the better method is the one whose coverage rate in about 10 seconds is better. Fig. 5 focuses from 5 seconds to 10 seconds for the response time and from 99.97% to 100% for the coverage rates for

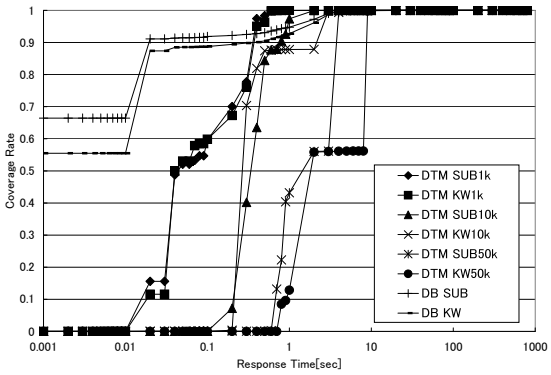

Figure 4: Comp. Time for All Documents

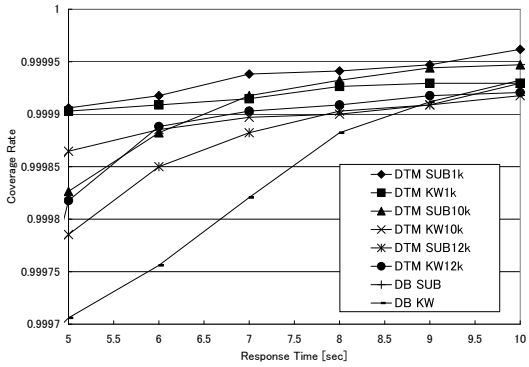

Figure 5: Comp. Time for Documents Containing "cancer"

a set of documents containing "cancer". Fig. 5 shows that the proposed method is inferior to the conventional method within 8-second constraint. Tables 1 and 2 summarize the experimental results for all of the documents and documents containing "cancer". Although the average computation time of the proposed method is lower than for the conventional method, the number of categories for which the method cannot return within 10 seconds may become greater than for the conventional method.

As shown in Table 2 and Fig. 5, the averages of the computation times for the proposed method are superior to the conventional method. However, the number of categories for which the proposed method cannot respond within 10 seconds is greater than for the conventional method. When a user specifies a category node whose preorder is *pre*, an SQL query to aggregate the distribution of the documents containing "cancer" is represented as

```
SELECT A.KEYWORD, COUNT(DISTINCT A.DOCID) AS COUNT
  FROM KEYWORD AS A, KEYWORD AS B
  WHERE A.PREORDER=pre  AND A.DOCID=B.DOCID AND B.KEYWORD='cancer'. (5)
```

The reason why the proposed method requires too computation time for certain categories is the self-join of the table `KEYWORD`, which contains 193,185,919 records. Therefore, we divide the table into multiple tables as follows. Let $id(T)$ be a function which returns a set of document IDs in a table T. We divide T into multiple tables $T_i (i > 2)$ which satisfies $id(T) = \cup_i id(T_i)$ and $id(T_i) \cap id(T_j) = \emptyset$ for $i \neq j$. Since SQL query (5) contains

Table 1: Comp. Time for All Documents

Method	avg. comp. time [sec]	10 sec. †	100 sec. ‡
DTM KW 1k	0.143	1	0
DTM KW 5k	0.284	26	0
DTM KW 10k	0.526	16	0
DTM KW 50k	4.293	75	0
DB KW	0.185	10	0
DTM SUB 1k	0.138	2	1
DTM SUB 5k	0.176	2	0
DTM SUB 10k	0.405	5	1
DTM SUB 50k	2.091	33	0
DB SUB	0.137	20	2

Table 2: Comp. Time for Documents Containing "cancer"

Method	avg. comp. time [sec]	10 sec. †	100 sec. ‡
DTM KW 1k	0.177	24	5
DTM KW 5k	0.355	116	5
DTM KW 10k	0.511	28	7
DTM KW 12k	0.594	27	7
DB KW	0.267	23	0
DTM SUB 1k	0.195	13	1
DTM SUB 5k	0.352	65	0
DTM SUB 10k	0.484	18	1
DTM SUB 12k	0.534	24	2
DB SUB	0.413	706	5

† The number of categories for which the results is not returned within 10 seconds.
‡ The number of categories for which the results is not returned within 100 seconds.

`A.DOCID=B.DOCID` in its WHERE phase, we can avoid joining tables that do not contain the common documents IDs.

