Web Mining for Protein-to-Protein Interaction Information

Hsi-Chieh Lee
Department of Information Management
Yuan Ze University
Taoyuan, Taiwan 320, R.O.C.
and
Department of Information Management
National Kinmen Institute of Technology
Jinning, Kinmen, 892 Taiwan, R.O.C.
E-mail: imhlee@saturn.yzu.edu.tw

Szu-Wei Huang
Department of Information Management
Yuan Ze University
Taoyuan, Taiwan 320, R.O.C.
E-mail: s907736@mail.yzu.edu.tw

Eldon Y. Li *
Department of Management Information Systems
National Chengchi University
Taipei, Taiwan 116, R.O.C.
and
Orfalea College of Business
California Polytechnic State University
San Luis Obispo, CA 93407, U.S.A.
E-mail: eli@calpoly.edu

**Corresponding author*.

Web Mining for Protein-to-Protein Interaction Information

Abstract

This study proposes a mining system for finding protein-to-protein interaction literatures from the databases on the Internet. In this system, we search for discriminating words for protein-to-protein interaction by way of statistics and the results from literatures. A threshold is also evaluated to check if a given literature is related to protein-to-protein interactions. In addition, a keypage-based search mechanism is used to find related papers for protein-to-protein interactions from a given document. To expand the search space and ensure better performance of the system, mechanisms for protein name identification and databases for protein names are also developed.

The system is designed with a web-based user interface and a job-dispatching kernel. Experiments are conducted and the results have been checked by a biomedical expert. The experimental results indicate that by using the proposed mining system, it is helpful for researchers to find protein-to-protein literatures from the overwhelming piece of information available on the biomedical databases over the Internet.

Keywords: Protein-to-protein interactions, rule-based system, Web mining, information retrieval, keypage-based search

Web Mining for Protein-to-Protein Interaction Information

1. Introduction

In recent years, due to advances in information technology, more and more biomedical-related information is available electronically on the Internet. For example, the **MEDLINE** database contains over twelve million citations dating back to the mid-1960's. Therefore, it has become an important issue for mining valuable biomedical information from the literature (Valencia-García, Ruiz-Sánchez, Vicente, Fernández-Breis, & Martínez-Béjar, 2004; Wang, Kuo, Chen, Hsiao, & Tsai, 2005), especially information on the Internet (Hong & Han, 2002). Expert systems and data mining techniques have been used for years in medical diagnosis domain (Chou, Lee, Shao, & Chen, 2004; Alonso, Caraça-Valente, González, & Montes, 2002).

In the post-genomic era, some scientists focus on finding meaningful information of **DNA** or try to use the information of gene sequence in solving problems. However, the spirit of post-genomic era can be view broadly in three ways. The first one is the sequences from DNA level, and the second one is the **Expressed Sequence Tag** (EST) from **RNA** level. The last one is proteome from the protein level. People can use the analyzed information to understand the interaction between each other and discover the meaning behind it. In other words, after decoding the sequence, scientist can analyze the interaction between gene and protein, and understand the role the gene is playing on an organism. It has been shown that the protein and genomics would become the main issue in the post-genomic era (Eisenberg, Marcott, Xenarios, & Yeates, 2000).

Moreover, scientists try to understand the interaction and relation between proteins from biochemistry and gene-related angles. For example, the **Database of Interaction Proteins** (DIP) developed in UCLA (Xenarios, Rice, Salwinski, Baron, Marcotte, & Eisenberg, 2000) has data about over 5900 proteins and 10500 protein-to-protein interactions. Besides **DIP**, there exist many other databases with the collection of the data regarding protein function and pathway. However, if people want to know the relationship between proteins, they have to search different literatures and try to find some relationships. It is considered mission impossible to check on MEDLINE manually where there exist more than fifteen million biomedical citations. It is time-consuming and ineffective. It would be helpful if the job can be processed automatically and the database can be updated as soon as new literatures are available.

Generally, mining the literatures of protein-to-protein interactions requires natural language processing. The literature discussing protein-to-protein interactions does not

contain a language that a computer can understand. As a result, there are two typical approaches in solving the problem. The first approach is transferring the format into a way that computer can understand by natural language processing. For example, in (Ono, Hishigaki, Tanigami, & Takagi, 2001), they brought up an idea to extract biomedical-related information with two steps. The first step is to scan the full document with a protein name dictionary. The second step is to extract content related to protein-to-protein interactions by predefined rules. The second approach is to extract biomedical-related information using statistics. The most typically way of statistics is calculating the frequency of words. In (Marcotte, Xenarios, & Eisenberg, 2001), they used statistics to find 83 words as discriminative words to check whether a paper is discussing protein-to-protein interaction.

In this study, we integrate both approaches mentioned above. The natural language processing techniques and statistics were utilized to process the biomedical-related literatures. Besides checking whether a paper is discussing protein-to-protein interaction, we also find the probable protein names in the target document and help people in finding other related literatures. The proposed mining system is described in detail in the following section and the experimental results are illustrated subsequently.

2. The Proposed System

The literature mining procedures of the proposed mining system is illustrated in Fig. 1. The system starts by feeding a target document (e.g., a protein-to-protein interaction paper) to the system. The first step of the mining system is to calculate the frequency of words and decide whether the target document can be classified as a paper related to protein-to-protein interaction. Next, the mining system will identify the potential protein names in the target document. The third step is to identify the category of interaction of the target document according to the frequencies of words in it. The final step is to compose keywords automatically and send the query to the **PubMed**. Once the query results are ready, the system will then retrieve the related literatures from the Internet (http://www.ncbi.nlm.nih.gov/PubMed/). The detailed description of each step is shown below.

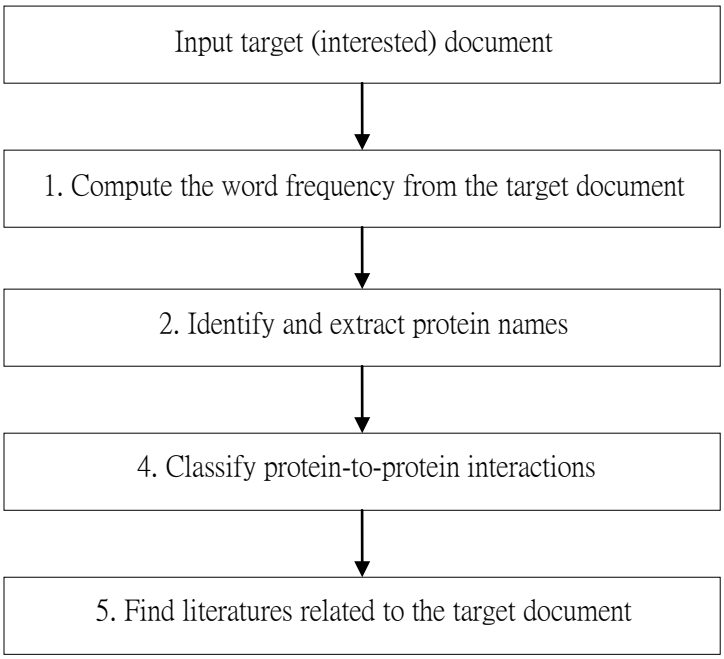

Fig. 1. The literature mining procedures of the proposed system

Step 1. Compute the frequency of the word tokens in the target (interested) document

In the proposed mining system, we used 20 words that have the most discriminate capability. To ensure that they are the most discriminate words, we retrieved 2203 abstracts from Database of Interacting Proteins (http://dip.doe-mbi.ucla.edu/) (Xenarios, Rice, Salwinski, Baron, Marcotte, & Eisenberg, 2000) – a well-known database dedicated for data about protein interaction. We calculated the frequency of the words in these documents and took them as the most discriminating words (stop words were excluded). When the researcher inputs the target document, the system will calculate the frequencies of the 20 words. If the frequencies are high enough, this target document is considered as discussing protein interactions by the proposed system. In a different study, Marcotte, Xenarios, and Eisenbert (2001) found statistics of 65807 literatures and derived 83 discriminate words.

Step 2. Identify and extract protein names

In biology domain, it is not uncommon to have new findings everyday. When there is a new finding, the researcher will give it a new name to discriminate it from others. As a result, new protein names are created rapidly which makes it difficult to identify all the protein names. In general, a protein name can be classified into one of the following types of word.

- Single words with upper case letters, numerical figures, and non-alphabetical letters. Mostly derived from gene name.

- Compound words with upper case letters, numerical letters, and non-alphabetical letters.
- Single word with only lower case letters.

The first two types of protein names are more commonly found in research papers while protein names of type 3 are rarely found (Fukuda, Tsunoda, Tamura, & Takagi, 1998).

Another difficulty of identifying a protein name is the inconsistency of naming convention. It is also not uncommon to find the same protein being expressed differently by different researchers, depending on the researchers' styles. For example, the term "epidermal growth factor receptor", some researchers describe it directly while others use abbreviations as "EGF receptor" or "EGFR". That is, even the abbreviation might be expressed differently. The following example demonstrates some variation of expression for a protein function (Fukuda, Tsunoda, Tamura, & Takagi, 1998).

