REGEN: Retrieval and Extraction of Genomics Data

Rocio Guillén, Tasnim Ferdous
Computer Science Department, California State University San Marcos
email:{rguillen,ferdo001}@csusm.edu

Abstract

In this paper we present REGEN (Retrieval and Extraction of GENomics Data) a natural language processing system that retrieves and extracts information from Genomics data. These two tasks are independent of each other in the sense that the retrieved documents are not input to the extraction task. The retrieval task is based on a combination of exact-match and partial-match searching. The extraction task uses syntactic and semantic cues as patterns to generate candidate GENERIFs. We are currently generating just one candidate.

1. Introduction

One of the goals of the TExt Retrieval Conference (TREC) is to encourage research in information retrieval from large test collections. The nature of the tasks, data and evaluation procedures is experimental. TREC activity is organized into "tracks" of common interest, such as question-answering, multi-lingual IR, Web searching, and interactive retrieval. Because a great deal of genomics information resources are available one of the new tracks for 2003 was the TREC Genomics track focused on the retrieval of texts to aid users to acquire new knowledge in a sub-area of biology linked with genomics information. We have implemented a search engine for information retrieval and an information extraction procedure to participate in TREC 2003 in the Genomics Track. The Genomics Track consisted of two tasks namely primary and secondary. The focus of the primary task was on ad-hoc information retrieval and the focus of the secondary task was on information extraction. The Information Retrieval (IR) approach used for the search engine was the Extended Boolean model with partial matching and term weighting for the ranking (1,2,3,4,5). For the extraction we analyzed the MEDLINE abstracts provided as training set looking for syntactic and semantic cues we could use as patterns for generating GENERIFs candidates.

2. Architecture

We have built a system with two independent modules, namely retrieval and extraction. The overall architecture of the system is presented in Figure 1.

2.1 Retrieval

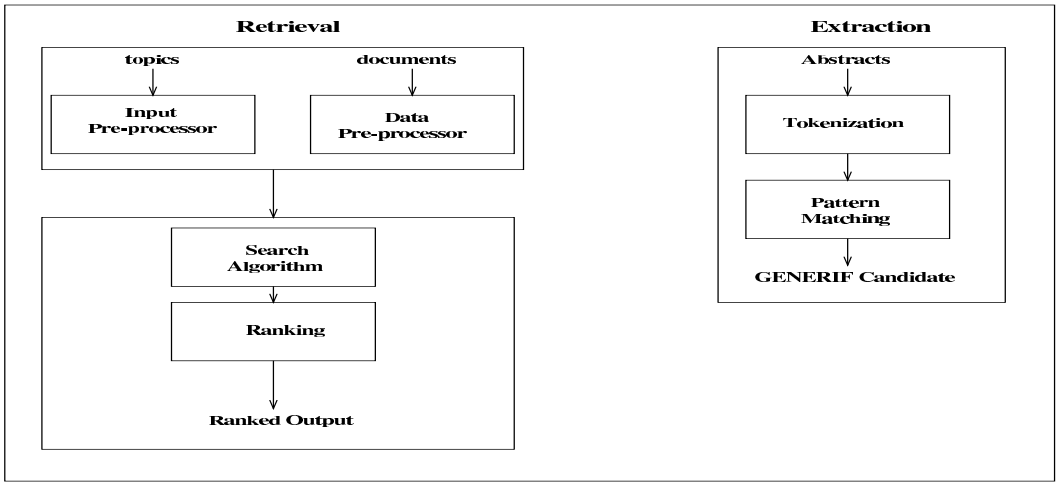

Figure 1 REGEN

Pre-processing of both the documents and queries was necessary to normalize, expand and modify data. We implemented a pre-processor for the document collection (Data Pre-processor) and one pre-processor for the topics or queries (Input Pre-processor). The Data pre-processor takes as input the document collection in XML format. and outputs two files called trec_XML.con and trec_XML.idx. The file called trec_XML.con stores the pre-processed documents and the file called trec_XML.idx keeps pointers to different positions of the trec_XML.con file. The purpose is to process information included in one of the following tags: MedlineID, PMID, ArticleTitle, AbstractText, DescriptorName, and NameOfSubstance. The Input pre-processor reads the topics from the topic file and then reorganizes them by eliminating redundant fields, LocusLink ID and name type, from the file. The basic steps carried out are the following:

1. Read a topic, which is a string of characters from the input file.
2. Write topic number in a new file.
3. Skip second substring and read third substring.
4. Replace third substring read for Organism name with appropriate names.
5. Separate Gene name and products using "—" as marker.

