平成16年度

学 位 論 文

Searching signal transduction pathways in protein-protein interaction networks

タンパク質間相互作用ネットワークから シグナル伝達パスウェイを検出する手法の開発

東京大学大学院新領域創成科学研究科

情報生命科学専攻

学生証番号　36801

岡　弘之

指導教官：　有田　正規

Master's Thesis

Searching signal transduction pathways in protein-protein interaction networks

Student ID: 36801
Name: Hiroyuki Oka
Advisor: Masanori Arita

Department of Computational Biology
Graduate School of Frontier Sciences
The University of Tokyo

2005

Abstract

We developed a new method to search functionally similar protein interaction networks with a given query signal transduction pathway from protein interaction networks. This method consists of two main parts: 1) a backtracking search algorithm to find topologically identical subgraphs and 2) a measurement of similarity between proteins by using Gene Ontology [1]. For validation of our method, we implemented a software tool and compared its performance with that of PathBLAST [2] on the search of MAPK signal transduction cascade [3]. The result showed that our software tool detects correct answers with low false-positive rate. This tool will provide better performance when combined with computational method of predicting protein functions.

Acknowledgements

I thank Assistant Professor Masanori Arita and Professor Kiyoshi Asai for all kinds of comments and discussions. I also thank the members and staffs of Arita laboratory and Asai laboratory for kind supports and good discussions.

The members of Arita laboratory

- Masanori Arita (Assistant Professor)
- Yukiko Fujiwara (Technical Staffs)
- Yukiko Nakanishi (Technical Staffs)
- Keitaro Nonaka (Technical Staffs)
- Kazuhiro Suwa (Technical Staffs)

The members of Asai laboratory

- Kiyoshi Asai (Professor)
- Taishin Kin (Assistant)
- Naotaka Kiryu (Philosophiae Doctor)
- Kinya Okada (Philosophiae Doctor Student)
- Masaru Kabetani (Master's Student)
- Yasuo Tabei (Master's Student)
- Yuki Nishimura (Master's Student)
- Jun'ichi Narukawa (Master's Student)
- Aiko Sasaki (Office Administrator)

Contents

Chapter 1 Introduction

1.1 Biology in Post-Genomic Era

Since the first complete genome (*H. influenzae*) was sequenced in 1995, genomes of various organisms including both prokaryotes and eukaryotes have been sequenced. The availability of genome sequences for a range of prokaryotic and eukaryotic organisms has given us a comprehensive view of genes encoded in these organisms. Since the biological functions of many of these genes remain uncharacterized, a critical problem in making sense of these genomes is the assignment of functional annotations to newly discovered genes. In order to identify their functions, computational approaches such as comparative genome sequence analysis with well-studied genes have been achieved. BLAST [4] is one of representative tools in which this methodology is implemented. BLAST, however, has uncovered functions of only a subset of genes because there are not a sufficient number of characterized genes.

In recent years, various experimental methods for functional analysis of genes have been developed. They include genome-wide measurement of transcriptional levels [5], determining deletion phenotypes of single genes [6], global measurement of the subcellular localizations [7, 8], global interaction mapping of genes and transcripts (i.e. ORFs and proteins) [9] and protein-protein interaction mapping [10-21]. These methods have generated new datasets that provide additional opportunities for inference of function. Especially, protein-protein interaction mapping is of interest because it has hierarchical information of biochemical processes in living cells with contrast to linear information of genome sequences.

In this paper, protein-protein interaction data is focused on, because the purpose of our study is searching signal transduction pathways (which are sequence of protein-protein interactions).

1.2 High-Throughput Protein Interaction Assay

Traditionally, protein interactions have been studied individually by genetic, biochemical and biophysical techniques. These experimental methods, however, require huge cost in time due to their complicated protocols. In recent years, various high-throughput protein interaction assays have been developed so as to overcome the limitations. The high-throughput protein interaction assays include mainly four types of methods: (1) yeast two-hybrid systems [10, 11], (2) protein complex purification techniques using mass spectrometry [12, 13], (3) correlated messenger RNA expression profiles [14, 15] and (4) computational interaction predictions derived from gene context analysis [16-21].

(1) **Yeast two-hybrid systems** [10, 11] use two protein domains that have specific functions: a DNA-binding domain (BD) that is capable of binding to DNA, and an activation domain (AD) that is capable of activating transcription of the DNA. Although this system can identify physically interacting protein pairs in the intracellular region, when biologically assessed, a lot of false-positives appear to exist in its data set.

(2) **Protein complex purification technique** includes two different protocols: tandem affinity purification (TAP) [12] and high-throughput mass-spectrometric protein complex identification (HMS-PCI) [13]. This technique can identify a protein complex formed around a bait protein.