- the Ras guanine nucleotide exchange factor Sos
- the Ras guanine nucleotide releasing protein Sos
- the Ras exchanger Sos

As mentioned above, protein names and their expressions are mainly decided by researchers' style. Therefore, it becomes a challenge to find them in the literature. The general way of finding protein names is constructing a dictionary of protein names manually and uses the dictionary for pattern matching (Thomas, Milward, Ouzounis, Pulman, &. Carroll, 2000). Another way of finding the protein names is to check the words that are used often around the literature. In addition, it is also applicable to analyze the characteristics of words and try to find protein names in the literature. In the proposed system, we integrate the three approaches mentioned above for identifying the protein names in the literature. The detailed process of extracting protein names is shown in Fig. 2 and the discussion follows.

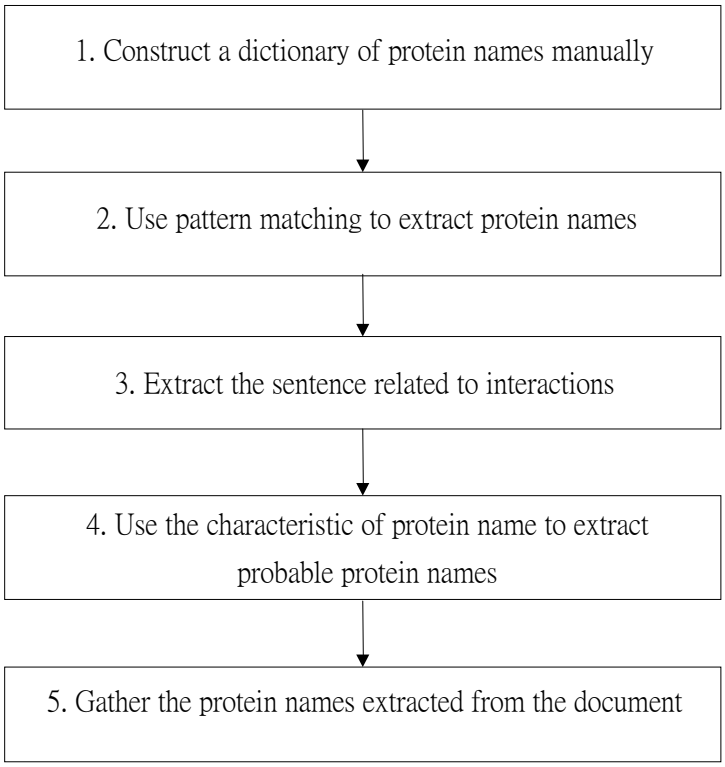

Fig. 2. Identification and extraction of protein names from a document

1. To construct a dictionary of protein names for the system, we retrieve the protein names from Protein Name Abbreviation Dictionary, (http://pnad.ontology.ims.u-tokyo.ac.jp/search/php/search.php) and then add other known protein names manually.
2. Using pattern-matching approach to compare every word in the literature and extract the protein names from the literature.

A set of word patterns is commonly used for recognition of protein-to-protein interaction, namely, 'interact', 'bind', 'associate', and 'complex'. Unfortunately, they might appear in different forms. As a result, we try to extract these word patterns using regular expression and the rule shown below. In addition, the Brill POS tagger package (Brill, 1994) is used to analyze the target test.

Rule: If the sentence matches the following part-of-speech pattern as indicated by regular expression, it can be extracted.

interact .* | complex .* | bind .* | associate .*

When getting the sentence from the target document, the Brill POS tagger extracts the words which are tagged as NN or NNP (see Table 1). Once the sentence contains word patterns of protein-to-protein interaction, we treat the words that are tagged as NN or NNP in the sentence to be potential protein name. To further eliminate words that are not related, for example, words like 'domain', 'function', the Porter's

Stemming Algorithm (Porter, 1980) is used. This algorithm processes suffix of words. If the word is plurality or other type, it will be transformed back to the original word. We will try to compare the words extracted to those on **WordNet** and try to eliminate words that are not interested. The Brill POS tagger definition of symbols is shown in Table 1.

Table 1. Definition of symbols

Symbol	Definition
,	Comma
:	Colon
;	Semi-colon
CC	Coordinating conjunction
DT	Determiner
IN	Preposition or subordinating conjugation
JJ	Adjective
NN	Noun, singular or mass
NNP	Proper noun, singular
NNS	Noun, plural
P(1/2)	Phrase
P(3/4/5)	Phrase without verb
VB(1/2)	Verb
VBN	Verb, past participle
VBZ	Verb, 3rd person singular present

3. In this step, we adopt Fukuda's (1998) approach to extracting protein names with the following characteristics.
   a. Based on characteristics of protein name, extract the words with upper cases, numerical figures, and / or special symbols.
   b. Use the rules listed below to filter out improbable words.
      - Exclude words whose length is more than 9 characters and consists of "-" and lower cases.
      - Exclude words in which more than half of its character string consists of special symbols.
      - Exclude words related to numbers such as units. Eight words（aa, AA, fold, bp, nM, microM, %, UV）are registered as units.
      - Exclude words that agree to the reference template prepared beforehand.
   c. If extracting words are in common use, they can be filtered out by words in

WordNet.

d. The words that are left over are the probable protein names.

4. Integrating the probable protein names found using three different approaches: comparing with database, extracting by regular expression, and extracting by characteristics of protein names.

Step 3. Classification of protein-to-protein interactions

After extracting the protein names, we further try to classify the interaction discussed in the target document into a category. Based on the words commonly used to describe interaction and by checking the words that are discriminated by the statistics, four words（interact, associate, complex, bind）were used. According to the frequencies of the four words, the highest one will be considered as the category of interaction that is discussed in the target document.

Step 4. Finding related literature of the target document

Once the protein names and the category of interaction are identified, they will be used as input for search on the search engine of PubMed (URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=PubMed) for retrieving the candidate research papers related to the target document which has been entered into the proposed mining system by the researcher. The results from PubMed are URLs of candidate research papers. The system will then retrieve the papers from the Internet and then extracting features from each of them and calculate the similarity with the target document using SimNet's **kernel function** (Lee, Dagli, Ercal, & Ozbayoglu, 1995). **SimNet** is a neuro-fuzzy system that integrates the neural network architecture and fuzzy theory. The idea of SimNet was first introduced for character recognition and has later been used for speaker identification. It combines a neural network structure with the subsethood concept of fuzzy logic to produce a rapid data clustering system that works similar to **Adaptive Resonance Theory** and **Self-Organizing Maps**. It has two neural network architectures (Lee, Dagli, Ercal, & Ozbayoglu, 1995), namely, a 2-layer unsupervised learning model and a 3-layer supervised learning model. In this study, only the similarity measurement equation is used to compare the similarity of documents. We use the algorithm to calculate similarity of documents and get the most related documents. The similarity measurement equation of SimNet is shown below.

$$MD(I,W)=\sqrt{\frac{(\sum_i \min(I_i,W_i))^2}{(\sum_i W_i)(\sum_i I_i)+\varepsilon}}$$, where ε is a non-zero minimal

This procedure turns the keyword-based search into a keypage-based search that reduces the amount of un-related information significantly.

3. Illustration of source programs

There are a number of program modules in the proposed literature mining system. For instance, the document input module, word frequency counting module, protein name identification module, protein name extracting module, protein classification module, Internet document retrieving module, protein name extracting module, interaction sentence extracting module, and etc. Four of these modules are illustrated in the Appendix. These include: 1. Extract protein names according to the database, 2. Extract words with protein name characteristics, 3. Retrieve potential protein names using regular expression, and 4. Retrieve similar documents.

4. Experimental Results

In order to decide whether the target document fell into the category that discussed **protein-to-protein interaction**, we started by finding the 20 most **discriminating words**. We downloaded 2203 abstracts from the DIP and then calculated the word frequencies. The average number of words in an abstract was 196 words. The top 20 most frequently used words were found as shown in Table 2. As we could see in Table 2, the word patterns, {interact, binding, complex} that were commonly used in protein-to-protein interaction papers were also included.

Table 2. The 20 most discriminating words

Discriminating word	Frequency	Discriminating word	Frequency
complex/complexes	1845	binding	1221
interact/interaction /interactions/interacts	2449	component	249
		required	626
two-hybrid	448	suggesting	256
With	3997	From	907
protein/proteins	4434	demonstrate	251
Function	607	Kinase	803
Domain	1039	essential	444

Since all the literatures in DIP were considered to have the topic regarding the protein-to-protein interaction. Therefore, we tried to analyze the behavior of the literatures in DIP by calculating the **Matching Degree** of each abstract in DIP with the overall frequency vector as shown in Table 2. The Matching Degree (MD) is the kernel function in SimNet (Lee, Dagli, Ercal, & Ozbayoglu, 1995). The similarity measurement equation MD is shown below.

$$MD(I,W)=\sqrt{\frac{(\sum_i \min(I_i,W_i))^2}{(\sum_i W_i)(\sum_i I_i)+\varepsilon}}$$, where ε is a non-zero minimal

In this equation, the vector "*I*" means the **word frequency** that is shown in Table 2 and notice that it has been normalized. The vector "*W*" means the frequency of 20 discriminate words of each abstract found in DIP. The detailed calculating steps are demonstrated in Fig. 3.