Fig. 6 shows the experimental results when we compared the aggregation with `KEYWORD` divided into 10 tables with the conventional method and the aggregation using a single table for `KEYWORD`. Each computation time in the result contains the time to narrow down to the documents containing "cancer" and run a category viewer. By dividing the table, the coverage rate within 10 seconds rises from 99.993% to 99.999% for KW, and from 99.79% to 99.93% for SUB. Although these experiments were run with a single computer, we can easily run on multiple computers, because such commercial database systems support parallelization.

Table 3 shows the computation times for categories in which many different words belong after narrowing down to the documents containing "cancer". This shows that the proposed method can more quickly compute the results for a category compared to the conventional method.

Table 3: Computation Times for Categories to Which Many Different Words Belong

category	# of different words	DTM KW 5k	DTM KW 12k	DB KW
.commonnoun	340,154	231.2 sec.	238.7 sec.	53.0 sec.
.propernoun	7,903	11.8 sec.	10.8 sec.	2.8 sec.
.verb	32,826	73.9 sec.	74.4 sec.	9.7 sec.
.adj	23,629	61.6 sec.	62.4 sec.	7.2 sec.
.rel_vn	1,337,891	171.3 sec.	177.7 sec.	7.6 sec.
.rel_nv	601,256	85.7 sec.	85.3 sec.	3.6 sec.
.rel_vnn	5,439,810	145.6 sec.	162.9 sec.	9.8 sec.
.rel_nvn	2,294,012	99.3 sec.	101.2 sec.	4.4 sec.
.rel_nnv	1,812,501	90.7 sec.	89.5 sec.	3.8 sec.
.affiliation	282,729	53.7 sec.	55.3 sec.	3.2 sec.
.pnsubstance	48,512	66.2 sec.	68.0 sec.	4.3 sec.
.majormesh	89,275	83.6 sec.	85.1 sec.	3.1 sec.
.minormesh	111,761	126.2 sec.	128.5 sec.	7.8 sec.
.chemical	10,146	17.9 sec.	18.2 sec.	3.1 sec.
.genesymbol	15,310	48.1 sec.	48.8 sec.	7.3 sec.
.protein	15,562	57.1 sec.	58.2 sec.	6.7 sec.
.biomedicalterms	91,670	133.7 sec.	137.8 sec.	21.1 sec.

6 Conclusion

In this paper, we proposed an SQL-based method for storing annotated words in a relational database and computing each function of MedTAKMI by SQL. Although the conventional method was implemented in a few thousands of lines of C++ code, the proposed method was implemented in a few lines of SQL and was comparable with the conventional method.

References

[1] S. Arlington, S. Barnett, S. Hughes, and J. Palo: *Pharma 2010: The Threshold of Innovation*, IBM Corporation.

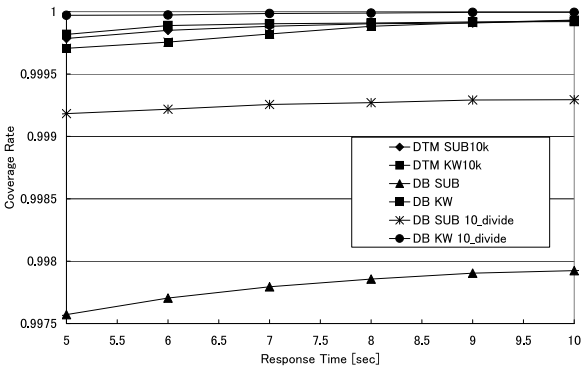

Figure 6: Computation Times for KEYWORD Divided into 10 Tables

[2] M. Hearst: Untangling Text Data Mining, *Proc. of Annual Meeting of the Association for Computer Linguistics*, pp. 3–10, 1999.

[3] D. Swanson: Medical Literature as a Potential Source of New Knowledge, *Bulletin of the Medical Library Association*, 78, No. 1, pp. 29–37, 1990.

[4] V. Brusic and J. Zeleznikow: Knowledge Discovery and Data Mining in Biological Databases, *Knowledge Engineering Review*, 14, No. 3, pp. 257–277, 1999.

[5] D. Swanson and N. Smalheiser: An Interactive System for Finding Complementary Literatures: *Artificial Intelligence*, 91, No. 2, pp. 183–203, 1997.

[6] N. Uramoto, H. Matsuzawa, T. Nagano, A. Murakami, H. Takeuchi, & K. Takeda: A text-mining system for knowledge discovery from biomedical documents. *IBM Systems Journal*, 43(3): pp. 516–533, 2004,

[7] E. Charniak: *Statistical Language Learning*, MIT Press, 1994.

[8] P. F. Dietz: Maintaining order in a linked list, *Proc. of the fourteenth annual ACM symposium on Theory of computing*, pp. 122–127, 1982.