(3) **Correlated expression profiles of messenger RNA** are also used to identify possibly interacting protein pairs [14, 15]. This is based on the idea that proteins whose expression patterns correlate to each other are activated at the same time. By combining protein localization data, interaction data obtained by this method becomes more confident.

(4) **Computational interaction predictions** derived from gene context analysis (gene fusion [16, 17], gene neighborhood [18, 19] and gene co-occurrences or phylogenetic profiles [20, 21]) complement the three experimental methods mentioned above. They are fast *in silico* techniques. Moreover, their coverage expands as more genomes are sequenced. However, they require a framework for assigning orthology between proteins, and may fail where orthologous relationships are not clear.

The large protein interaction maps obtained by these high-throughput methods have provided new insights into the relationships among the predicted genes of sequenced genomes of model organisms. Not only a pair of proteins, but multiple proteins may bind to each other to form a functional system, which is considered to play the central role of biological systems. In the next part, we argue these systems of protein interaction networks.

1.3 Modularity of Biological Networks

Proteins and protein complexes interact with preferred partners according to time, space and conditions in order to form a biological system serving a specific collective function [22]. For example, a MAPK cascade, together with its scaffold proteins and various regulators and effectors, forms signal-amplification machinery [3]. As another example, a spindle-pole body is a community of protein complexes that form a hub for the attachment and organization of microtubules. These biological systems have been termed as "*biological modules*". The concept of this biological modularity assumes that cellular functionality can be seamlessly partitioned into a collection of modules. Each module is a discrete entity of several elementary components, and performs an identifiable task.

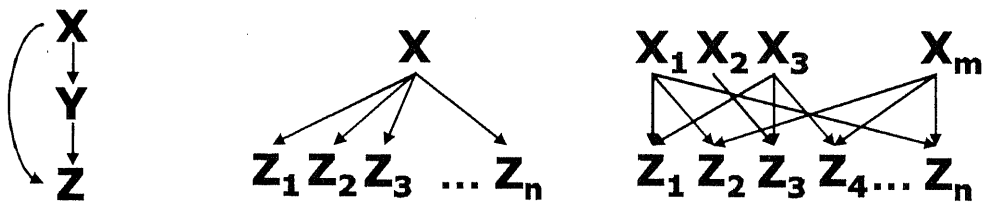
Biological Modularity may facilitate living organisms' evolutionary adaptation to the changing environment. Embedding or removing particular functions in a module allows its core function to be robust to change. Moreover, altering the connections between different modules allows for changes in the properties and functions of a cell. If the function of a protein were to directly affect all properties of the cell, it would be hard to change that protein because an improvement in one function would probably be offset by impairments in others. But if the function of a protein is restricted in one module, and the connections of that module to other modules are through individual proteins, it will be much easier to modify their comprehensive functions. This idea is supported by the analogous observation that proteins interact with many other proteins, such as histones, actin and tubulin, have changed very little during evolution.

In order to find modules *in silico*, Spirin et al. focused on multibody interactions and searched for sets of proteins having many more interactions among themselves than with the rest of the networks (clusters) [23]. They developed algorithms to which both graph theory and physical theory were applied in order to find such clusters in a yeast protein-protein interaction network. They found >50 known and previously uncharacterized protein clusters, which have high statistical significance and consistent functional annotation. They also found that most of identified clusters correspond to either of the two types of cellular modules: functional complexes and functional pathways. Functional complexes are groups of proteins that interact with each other at the same time and place, forming specific functions. Examples of identified protein complexes include several large transcriptional factors, RNA splicing and so on. Functional pathways, in contrast, consist of proteins that participate in a particular cellular process while binding each other at a different time, space and conditions. Examples of identified functional modules include the CDK/cyclin module responsible for cell-cycle progression, the yeast pheromone response pathway, MAP signaling cascades and so forth.

Han et al. applied mRNA expression profiles for the purpose of searching these two types of modules [24]. In their approach, they identified hierarchical modularity in the yeast protein-protein interaction network.

While the approaches of assuming biological modularity is advantage for identifying biological functions, in-depth exploration of biological modularity has been needed for the clarification of cell functions. Therefore, new approaches using both graph theory and control theory have come to be under requisition.

Transcriptional regulatory networks can be described as directed graphs, in which the nodes are genes or transcripts, the edges are interactions among them. It was recently found that these networks contain significantly recurring wiring patterns termed "*network motifs*" [25-27]. Network motifs are patterns that occur in the network far more often than in randomized networks of the same degree distribution. The transcription networks of the bacterium *Escherichia coli* [25, 26] and the yeast *Saccharomyces cerevisiae* [26, 27] were found to contain the same small set of highly significant motifs. Shen-Orr et al. found that the transcriptional network of the bacterium *E. coli* contains three types of network motifs [13]. They are (a) feed-forward loop, (b) single-input module, and (c) dense-overlapping regulons (shown in Fig. 1.1). Each of these network motifs seems to have a specific function in determining gene expression, such as generating temporal expression programs and governing the responses to fluctuating external signals.