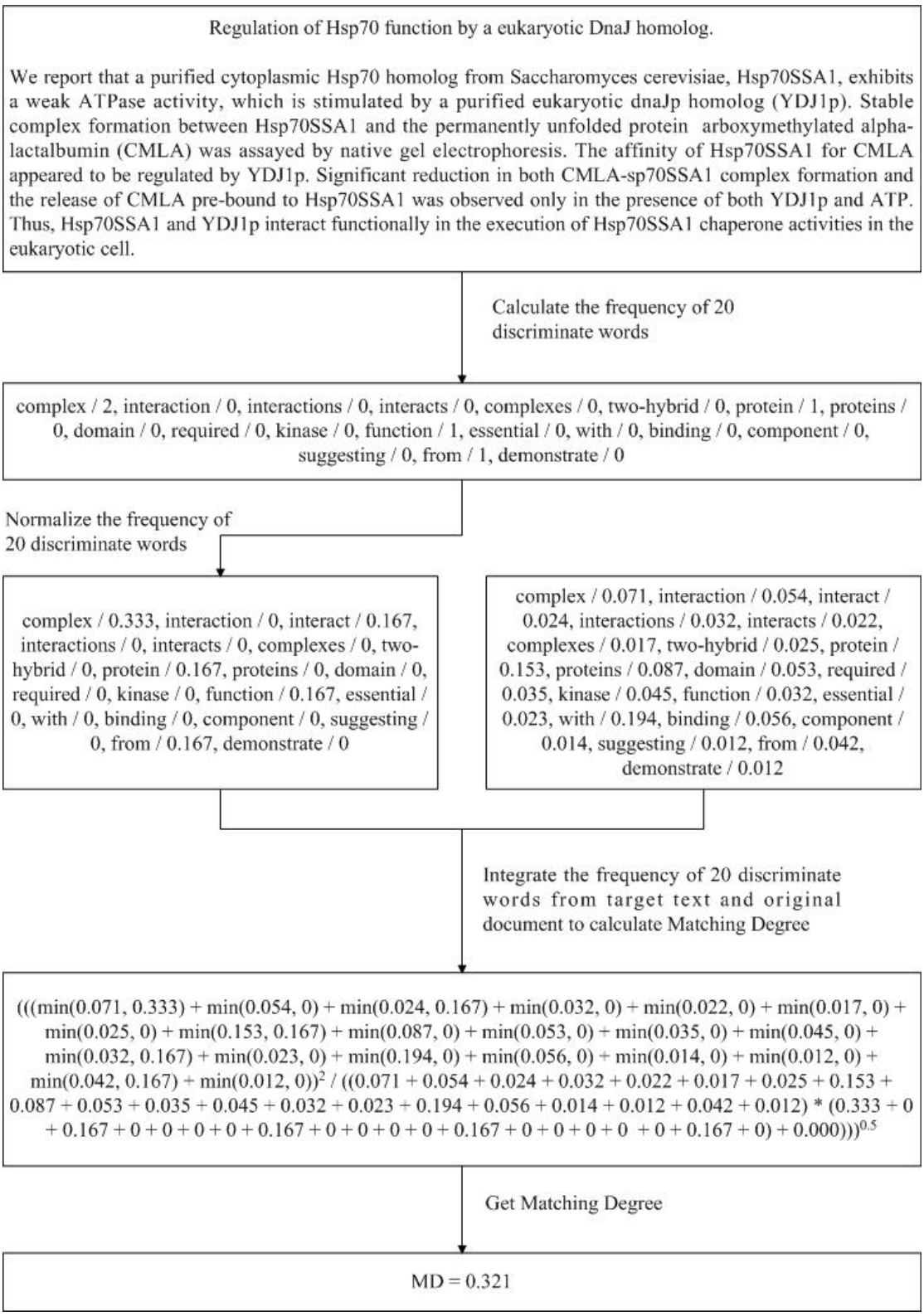

Fig. 3. The steps of calculating the Matching Degrees

The experiment found the Matching Degrees (MDs) of the 2203 abstracts from DIP. The experimental results shown in Fig. 4 reveal that the values of MD were in the range between 0.02 and 0.74 and the average value of MD was 0.457. If the

threshold was set to 0.02, all the literatures would be accepted as papers of protein-to-protein interaction and the recall rate would be maximized. However, the precision rate did not appear to be good. When we set the threshold to the average value 0.457, 45% of the literatures would be accepted and 55% would be dropped.

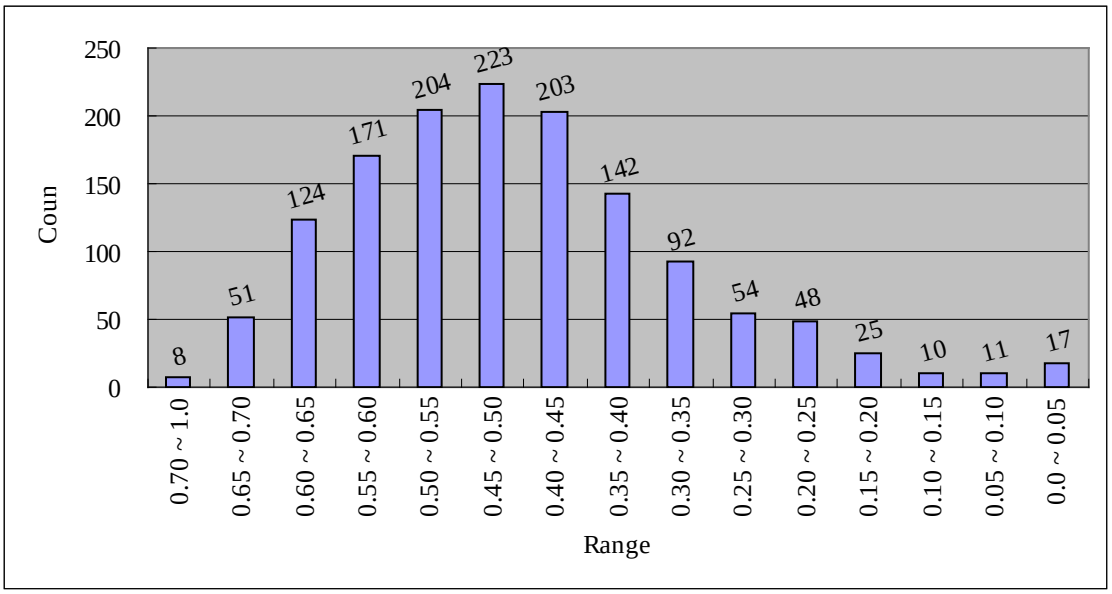

Fig. 4. The Matching Degree (MD) of literatures in DIP.

To generalize the results for the MD threshold and to find the precision and recall rate with different thresholds, we randomly chose 535 literatures from DIP that represented the research papers of protein-to-protein interaction. Moreover, we used the four keywords, ‘interact’, ‘associate’, ‘bind’, and ‘complex’ as inputs to MEDLINE. There were 30277 abstracts obtained by the MEDLINE search using these keywords as the **MeSH** terms. With a random selection and a duplication-elimination process, 10531 abstracts were retrieved from MEDLINE. The MDs were calculated from a total of 11066 abstracts, in which, 535 abstracts derived from DIP and 10531 abstracts derived from MEDLINE. The detailed calculating steps were demonstrated in Fig. 3 and the MDs of these abstracts were shown in Fig. 5. It could be observed that the test results of the 11066 abstracts shown in Fig. 5 looked similar to the test results of the 2203 abstracts in Fig. 4.

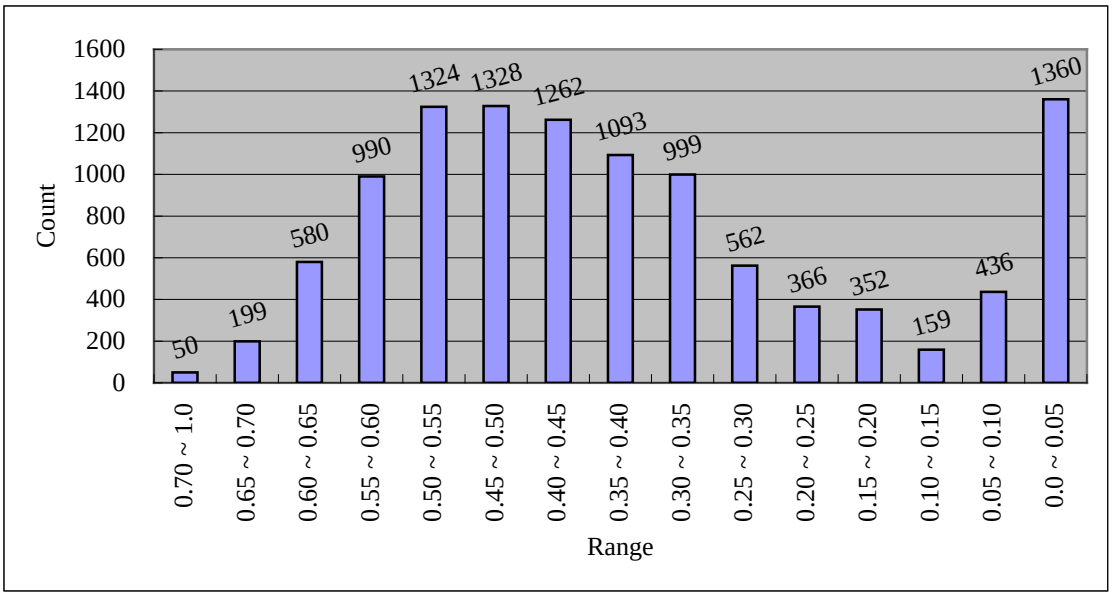

Fig. 5. The Matching Degree (MD) of 11066 abstracts

To test if the MD thresholds could be applied to those literatures that were not included in the set of the 11066 abstracts, a new experiment was conducted and the results for the precision and recall rate were verified by an expert of protein chemistry. This expert was a physician from a local hospital who held a Ph.D. degree from an institution in the U.S. and specialized in Protein Chemistry, Matrix Biology, and Glycobiology. A total of 740 abstracts were used for calculating the Matching Degree. Out of the 740 abstracts, 100 of them were selected randomly from DIP, and the rest of them were selected randomly from MEDLINE that were considered not literatures of protein-to-protein interaction. The experimental results of the 740 abstracts were shown in Fig. 6.