Mapping the name of species into corresponding MeSH equivalent names modifies the third substring of the topic file, which represents the Organism name. Thus,

* the string Human replaces Homo Sapiens.
* the string Mice replaces Mus musculus.
* the string Rat replaces Rattus norvegicus.

* the string Drosophila replaces Drosophila melanogaster.

The topics thus processed are further expanded applying some specific rules, presented next, to generate queries used by the search engine. The rules used were obtained after carrying out experiments to find patterns in the topics, and are consistent for all the topics. These rules are applied on gene names with OFFICIAL_GENE_NAME type only. The reason behind this restriction is that we observed that mainly the gene names of this type are rather complex and need to be split for further processing.

2.1.1 Rules

We came up with ten rules for normalizing and expanding queries. The rules reflect the patterns we found by carrying out different experiments and analyzing the results generated.

Rule 1. A gene name with a pattern like <cyclin-dependent kinase inhibitor 1A (p21, Cip1)> is split into two strings. The first string, s1, corresponds to all the characters appearing before "(", i.e., "cyclin- dependent kinase inhibitor 1A". The second string corresponds to "(" , ")", i.e., "(p21, Cip1)", which includes two substrings separated by a comma. The second string is split into s2 that corresponds to "p21" and s3 that corresponds to "Cip1", and the gene name is represented as <s1(s2,s3)>. Where, each string will be considered as the new query ""cyclin-dependent kinase inhibitor 1A p21 Cip1".

Rule 2. A gene name with a pattern like <glycine receptor, alpha 1 (startle disease/hyperekplexia, stiff man syndrome)> can be represented as four strings <s1, s2 (s3, s4)>. s1 corresponds to "glycine receptor", s2 corresponds to "alpha 1", s3 corresponds to "startle disease/hyperekplexia", and s4 corresponds to "stiff man syndrome". Such pattern can be further expanded as s1, s1 + s2, s1 + s3, and s1 + s4.

Rule 3. A gene name like <luteinizing hormone/choriogonadotropin receptor> can be represented as the pattern <s1 s2/s3 s4>. s1 corresponds to "luteinizing", s2 corresponds to "hormone", s3 corresponds to "choriogonadotropin" and s4 corresponds to "receptor". The pattern is further expanded as s1 + s2 + s4, and s1 + s3 + s4. Therefore, the newly generated queries are "luteinizing hormone receptor" and "luteinizing choriogonadotropin receptor".

Rule 4. A gene name with a pattern like <phospholipase C, gamma 1> can be represented as <s1, s2> where s1 corresponds to "phospholipase C" and s2 corresponds to "gamma 1". The pattern is further expanded to s1 + s2 and reduced to s1. Therefore,

the new generated queries are "phospholipase C gamma 1" and "phospholipase C".

Rule 5. Any single number (N) or literal (L) in a gene name or a newly generated query is concatenated with the previous word. If a gene name or a newly generated query is in the pattern <Calcineurin B> that is <s1 L> where s1 corresponds to "Calcineurin" and L corresponds to "B" then the new pattern is s1L. That is, the newly generated query is "CalcineurinB". If a gene name or a newly generated query is in the pattern < alpha 1> that is <s1 N> where s1 is "alpha" and N is "1" then the new pattern becomes s1N. This means that the newly generated query is "alpha1".

Rule 6. When the pattern is a list of names separated by comma and within parentheses then each of the elements in the list becomes a query itself. For instance, a gene name with the pattern <Tachykinin (Substance P, Neurokinin A, Neuropeptide K, Neuropeptide gamma)> can be represented as <s1 (s2, s3, s4, s5, s6..... sn)>. s1 corresponds to "Tachykinin", s2 to "Substance P", s3 to "Neurokinin A", s4 to "Neuropeptide K", and s5 to "Neuropeptide gamma". Therefore the newly generated queries are s1, s2. s3. s4, and s5.

Rule 7. When the pattern is a list of names separated by comma like <major histocompatibility complex, class II, DQ beta 1> then it can be represented as <s1, s2, s3, ..., sn>. This will generate new queries s1 that corresponds to "major histocompatibility complex", s1 + s2 where s2 corresponds to "class II", s1 + s3 where s3 corresponds to "DQ beta 1". In general, the new queries are s1 + sj, where j = 2, ", n. Therefore, for the gene name shown above the newly generated queries will be "major histocompatibility complex", "major histocompatibility complex class II" and "major histocompatibility complex DQ beta 1".