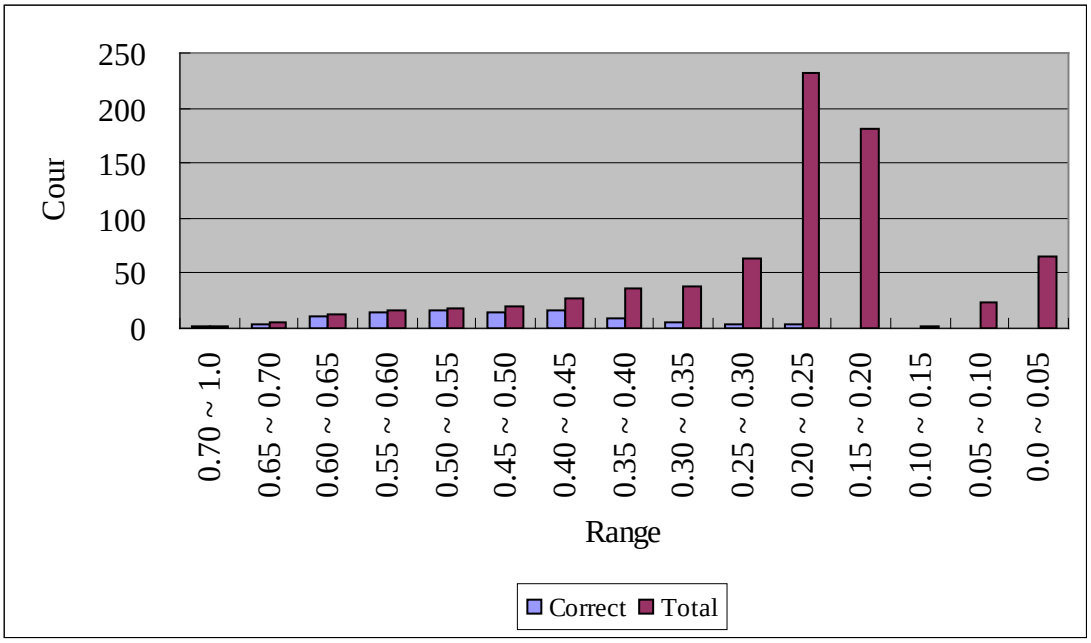

Fig. 6. The Matching Degree (MD) of 740 abstracts

The experimental results for the 740 abstracts were transformed into the precision-recall chart as shown in Fig. 7. When the MD threshold was set to 0.35, 87% abstracts would be correctly classified as papers of protein-to-protein interaction. However, the precision rate was not as high.

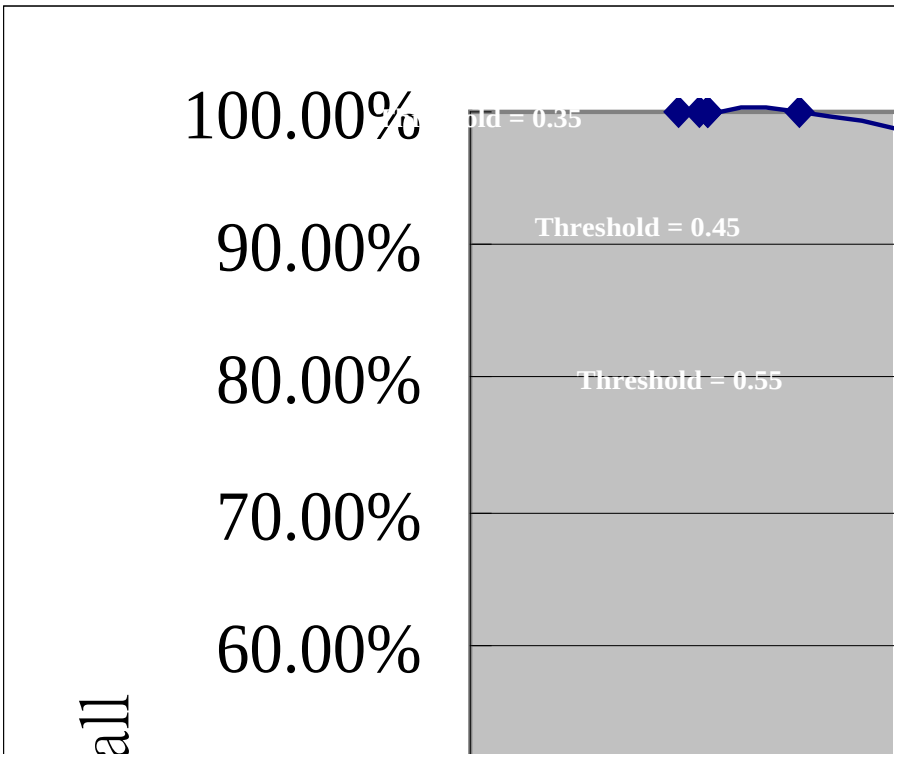

Fig. 7. Precision-recall chart with different thresholds

There was a tradeoff between the recall rate and the precision as shown in Fig. 7. In other words, to get higher precision, the MD threshold must be higher than 0.45. Conversely, to get higher recall rate, the MD threshold must be lower than 0.35. The selection of MD threshold mainly depended on the purpose of using the proposed mining system. For example, to retrieve more protein-to-protein interaction papers that were related to the target document, a lower threshold was preferred. In contrast, higher MD threshold was suggested if less but accurate results were demanded. Note that the mining job requested in this study was a non-trivial computation-intensive and communication-demanding task. The main reason was that the proposed system needed to analyze the target document in order to find the probable protein names and then send the query to the Internet search engines. In addition, each and every document found should also be retrieved from the Internet and the analysis process would be applied to each and every document again. Finally, computation for MDs between the target document and the documents retrieved from the Internet should be done to find the similarity. Fortunately, this time-consuming process could be resolved using a parallel and distributed environment proposed in (Chen, & Lee, 2002).

5. Managerial Implications to Bioinformatics

As human's genetic disease has been proven to relate to gene variation, genomic sequence has been used for searching the genetic sequence of disease, researching the way of prevention, diagnosis, and treatment. It also helps people in understanding the relation between environment and evolution by comparing cross gene of species. Besides analyzing the gene, the analysis of protein is another important approach. Moving the research from the one dimension of DNA into the three dimension of protein, people can understand the protein's structure and function. We can also use this information to develop new medicine or new prevention mechanism for diseases. During the course of this research, we must understand the interaction and relation between proteins in biochemistry and gene-related ways. In order to conserve the resources demanded by experimental studies, we can find the related experimental outcomes published in the literature. However, the information available on the extant literature is overwhelming. For example, just the MEDLINE (see http://www.ncbi.nlm.nih.gov/Literature/) alone contains over 15,000,000 biomedical journal citations. Likewise, Database of Interaction Proteins (Xenarios, Rice, Salwinski, Baron, Marcotte, & Eisenberg, 2000) alone contains over 5,900 proteins and 10,500 interactions data about protein-to-protein interaction. It is impossible for a human being to search through all these pieces of data. Therefore, our proposed

mining system will help researcher in finding related papers of protein interactions from the literatures available over the Internet, with a simple input of an interested document. The proposed mining system is expected to reduce the time and effort significantly for researchers in the stage of literature review.

6. Conclusions

In this study, a mining system has been proposed for finding protein-to-protein interaction literatures from the database on the Internet. Through this system, we found discriminating words for protein-to-protein interaction by way of statistics and results from literatures. A threshold was also evaluated using the MD function of SimNet to check if a given literature was related to protein-to-protein interactions. In addition, a keypage-based search mechanism was used to find papers related to protein-to-protein interactions from a given document. Moreover, to expand the search space and ensure better performance of the system, both mechanisms for protein name identification and databases for protein names were used. Experiments were conducted and the results were checked by an expert of protein chemistry.

The findings of the study suggest that to get precision as high as 90%, the MD threshold can be adjusted to a value higher than 0.45. In addition, to get recall rate as high as 90%, the MD threshold can be adjusted to be lower than 0.35. These indicate that by using the proposed mining system, it is helpful for researchers to find protein-to-protein literatures from the overwhelming amount of information available on the biomedical database over the Internet. Moreover, because of the nature of the architecture, the results can be derived in a reasonable time with the help of the parallel and distributed mechanism.

Finding related a literature of protein interactions from the Internet using simply an interested document is the major contribution of this study. It keeps the researchers from inputting many possible keywords and check hundreds and thousands of results returned from general search engines. However, there are two major concerns in this research which needs further investigation and study in the future. The first one is the prediction of protein names. As long as there exists no “golden” standard for protein naming convention, there exists the problem of “guessing” protein name available in the literatures. The second concern for implementing such a mining system is the performance of the system which is usually measured by precision and recall rate. To measure the performance, one may build a benchmarking database in any target domain (e.g., DIP for protein interactions) which is a job requires considerably involvement of domain experts.

Acknowledgement

This research is supported partially under the grants of NSC91-2745-P-155-003 and NSC93-2745-E-155-010-URD, Taiwan, R.O.C.

References

1. Brill, E. (1994). A report of recent progress in transformation-based error-driven learning. Proc. {ARPA-94} Human Language Technology Workshop, Princeton, NJ, USA.
2. Chen, P. W., & Lee, H. C. (2002). C^2AS: An agent-based distributed and parallel processing virtual machine. Seventh International Conference on Applications of High-Performance Computers in Engineering (HPC 2002), September 23–25, Bologna, Italy.
3. Chou, S. M., Lee, T. S., Shao, Y. E., & Chen, I. F. (2004). Mining the breast cancer pattern using artificial neural networks and multivariate adaptive regression splines. Expert Systems with Applications, 27(1), 133-142.
4. Eisenberg, D., Marcott, E. M., Xenarios, I., & Yeates, T. O., (2000). Protein function in the post-genomic era. Nature, 405(6788), 823-6.
5. Alonso, F., Caraça-Valente, J. P., González, A. L., & Montes, C. (2002). Combining expert knowledge and data mining in a medical diagnosis domain. Expert Systems with Applications, 23(4), 367-375.
6. Fukuda, K., Tsunoda, T., Tamura, A., & Takagi, T. (1998). Toward information extraction: Identifying protein names from biological papers. Proceedings of the Pacific Symposium on Biocomputing (PSB'98, pp. 707-718).
7. Hong, T., & Han, I. (2002). Knowledge-based data mining of news information on the Internet using cognitive maps and neural networks. Expert Systems with Applications, 23(1), 1-8.
8. Lee, H. C., Dagli, C. H., Ercal, F., & Ozbayoglu, A. M. (1995). SimNet: A parallel neuro-fuzzy paradigm for data clustering. OAI Neural Networks Symposium and Workshop (OAINN '95), Athens, Ohio, USA.
9. Marcotte, E. M., Xenarios, I., & Eisenberg, D. (2001). Mining literature for protein-to-protein interactions. Bioinformatics, 17(4), 359-363.
10. Ono, T., Hishigaki, H., Tanigami, A., & Takagi, T. (2001). Automated extraction of information on protein-to-protein interactions from the biological literature. Bioinformatics, 17(4), 155-161.
11. Porter, M. F. (1980). An algorithm for suffix stripping. Program, 14(3), 130-137.
12. Valencia-García, R., Ruiz-Sánchez, J.M., Vicente, P.J.V., Fernández-Breis, J.T., & Martínez-Béjar, R. (2004). An incremental approach for discovering medical knowledge from texts. Expert Systems with Applications, 26(3), 291-299.
13. Thomas, J., Milward, D., Ouzounis, C., Pulman, S., &. Carroll, M. (2000). Automatic extraction of protein interactions from scientific abstracts. Pac. Symp. Biocomput (pp. 541-552).
14. Xenarios, I., Rice, D. W., Salwinski, L., Baron, M. K., Marcotte, E. M., &

Eisenberg, D. (2000). DIP: The database of interacting proteins. Nucleic Acids Res. 28(1), 289-91.

15. Wang, H. C., Kuo, H. C., Chen, H. H., Hsiao, Y. Y, & Tsai, W. C. (2005). KSPF: Using gene sequence patterns and data mining for biological knowledge management. Expert Systems with Applications, 28(3), 537-545.

Appendix: Illustration of source programs

In this section, the source codes for four important modules of the proposed literature mining system are provided and commented accordingly. These include: 1. Extract protein names according to the database, 2. Extract words with protein name characteristics, 3. Retrieve potential protein names using regular expression, and 4. Retrieve similar documents.