Rule 8. A gene name like <Janus kinase 2 (a protein tyrosine kinase)> can be represented as the pattern <s1 (s2)>. s1 corresponds to "Janus kinase 2", and s2 corresponds to "a protein tyrosine kinase". Therefore, the newly generated queries are "Janus kinase 2" and "a protein tyrosine kinase".

Rule 9. A gene name like <metallothionein 3 (growth inhibitory factor (neurotrophic))> can be represented as the pattern <s1 (s2 (s3))>. s1 corresponds to "metallothionein 3", s2 corresponds to "growth inhibitory factor" and s3 corresponds to "neurotrophic". Therefore, the newly generated queries are "metallothieonein 3", "growth inhibitory factor", and "neurotrophic".

Rule 10. If there is a literal (L) and a number (N) or a number (N) and a literal (L) after any string then they are concatenated together. For instance, if the gene name is <Alpha 1A> it can be represented as the pattern <s1 NL> where s1 corresponds

to "Alpha", N corresponds to "1" and L corresponds to "A" then the new pattern is <s1NL>. That means that the newly generated query is "Alpha1A". Similarly, the gene name <Alpha A1> can be represented as the pattern <s1 LN>. Therefore, the newly generated query is "AlphaA1".

2.1.2 Applying Rules

The procedure for applying these rules is summarized as follows. Start by reading the first gene name that is the OFFICIAL_GENE_NAME from the processed topic file. The gene name is a term composed of different subterms. For term T, let t1, t2, t3 be different subterms. Thus term T is decomposed using the following rules:

a. If the pattern is t1(t2,t3,t4,..) then replace it with the pattern t1/t2/t3/t4.
b. If the pattern is t,t1,t2,t3 then replace it with the pattern t, t1/t2/t3 .

Each pattern [t1/t2/t3] generated is a term (denoted by E). Thus,
a. t1/t2/t3 is replaced with E1.
b. t, t1/t2/t3 which is replaced with E1, E2.

Let the term T be decomposed as E1 [,] E2 E3. The expanded terms generated are as follows.
a. T1, where T1 $\in$ E1 and there is a comma after E1.
b. T1 T2 T3 ..., where T1 $\in$ E1, T2 $\in$ E2 and T3 $\in$ E3.

Lastly, concatenate all numerals, single letters, any sequence of one numeral and one letter.

2.1.3 Search

We used the extended Boolean model for our retrieval system with partial matching and term weighting. The original search procedure developed works as the combination of three tests defined as follows.

Test1: Check whether there is a match between the name of the organism in the topic of interest and the contents of <DescriptorName> tag in the document.
Test2: Check whether there is a match between the gene name or any of its symbol or alias symbol in the topic and the contents of the <ArticleTitle> tag and/or <AbstractText> tag in the document.
Test3: Check whether there is a match between the gene name or any of its symbol or alias symbol in the topic and the contents of the <NameOfSubstance> tag.

If (Test1 AND Test2 AND Test3) is true for a document given a topic then that document is considered relevant to that specific topic. We used partial matching when the search did not retrieve at least three relevant documents for a given topic. In this case a document is considered relevant if (Test1 AND (Test2 OR Test 3)) is true.

The algorithm for the different phases of the search is given below.

Until all queries are processed
1. Read a query (topic) Q.
2. For each document D in the collection.
Do
2.1 Search for the organism name of Q in Descriptor_Name tag of D.
If there is a match, for each term T in Q except the first one,
2.2 Search for T in the ArticleTitle and AbstractText of D.
If there is a match, for each term T in Q except the first one,
2.3 Search for T in the Name of Substance.
If there is an match then D is relevant to Q.

Other factors considered in the retrieval module are the following.

1. Multi-word spanning: This feature allows to search for a string in the query as two strings in the document and vice versa. For instance, *histocompatibilty* can be searched as *histo compatibility* and vice versa.

2. 80% similarity rule: Using this feature two strings can be said to be similar if they match 80%. The 80% is calculated taking the length of the strings. For instance, *METALLOTHIONEINIII* and *METALLOTHIONEIN3* are similar using this rule.

3. Stemming: Only rules for plurals were implemented.

4. Expansion of abbreviations: If a gene name has its abbreviation in the topic then it is expanded using the different combinations of the gene name and the abbreviation of the gene name. For instance, the gene name <*luteinizing hormone receptor*> and its abbreviation <*LHR*> was used to expand the queries as <*Luteinizing HR*>, <*Leuteinizing Hormone R*>, <*LH Receptor*> and <*L Hormone Receptor*>.