1. Extract protein names according to the database

```
//Try to find protein name in the target document according to the data in the database.
  pSet = new CProtein;
  ASSERT_VALID( pSet );
  pSet->Open();
  for( i=0; i<(DWORD) pKeyWord->GetCount(); i++)
  {
       pSet->m_strFilter.Format("%s%s%s","protein='",(*pKeyWord)[i],"'");
       if(!pSet->Requery())
            break;
       if( !pSet->IsEOF() )
       {
            pProteinName->Add( (*pKeyWord)[i]);
       }
  }
  pSet->Close();
  delete pSet;

//Add the protein found into variable m_sDatabaseProtein.
  for( i=0; i<(DWORD)pProteinName->GetCount(); i++)
  {
       for ( bResult=TRUE, ListProteinIterator = m_lpProtein.begin(); ListProteinIterator !=
m_lpProtein.end(); ++ListProteinIterator )
       {
            if ( (*ListProteinIterator)->szProteinName == pProteinName->GetAt(i) )
            {
                 bResult = FALSE;
                 pProtein = (LPPROTEIN) (*ListProteinIterator);
                 break;
            }
       }

       if ( bResult )
       {
            pProtein = new PROTEIN;
            ASSERT( pProtein != NULL );
            pProtein->szProteinName = pProteinName->GetAt(i);
            pProtein->dwCount = 1;
       }
       else
            continue;
       for( bResult=TRUE, j=0; j<(DWORD)m_sProteinName.GetCount(); j++ )
       {
            if( pProteinName->GetAt(i) == m_sProteinName.GetAt(j) )
            {
```

```
                    bResult = FALSE;
                    pProtein->dwCount++;
                    break;
               }
          }
          if( bResult )
          {
               m_sDatabaseProtein.Add( pProteinName->GetAt(i) );
               m_sProteinName.Add( pProteinName->GetAt(i) );
               m_lpProtein.push_back( pProtein );
          }
     }

//Extract words that contains uppercase letter, “/” sign, or “-“ sign.
     for( i=0; i<(DWORD)pKeyWord->GetCount(); i++)
     {
          bResult = TRUE;
          TRACE( "%s\n", pKeyWord->GetAt(i) );
          for( j=0; j<(DWORD)pKeyWord->GetAt(i).GetLength(); j++)
          {
               if(pKeyWord->GetAt(i).GetAt(j) >= 'A' && pKeyWord->GetAt(i).GetAt(j) <= 'Z')
               {
                    bResult = TRUE;
                    break;
               }
               else if( pKeyWord->GetAt(i).GetAt(j) == '/')
               {
                    bResult = TRUE;
                    break;
               }
               else if( pKeyWord->GetAt(i).GetAt(j) == '-')
               {
                    bResult = TRUE;
                    break;
               }
               else
                    bResult = FALSE;
          }
          if( !bResult )
          {
               pKeyWord->RemoveAt(i);
               i--;
          }
     }

//Remove words without required characteristics using filter.
     for( i=0; i<(DWORD)pKeyWord->GetCount(); i++)
     {
          bResult = TRUE;
          dwCount = 0;
          if( pKeyWord->GetAt(i).GetLength() > 9)
          {
               for( j=0; j<(DWORD)pKeyWord->GetAt(i).GetLength(); j++)
               {
                    if( pKeyWord->GetAt(i).GetAt(j) == '-')
                    {
                         bResult = FALSE;
                         break;
                    }
```

```
                }
                if( !bResult )
                {
                     for( j=0; j<(DWORD)pKeyWord->GetAt(i).GetLength(); j++)
                     {
                          if( pKeyWord->GetAt(i).GetAt(j) >= 'A' && pKeyWord->GetAt(i).GetAt(j)
<= 'Z')
                          {
                               bResult = TRUE;
                               break;
                          }
                     }
                }
                if( !bResult )
                {
                     pKeyWord->RemoveAt(i);
                     i--;
                }
           }
           else
           {
                for( j=0; j<(DWORD)pKeyWord->GetAt(i).GetLength(); j++)
                {

  //Count the numbers of special characters.
                     switch( pKeyWord->GetAt(i).GetAt(j) )               {
                     case '<':case
',':case'?':case'~':case'!':case'@':case'#':case'$':case'^':case'&':case'*':
                     case '(':case ')':case '+':case '|':case '{':case
'}':case'>':case'-':case'\n':
                          dwCount++;
                          break;
                     case'%':
                          bResult = FALSE;
                          break;
           case'1':case'2':case'3':case'4':case'5':case'6':case'7':case'8':case'9':case'0':
                          dwNumber++;
                          break;
                     }
                }
                if( pKeyWord->GetAt(i).GetLength() > 3
&&pKeyWord->GetAt(i).Find("aa",pKeyWord->GetAt(i).GetLength()-2) != -1)
                     bResult = FALSE;
                if( pKeyWord->GetAt(i).GetLength() > 3 &&
pKeyWord->GetAt(i).Find("AA",pKeyWord->GetAt(i).GetLength()-2) != -1)
                     bResult = FALSE;
                if( pKeyWord->GetAt(i).GetLength() > 5 &&
pKeyWord->GetAt(i).Find("fold",pKeyWord->GetAt(i).GetLength()-4) != -1)
                     bResult = FALSE;
                if( pKeyWord->GetAt(i).GetLength() > 3 &&
pKeyWord->GetAt(i).Find("bp",pKeyWord->GetAt(i).GetLength()-2) != -1)
                     bResult = FALSE;
                if( pKeyWord->GetAt(i).GetLength() > 3 &&
pKeyWord->GetAt(i).Find("nM",pKeyWord->GetAt(i).GetLength()-2) != -1)
                     bResult = FALSE;
                if( pKeyWord->GetAt(i).GetLength() > 7 &&
pKeyWord->GetAt(i).Find("microM",pKeyWord->GetAt(i).GetLength()-6) != -1)
                     bResult = FALSE;
                if( pKeyWord->GetAt(i).GetLength() > 3 &&
```

```
pKeyWord->GetAt(i).Find("UV",pKeyWord->GetAt(i).GetLength()-2) != -1)
                    bResult = FALSE;
               if( dwCount > (DWORD)pKeyWord->GetAt(i).GetLength()/2 || bResult == FALSE ||
dwNumber > (DWORD)pKeyWord->GetAt(i).GetLength()/2)
               {
                    pKeyWord->RemoveAt(i);
                    i--;
               }
          }
     }

//Transfer words into their origins using porter stemmer.
symbol sBuffer[1024] = {0};
TCHAR  sBufferFinish[1024] = {0};
SN_env * pSNenv = NULL;
     pSNenv = porter_create_env();
     for ( i=0; i<(DWORD)pKeyWord->GetCount(); i++ )
     {
          ::FillMemory( sBuffer, 1024, 0 );
          CopyMemory( sBuffer, (LPCTSTR) (pKeyWord->GetAt(i)), pKeyWord->GetAt(i).GetLength()
> 1023 ? 1023 : pKeyWord->GetAt(i).GetLength() );
          TRACE( "orig = %s ", pKeyWord->GetAt(i) );
          SN_set_current( pSNenv, pKeyWord->GetAt(i).GetLength() > 1023 ? 1023 :
pKeyWord->GetAt(i).GetLength(), sBuffer );
          porter_stem( pSNenv );
          ::FillMemory( sBufferFinish, 1024, 0 );
          CopyMemory( sBufferFinish, pSNenv->p, pSNenv->l > 1023 ? 1023 : pSNenv->l );
          TRACE( "%s\n", sBufferFinish );
          (*pKeyWord)[i] = sBufferFinish;
     }

     porter_close_env( pSNenv );

  //Remove words that can be found in WordNet.
     pNoun = new CNoun;
     ASSERT_VALID( pNoun );
     pNoun->Open();
     for( i=0; i< (DWORD)pKeyWord->GetCount(); i++)
     {
          if( !pKeyWord->IsEmpty() && pKeyWord->GetAt(i).Find("'") == -1)
          {
               szTemp = (*pKeyWord)[i];
               szTemp.Trim( );
               szTemp.Trim(",%&.@#*!~?");
               pNoun->m_strFilter.Format("%s%s%s","noun='",szTemp.MakeLower(),"'");
               if(!pNoun->Requery())
                    break;
               if( !pNoun->IsEOF() )
               {
                    pKeyWord->RemoveAt(i);
                    i--;
               }
          }
     }
     pNoun->Close();
     delete pNoun;

  //Add the protein found into variables m_sCProtein and m_sProteinName.
     for( i=0; i<(DWORD)pKeyWord->GetCount(); i++ )
```

```
    {
        for ( bResult = TRUE, ListProteinIterator = m_lpProtein.begin();
ListProteinIterator != m_lpProtein.end(); ++ListProteinIterator )
        {
            if( (*ListProteinIterator)->szProteinName == pKeyWord->GetAt(i) )
            {
                bResult = FALSE;
                pProtein = (LPPROTEIN) (*ListProteinIterator);
                break;
            }
        }
        if( bResult )
        {
            pProtein = new PROTEIN;
            ASSERT( pProtein != NULL );
            pProtein->szProteinName = pKeyWord->GetAt(i);
            pProtein->dwCount = 0;
        }
        for(bResult = TRUE, j=0; j<(DWORD)m_sProteinName.GetCount(); j++ )
        {
            if( pKeyWord->GetAt(i) == m_sProteinName[j] )
            {
                bResult = FALSE;
                pProtein->dwCount++;
                break;
            }
        }
        if( bResult )
        {
            m_sProteinName.Add( pKeyWord->GetAt(i) );
            pProteinName->Add( pKeyWord->GetAt(i) );
            m_sCProtein.Add( pKeyWord->GetAt(i) );
            m_lpProtein.push_back( pProtein );
        }
    }
```

2. Extract words with protein name characteristics

```
//Extract words that contain uppercase letter, “/” sign, or “-“ sign.
    for( i=0; i<(DWORD)pKeyWord->GetCount(); i++)
    {
        bResult = TRUE;
        TRACE( "%s\n", pKeyWord->GetAt(i) );

        for( j=0; j<(DWORD)pKeyWord->GetAt(i).GetLength(); j++)
        {
            if(pKeyWord->GetAt(i).GetAt(j) >= 'A' && pKeyWord->GetAt(i).GetAt(j) <= 'Z')
            {
                bResult = TRUE;
                break;
            }
            else if( pKeyWord->GetAt(i).GetAt(j) == '/')
            {
                bResult = TRUE;
                break;
            }
            else if( pKeyWord->GetAt(i).GetAt(j) == '-')
            {
                bResult = TRUE;
```

```
                    break;
               }
               else
                    bResult = FALSE;
          }
          if( !bResult )
          {
               pKeyWord->RemoveAt(i);
               i--;
          }
     }

  //Remove atypical words using filter.
     for( i=0; i<(DWORD)pKeyWord->GetCount(); i++)
     {
          bResult = TRUE;
          dwCount = 0;
          if( pKeyWord->GetAt(i).GetLength() > 9)
          {
               for( j=0; j<(DWORD)pKeyWord->GetAt(i).GetLength(); j++)
               {
                    if( pKeyWord->GetAt(i).GetAt(j) == '-')
                    {
                         bResult = FALSE;
                         break;
                    }
               }
               if( !bResult )
               {
                    for( j=0; j<(DWORD)pKeyWord->GetAt(i).GetLength(); j++)
                    {
                        if( pKeyWord->GetAt(i).GetAt(j) >= 'A' && pKeyWord->GetAt(i).GetAt(j)
<= 'Z')
                         {
                              bResult = TRUE;
                              break;
                         }
                    }
               }
               if( !bResult )
               {
                    pKeyWord->RemoveAt(i);
                    i--;
               }
          }
          else
          {
               for( j=0; j<(DWORD)pKeyWord->GetAt(i).GetLength(); j++)
               {

  //Count the numbers of special characters.
                    switch( pKeyWord->GetAt(i).GetAt(j) )
                    {
                    case '<':case
',':case'?':case'~':case'!':case'@':case'#':case'$':case'^':case'&':case'*':
                    case '(':case ')':case '+':case '|':case '{':case
'}':case'>':case'-':case'\n':
                         dwCount++;
                         break;
```

```
                    case'%':
                        bResult = FALSE;
                        break;

    case'1':case'2':case'3':case'4':case'5':case'6':case'7':case'8':case'9':case'0':
                        dwNumber++;
                        break;
                    }
                }
                if( pKeyWord->GetAt(i).GetLength() > 3
&&pKeyWord->GetAt(i).Find("aa",pKeyWord->GetAt(i).GetLength()-2) != -1)
                    bResult = FALSE;
                if( pKeyWord->GetAt(i).GetLength() > 3 &&
pKeyWord->GetAt(i).Find("AA",pKeyWord->GetAt(i).GetLength()-2) != -1)
                    bResult = FALSE;
                if( pKeyWord->GetAt(i).GetLength() > 5 &&
pKeyWord->GetAt(i).Find("fold",pKeyWord->GetAt(i).GetLength()-4) != -1)
                    bResult = FALSE;
                if( pKeyWord->GetAt(i).GetLength() > 3 &&
pKeyWord->GetAt(i).Find("bp",pKeyWord->GetAt(i).GetLength()-2) != -1)
                    bResult = FALSE;
                if( pKeyWord->GetAt(i).GetLength() > 3 &&
pKeyWord->GetAt(i).Find("nM",pKeyWord->GetAt(i).GetLength()-2) != -1)
                    bResult = FALSE;
                if( pKeyWord->GetAt(i).GetLength() > 7 &&
pKeyWord->GetAt(i).Find("microM",pKeyWord->GetAt(i).GetLength()-6) != -1)
                    bResult = FALSE;
                if( pKeyWord->GetAt(i).GetLength() > 3 &&
pKeyWord->GetAt(i).Find("UV",pKeyWord->GetAt(i).GetLength()-2) != -1)
                    bResult = FALSE;
                if( dwCount > (DWORD)pKeyWord->GetAt(i).GetLength()/2 || bResult == FALSE ||
dwNumber > (DWORD)pKeyWord->GetAt(i).GetLength()/2)
                {
                    pKeyWord->RemoveAt(i);
                    i--;
                }
            }
    }

//Transfer words into their origins using porter stemmer.
symbol sBuffer[1024] = {0};
TCHAR  sBufferFinish[1024] = {0};
SN_env * pSNenv = NULL;

    pSNenv = porter_create_env();

    for ( i=0; i<(DWORD)pKeyWord->GetCount(); i++ )
    {
        ::FillMemory( sBuffer, 1024, 0 );
        CopyMemory( sBuffer, (LPCTSTR)(pKeyWord->GetAt(i)), pKeyWord->GetAt(i).GetLength()
> 1023 ? 1023 : pKeyWord->GetAt(i).GetLength() );
        TRACE( "orig = %s ", pKeyWord->GetAt(i) );
        SN_set_current( pSNenv, pKeyWord->GetAt(i).GetLength() > 1023 ? 1023 :
pKeyWord->GetAt(i).GetLength(), sBuffer );
        porter_stem( pSNenv );
        ::FillMemory( sBufferFinish, 1024, 0 );
        CopyMemory( sBufferFinish, pSNenv->p, pSNenv->l > 1023 ? 1023 : pSNenv->l );
        TRACE( "%s\n", sBufferFinish );
        (*pKeyWord)[i] = sBufferFinish;
```

```
    }
    porter_close_env( pSNenv );

  //Remove words that can be found in WordNet.
    pNoun = new CNoun;
    ASSERT_VALID( pNoun );
    pNoun->Open();
    for( i=0; i< (DWORD)pKeyWord->GetCount(); i++)
    {
        if( !pKeyWord->IsEmpty() && pKeyWord->GetAt(i).Find("'") == -1)
        {
            szTemp = (*pKeyWord)[i];
            szTemp.Trim( );
            szTemp.Trim(",%&.@#*!~?");
            pNoun->m_strFilter.Format("%s%s%s","noun='",szTemp.MakeLower(),"'");
            if(!pNoun->Requery())
                break;
            if( !pNoun->IsEOF() )
            {
                pKeyWord->RemoveAt(i);
                i--;
            }
        }
    }
    pNoun->Close();
    delete pNoun;

  //Add the protein found into variables m_sCProtein and m_sProteinName.
    for( i=0; i<(DWORD)pKeyWord->GetCount(); i++ )
    {
        for ( bResult = TRUE, ListProteinIterator = m_lpProtein.begin();
ListProteinIterator != m_lpProtein.end(); ++ListProteinIterator )
        {
            if( (*ListProteinIterator)->szProteinName == pKeyWord->GetAt(i) )
            {
                bResult = FALSE;
                pProtein = (LPPROTEIN) (*ListProteinIterator);
                break;
            }
        }
        if( bResult )
        {
            pProtein = new PROTEIN;
            ASSERT( pProtein != NULL );
            pProtein->szProteinName = pKeyWord->GetAt(i);
            pProtein->dwCount = 0;
        }
        for(bResult = TRUE, j=0; j<(DWORD)m_sProteinName.GetCount(); j++ )
        {
            if( pKeyWord->GetAt(i) == m_sProteinName[j] )
            {
                bResult = FALSE;
                pProtein->dwCount++;
                break;
            }
        }
        if( bResult )
        {
            m_sProteinName.Add( pKeyWord->GetAt(i) );
```

```
                pProteinName->Add( pKeyWord->GetAt(i) );
                m_sCProtein.Add( pKeyWord->GetAt(i) );
                m_lpProtein.push_back( pProtein );
            }
        }
```