2.1.4 Scoring

After a document is determined to be relevant, the scoring is done based on frequency and on weights. We first score a document based on frequency. We assign one

point each time any of the gene name, its symbol, alias symbol, product or alias product appears in the document. Next, we score a document by assigning some constant weights based on its performance. The different constant weights used are Title_weight, Partial_weight, Exact_weight and Complete_weight. Title_weight is given to a document, which has either the gene name or its symbol or alias symbol, or the product or alias product in the title, i.e., within the ArticleTitle tag. Partial_weight is given to a document where some sub-strings of the gene name, its symbol, alias symbol, product or alias product is fully matched and the remaining is partially matched. Exact_weight is used for the documents that contain the gene name, its symbol, alias symbol, product or alias product exactly the way they are in the NameOfSubstance. Complete_weight was assigned to the documents for which the outcome of the three tests mentioned earlier was successful for determining relevancy. This weight was assigned to maintain some distinction between the documents retrieved by the original retrieval and the documents that were retrieved using partial- match. Except for the Title_weight no other constant weights are added to the documents retrieved with just partial-match. After scoring all the documents, the top 100 documents are selected as the relevant documents for the result.

2.2 Experiments

We carried out six experiments in which we modified the search procedure to improve recall and precision. The first five experiments focused on improving recall and the last experiment, which itself consisted of several tests (experiments), focused on improving precision.

2.2.1 Experiments for recall

The first experiment used only the three tests of relevancy. For the second experiment we added multi-word spanning and the 80% similarity rule. For the third experiment we added stemming. Next we implemented expansion of abbreviations. Lastly, we implemented partial matching. We observed a monotonous increment of recall in every new experiment which is shown in Figure 2.

2.2.2 Experiments for precision

To improve precision it was understood that we needed to rank the relevant documents higher than the non-relevant ones. We assigned different types of weights as discussed in section 3. We carried out experiments with different sets of values to determine the set that generated the best average precision for the training data. A comparison of the different sets of values is presented in Table 1.

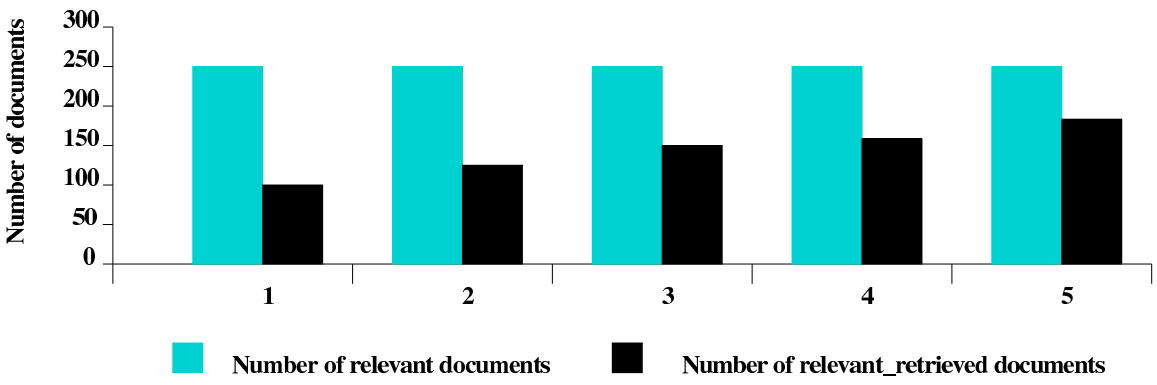

Figure 2. Experiments for improving recall

Value Set	Title Weight	Partial Weight	Exact Weight	Complete Weight	Average Precision
1	3	7	7	3	0.47
2	7	5	10	7	0.452
3	3	5	5	3	0.456
4	3	4	5	2	0.394
5	3	5	7	5	0.393

Table 1: Comparing average precision for different scores.

3. Extraction Module

The main purpose of participating in the secondary task was to carry out preliminary studies and experiments with the resources we have available at our site. Currently, the extraction module has two components, namely tokenization and pattern-matching.