3. retrieve potential protein names using regular expression

```
if( pszText != NULL && pszText->IsEmpty() == FALSE )
    {
        ProcessFullStop( pszText);
        file.Open("c:\\text.txt",CFile::modeCreate | CFile::modeReadWrite);
        file.WriteString( (LPCTSTR) (*pszText) );
        file.Close();
        dwResult = system( "D:\\ProteinInteraction\\POStagger\\mytagger.exe
        D:\\ProteinInteraction\\POStagger\\LEXICON.BROWN.AND.WSJ c:\\TEXT.TXT
        D:\\ProteinInteraction\\POStagger\\BIGRAMS
        D:\\ProteinInteraction\\POStagger\\LEXICALRULEFILE.WSJ
        D:\\ProteinInteraction\\POStagger\\CONTEXTUALRULEFILE.WSJ > C:\\res.txt ");
        file.Open("C:\\res.txt",CFile::modeReadWrite);
        pszText->Empty();
        szTemp = (LPTSTR) ::GlobalAlloc( GPTR,(SIZE_T) file.GetLength()+1 );
        ASSERT( szTemp!=NULL );
        file.Read( szTemp ,(UINT)file.GetLength());
        file.Close();
        *pszText = szTemp;
        ::GlobalFree( szTemp );
        szTemp = NULL;
        MakePause( pszText, &saSentence);
    }
```

```
 //Find sentences with interaction keywords and extract those with NN tag using regular
expression.
    RE.SetExpression("interact.*|bind.*|associat.*|complex.*|two-hybrid.*|protein.*|domain
.*|require.*|kinase.*|function.*|essential.*|component.*|suggest.*|demonstrate.*");
    for( i=0; i<(DWORD)saSentence.GetCount(); i++)
    {
        RE.SetStringToMatch(saSentence.GetAt(i));
        if( RE.Grep() == 0)
        {
            saSentence.RemoveAt(i);
            i--;
        }
        else
        {
            TRACE("The sentence =%s\n",saSentence[i]);
            for( dwFind = 0;dwFind< (DWORD)saSentence[i].GetLength(); )
            {
                bResult = FALSE;
                dwFind = saSentence[i].Find("/NN",dwFind);
                if( dwFind != -1 )
                {
                    for( j=dwFind; j>0; j--)
                    {
                        if( saSentence[i].GetAt(j) == '>')
                        {
                            k = j;
                            while( k > 0)
                            {
```

```
                                        if( saSentence[i].GetAt(k) != '<')
                                            k--;
                                        else
                                        {
                                            bResult = TRUE;
                                            break;
                                        }
                                    }
                                }
                                if( saSentence[i].GetAt(j) == ' ' && j!=dwFind )
                                {
                                    if( bResult )
                                    {
                                        szMid = saSentence[i].Mid(j,dwFind-k).Trim();
                                        szMid.Trim(_T("?!'[]*,.();<>^|\""));
                                        TRACE( "rexrexrex %s\n", szMid );
                                        saTempWord.Add( szMid );
                                        break;
                                    }
                                    else
                                    {
                                        szMid = saSentence[i].Mid(j,dwFind-j).Trim();
                                        szMid.Trim(_T("?!'[]*,.();<>^|\""));
                                        TRACE( "rexrexrex %s\n", szMid );
                                        saTempWord.Add( szMid );
                                        break;
                                    }
                                }
                            }
                            dwFind = dwFind + 5;
                        }
                    }
                }
            }

//Find word frequency related to the word “interact.”
            for( i=0; i<(DWORD)saSentence.GetCount(); i++ )
            {
                for( dwResult=0; dwResult<(DWORD)saSentence[i].GetLength(); )
                {
                    dwResult = saSentence[i].Find("interact",dwResult);
                    if( dwResult != -1)
                    {
                        dwInteract++;
                        dwResult = dwResult + 8;
                    }
                }
            }

//Find word frequency related to the word “associate.”
            for( i=0; i<(DWORD)saSentence.GetCount(); i++ )
            {
                for( dwResult=0; dwResult<(DWORD)saSentence[i].GetLength(); )
                {
                    dwResult = saSentence[i].Find("associat",dwResult);
                    if( dwResult != -1)
                    {
                        dwAssociate++;
                        dwResult += 8;
```

```
            }
        }
    }

//Find word frequency related to the word “bind.”
    for( i=0; i<(DWORD)saSentence.GetCount(); i++ )
    {
        for( dwResult=0; dwResult<(DWORD)saSentence[i].GetLength(); )
        {
            dwResult = saSentence[i].Find("bind",dwResult);
            if( dwResult != -1)
            {
                dwBind++;
                dwResult += 4;
            }
        }
    }

//Find word frequency related to the word “complex.”
    for( i=0; i<(DWORD)saSentence.GetCount(); i++ )
    {
        for( dwResult=0; dwResult<(DWORD)saSentence[i].GetLength(); )
        {
            dwResult = saSentence[i].Find("complex",dwResult);
            if( dwResult != -1)
            {
                dwComplex++;
                dwResult += 7;
            }
        }
    }
    m_dwWordCount.Add( dwAssociate );
    m_dwWordCount.Add( dwBind );
    m_dwWordCount.Add( dwComplex );
    m_dwWordCount.Add( dwInteract );

//Remove words from stop word dictionary.
    pSet = new CWordsnot;
    ASSERT_VALID( pSet );
    pSet->Open();
    for( i=0; i<(DWORD)saTempWord.GetCount(); i++ )
    {
        if( !saTempWord.IsEmpty() && saTempWord[i].Find("'") == -1)
        {
            szMakelower = saTempWord[i];
            pSet->m_strFilter.Format("%s%s%s","wordsnot='",szMakelower.MakeLower(),"'");
            if( !pSet->Requery() )
                break;
            if( !pSet->IsEOF() )
            {
                saTempWord.RemoveAt(i);
                i--;
            }
        }
    }
    pSet->Close();
    delete pSet;

//Remove words that can be found in WordNet.
```

```
      pNoun = new CNoun;
      ASSERT_VALID( pNoun );
      pNoun->Open();
      for( i=0; i<(DWORD)saTempWord.GetCount(); i++)
      {
            if( !saTempWord.IsEmpty() && saTempWord[i].Find("'") == -1)
            {
                  szMakelower = saTempWord[i];
                  szMakelower.Trim();
                  szMakelower.Trim("[]\"<>,%&.@#*!~?");
                  pNoun->m_strFilter.Format("%s%s%s","noun='",szMakelower.MakeLower(),"'");
                  if( !pNoun->Requery())
                        break;
                  if( !pNoun->IsEOF() )
                  {
                        saTempWord.RemoveAt(i);
                        i--;
                  }
            }
      }
      pNoun->Close();
      delete pNoun;

  //Remove plurality.
      pPlurality = new CPlurality;
      ASSERT_VALID( pPlurality );

      pPlurality->Open();
      for( i=0; i<(DWORD)saTempWord.GetCount(); i++)
      {
            if( !saTempWord.IsEmpty() && saTempWord[i].Find("'") == -1)
            {
                  szMakelower = saTempWord[i];
      pPlurality->m_strFilter.Format("%s%s%s","word='",szMakelower.MakeLower(),"'");
                  if( !pPlurality->Requery())
                        break;
                  if( !pPlurality->IsEOF() )
                  {
                        saTempWord.RemoveAt(i);
                        i--;
                  }
            }
      }
      pPlurality->Close();
      delete pPlurality;

  //Remove suffix using porter.stem2.
  symbol sBuffer[1024] = {0};
  TCHAR  sBufferFinish[1024] = {0};
  SN_env * pSNenv = NULL;
      pSNenv = porter_create_env();
      for ( i=0; i<(DWORD)saTempWord.GetCount(); i++ )
      {
            ::FillMemory( sBuffer, 1024, 0 );
            CopyMemory( sBuffer, (LPCTSTR) (saTempWord.GetAt(i)),
  saTempWord.GetAt(i).GetLength() > 1023 ? 1023 : saTempWord.GetAt(i).GetLength() );
            TRACE( "orig = %s ", saTempWord.GetAt(i) );
            SN_set_current( pSNenv, saTempWord.GetAt(i).GetLength() > 1023 ? 1023 :
  saTempWord.GetAt(i).GetLength(), sBuffer );
```

```
            porter_stem( pSNenv );
            ::FillMemory( sBufferFinish, 1024, 0 );
            CopyMemory( sBufferFinish, pSNenv->p, pSNenv->l > 1023 ? 1023 : pSNenv->l );
            TRACE( "%s\n", sBufferFinish );
            saTempWord[i] = sBufferFinish;
      }

      porter_close_env( pSNenv );

   //Remove words that can be found in WordNet again.
      pNoun = new CNoun;
      ASSERT_VALID( pNoun );
      pNoun->Open();
      for( i=0; i<(DWORD)saTempWord.GetCount(); i++)
      {
            if( !saTempWord.IsEmpty() && saTempWord[i].Find("'") == -1)
            {
                  szMakelower = saTempWord[i];
                  pNoun->m_strFilter.Format("%s%s%s","noun='",szMakelower.MakeLower(),"'");
                  if( !pNoun->Requery())
                        break;
                  if( !pNoun->IsEOF() )
                  {
                        saTempWord.RemoveAt(i);
                        i--;
                  }
            }
      }
      pNoun->Close();
      delete pNoun;

   //Add protein name found using regular expression into variables sREProtein and m_sProteinName.
      for( i=0; i<(DWORD)saTempWord.GetCount(); i++)
      {
            for( bResult = TRUE, ListProteinIterator = m_lpProtein.begin();
ListProteinIterator != m_lpProtein.end(); ++ListProteinIterator)
            {
                  if( (*ListProteinIterator)->szProteinName == saTempWord.GetAt(i) )
                  {
                        bResult = FALSE;
                        pProtein = (LPPROTEIN) (*ListProteinIterator);
                        break;
                  }
            }
            if( bResult )
            {
                  pProtein = new PROTEIN;
                  ASSERT( pProtein != NULL );
                  pProtein->szProteinName = saTempWord.GetAt(i);
                  pProtein->dwCount = 0;
            }
            for( bResult = TRUE,j=0; j<(DWORD)m_sProteinName.GetCount(); j++)
            {
                  if( saTempWord.GetAt(i) == m_sProteinName.GetAt(j) )
                  {
                        bResult = FALSE;
                        pProtein->dwCount++;
                        break;
                  }
```

```
            }
            if( bResult )
            {
                  m_sREProtein.Add( saTempWord[i] );
                  m_sProteinName.Add( saTempWord[i] );
                  m_lpProtein.push_back( pProtein );
            }
      }
```