The first step before extracting and generating GENERIF candidates was to download the abstracts from MEDLINE. Next we created XML-like tags for each abstract to make the tokenization more efficient. The tags we added are the following: 1) a number tag <N< and </N> that is a sequential number from 1 to 139; 2) a title tag <TI> and </TI>; 3) an author tag <AU> and </AU>; 4) an address tag <AD> and </AD>; 4) a text tag <TX> and </TX> for the abstract itself; and 5) an identifier tag <ID> and </ID> for the PMID. An example is shown below.

```
<N> 1 </N><TI>Regulation of Fas-associated death domain interactions
by the death effector domain identified by a modified reverse ... </TI>
<AU> Thomas LR, Stillman DJ, Thorburn A.</AU><AD>Department of Cancer
Biology, ...</AD><TX>The adapter protein FADD consists of two protein
... </TX> <ID> PMID: 12107169 [PubMed - indexed for MEDLINE] </ID>
```

Next, we analyzed the abstracts in terms of the GENERIFs given, looking for syntactic and semantic cues that could be used as patterns to extract information for generating GENERIF candidates. The GENERIF given was compared with the title of the abstract to find similarities and differences. In many instances, there was an exact match with the title of the abstract. Thus, the title became a default GENERIF candidate. If the match was not an exact match differences were checked. We observed that some of the differences occurred with verbs. For instance, a verb in the GENERIF given was nominalized in the title or abstract text and viceversa; a verb appeared in passive form in the GENERIF given whereas it was in active form in the title or abstract text and viceversa.

Then we examined the abstract text and we found verbs, nouns and adverbs potentially useful for our task. Among the verbs are "demonstrate", "activate", "promote", "induce", "show", "suggest", "provide", "indicate", "identify", "regulate", "conclude", "reveal", and "mediate". The nouns that we determined were more relevant for extraction purposes were "data", "results" and "study" when they appeared with verbs such as "demonstrate", "show", "indicate" and "suggest". Adverbs such as "therefore", "together", and "thus" appeared to be an important link between the GENERIF given and the abstract text. In other words, these adverbs are relevant to those who generated the GENERIFs given. The result of this analysis was a set of patterns that we used for generating GENERIFs. We used a small subset of patterns in the actual implementation. The patterns in this subset are "therefore", "indicate" and "together". The latter when it occurs at the beginning of a sentence. The beginning of a sentence is marked by a "." followed by a string whose first character is an uppercase letter. A sentence is the set of strings between two ".".

3.1 Implementation

The extraction module has two main functions a tokenizer and a pattern search. The tokenizer scans the abstract to identify boundaries of words and tags. We are interested in identifying tag boundaries first. Once a tag is identified, it is further processed to determine whether it is a title or the abstract text. The other tags are ignored except for the tag that identifies the sequential number which is used in the output. A default GENERIF is generated directly from the title. The tokenizer then scans the abstract text to identify boundaries of words.

The pattern search compares each of the words with the subset of patterns. If a match is found a GENERIF is generated from the rest of the sentence whenever the pattern is an adverb. The GENERIF will include the pattern whenever the pattern is a verb. If there is no match we use the default GENERIF as the GENERIF candidate. We did not use probabilities nor weights to rank candidates because the subset of patterns is small. In most cases the default GENERIF was kept.

3.2 Evaluation

Results of the evaluation show that our preliminary approach did well or poorly. In average the figures for the Classic Dice, Modified unigram Dice, Modified bigram Dice and Modified bigram Dice phrases are the same or slightly above the median. We conclude that the number of GENERIFs generated by the patterns was relatively small which is likely due to the fact that the number of patterns used was small. Therefore, we need to carry more experiments with a larger set of patterns, among other tasks, to determine whether syntactic cues are useful in information extraction of genomics data.

4. Conclusion

Preliminary results and evaluation have shown that our system has the potential to become an efficient and accurate system for retrieval and extraction of genomics data. We are carrying out research to implement a different information retrieval model and exploring natural language processing approaches for the information extraction task.

5. References

1. D. Grossman and O. Frieder. Information Retrieval: Algorithms and Heuristics, Kluwer Academic Press, 1998.
2. C. J. van RIJSBERGEN. Information Retrieval, London ; Boston: Butterworths, 1979.
3. Ricardo Baeza - Yates and Berthier Ribeiro-Neto, Modern Information Retrieval , Addison-Wesley, 1999.
4. Gerard Salton and Edward A. Fox, and Harry Wu. Extended Boolean information retrieval. Communication of the ACM, 26(11): 1022-1036, November 1983.
5. William Hersh, Information Retrieval: A Health and Biomedical Perspective. Springer Verlag. 2003.