4. Retrieve similar documents from the Internet

```
  //Organize query string and submit the URLs.
qsort(FindKeyWord, 20, sizeof(KEYWORD), CPreprocess::compare );
      for( i=0, ListProteinIterator = m_KI.m_lpProtein.begin(); i<3 &&
ListProteinIterator!=m_KI.m_lpProtein.end(); i++,++ListProteinIterator )
      {
            saKeyword.Add( (*ListProteinIterator)->szProteinName );
      }
      szURL = "http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=PubMed&orig_db=PubMed&term=" +
(*FindKeyWord[0].sKeyword);
      szURL = szURL + "AND";
      szURL = szURL + FindKeyWord[1].sKeyword + "AND";
      szURL = szURL + FindKeyWord[2].sKeyword + "OR";
      szURL = szURL + saKeyword[0] + "OR";
      szURL = szURL + saKeyword[1] + "OR";
      szURL = szURL + saKeyword[2] + "&dopt=DocSum&dispmax=";
      szTemp.Format("%d", m_dwSetCount);
      szURL = szURL + szTemp;
      ::AfxMessageBox( szURL,MB_OK);
      m_szTempURL = szURL;

  //Retrieve the web content according to the URL provided by the user and save it in m_pszBuffer.
      m_szURL = url;
      if ( m_pszBuffer )
      {
            delete m_pszBuffer;
            m_pszBuffer = NULL;
      }
      m_pszBuffer = new CString;
      ASSERT( m_pszBuffer != NULL );
      pCurl = curl_easy_init();
      curl_easy_setopt(pCurl,CURLOPT_URL,m_szURL);
      curl_easy_setopt(pCurl,CURLOPT_WRITEFUNCTION,CallbackWrite);
      curl_easy_setopt(pCurl,CURLOPT_WRITEDATA, (LPVOID) m_pszBuffer);
      res = curl_easy_perform( pCurl );
      curl_easy_cleanup( pCurl );

  // Retrieve corresponding URLs for the similar documents.
      for( dwFind=0; dwFind<(DWORD)pszText->GetLength(); )
      {
            dwStart = pszText->Find( "PMID:", dwFind);
            if( dwStart == -1)
            {
                  dwEnd = pszText->Find( "[PubMed - in process]", dwStart);
                  szTemp =
"http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&list_uids=" +
pszText->Mid(dwStart+5,dwEnd-(dwStart+5)) + "&dopt=Abstract";
                  TRACE("URL:%s \n",szTemp);
                  m_sListURL.AddTail( szTemp );
```

```
                dwFind = dwEnd + 22;
            }
        }
        if( m_sListURL.IsEmpty() == FALSE)
        {
            if( pSet )
            {
                delete pSet;
                pSet=NULL;
            }
            pSet = new CSimurl;
            ASSERT_VALID( pSet );
            pSet->Open();
            for( pos = m_sListURL.GetHeadPosition(); pos != NULL; )
            {
                pSet->AddNew();
                pSet->m_profile = (*pszProfile);
                pSet->m_favname = (*pszFavname);
                pSet->m_urladr = m_sListURL.GetAt(pos);
                pSet->m_ntchk = '2';
                pSet->Update();
                m_sListURL.GetNext(pos);
            }
            pSet->Close();
        }
        delete pSet;
        pSet = NULL;

    //Calculate the similarity.
        for( i=0; i<20; i++)
        {
            dNormalize = dNormalize + FindKeyWord[i].dCount;
        }

    //Normalization process.
        for( i=0; i<20; i++)
        {
            FindKeyWord[i].dCount = FindKeyWord[i].dCount / dNormalize;
        }

        for( i=0; i<20; i++)
        {
            dSumFind = dSumFind + FindKeyWord[i].dCount;
            dSumOrig = dSumOrig + OrigKeyWord[i].dCount;
        }
        for( i=0; i<20; i++)
        {
            if( OrigKeyWord[i].dCount < FindKeyWord[i].dCount )
                dSum = dSum + OrigKeyWord[i].dCount;
            else
                dSum = dSum + FindKeyWord[i].dCount;
        }
        dMD = pow((dSum * dSum )/((dSumFind*dSumOrig)+0.0001),0.5);
        m_dSimilarity = dMD;
```