(a) feed-forward loop (b) single-input module (c) dense-overlapping regulons

Fig. 1.1. Network motifs found in the *E. coli* transcriptional regulation network. X and Y imply transcription factors. Z implies a regulated operon. Each arrow implies regulation from a transcription factor to a second transcription factor or a regulated operon.

In addition to transcriptional regulatory networks, protein-protein interaction networks also have network motifs which regulate both processes of metabolic pathways and signal transduction cascades. Discovering network motifs in protein-protein interaction networks is required for understanding biological modularity in a cell. For this purpose, we aim to detect network components functionally similar to signal transduction cascades which have already been characterized and been clarified of its behavior. As a preceding study, Kelley et al. developed PathBLAST [2, 28] which detects topologically and evolutionally similar network components.

1.5 PathBLAST as a Tool for Detecting Evolutionally Similar Networks

Kelley et al. developed PathBLAST [2, 28] which detects topologically and evolutionally similar network components from protein-protein interaction networks. The core methods of their tool are (1) a heuristic algorithm which detects approximately the same networks in topology, (2) a pathway alignment method, and (3) BLAST E-value between protein sequences.

Given a pathway or a protein-protein interaction network as a query and given a huge protein-protein interaction network as a target, PathBLAST divides both the query and target networks into a set of linear pathways. Next, by using a procedure based on dynamic programming, a pair of pathways is selected which is aligned its nodes (i.e. proteins) and edges (i.e. interactions) so that alignment score *S(P)* becomes high (shown in Eq. 1.1). *S(P)* is a log probability score that decomposes over the node pairs v and edges e of a pathway pair P, where *p(v)* is the probability of true homology within the protein pair represented by v, given its pairwise protein sequence similarity expressed as a BLAST E-value, and *q(e)* is the probability that the protein-protein interactions represented by e are real, i.e. not false-positive errors (see Table 1.1 and Eq. 1.2). The background probabilities p_{random} and q_{random} are the expected values of *p(v)* and *q(e)*. Repeating these both selection and alignment procedure detect high-scored pathway pairs which are topologically and evolutionally similar network components between query and target networks.

PathBLAST finds a lot of evolutionally similar pathways between protein-protein interaction of the yeast *S. cerevisiae* and that of the bacterial pathogen *H. pylori*, but it has critical drawbacks: it cannot find cyclic pathways, and it cannot always detect functionally similar pathways because sequence similarity detects only evolutional relationship. Functional similarity of proteins does not imply evolutional similarity (though the reverse is generally considered to hold), and the information of sub-cellular localization or interaction timing is not always reflected in the BLAST E-values. To flexibly detect protein modules, therefore, such additional information should be integrated.

$$S(P) = \sum_{v \in P} \log_{10} \frac{p(v)}{p_{random}} + \sum_{e \in P} \log_{10} \frac{q(e)}{q_{random}} \tag{1.1}$$

$$q(e) = \prod_{i \in e} \Pr(i) \tag{1.2}$$

Table 1.1. The probability that the protein-protein interactions are not false-positive errors is quantified as following scores depending on the number of its studies. The reason why this probability needs to be used is that protein-protein interaction data are considered to have a lot of false-positive interactions [29].

Number of studies	*Pr(i)*
1	0.1
2	0.3
3	0.9

Here we present a new method overcoming the forementioned limitation in PathBLAST. We use a backtracking search algorithm provided by McGregor [30] which detects topologically identical subnetworks between two networks. We use Gene Ontology (GO) to compute the distance between functional annotations of proteins, which describe molecular function, cellular component and biological process of gene products [1]. In place of sequence similarity, we use "*Semantic Similarity*" proposed by Lord et al. [31], which is computed from relativities between Gene Ontology terms annotated for each protein. Lord et al. reported that Semantic Similarity of GO terms well captures molecular function, cellular components and biological processes, and it still correlates with BLAST bit-scores of protein sequences with respect to molecular function. By combining backtracking algorithm and Semantic Similarity measurement, it is possible to find protein networks that are functionally similar with a given pathway.

This paper is organized as follows. In Section 2, we introduce both a simple backtracking search algorithm to detect topologically identical subnetworks and the semantic similarity measurement used in our tool. In Section 3, we investigate the performance of our software tool. Lastly, we discuss the results of our software tool in Section 4.

Chapter 2 Materials and Methods

2.1 Detecting Maximal Common Subgraphs

In order to detect network motifs, it is necessary to find network components which are topologically similar to a network motif given as a query. The problem of finding topological identity between graph subcomponents is called *subgraph isomorphism*, and it is computationally intractable (NP-hard). We adopted here a graph matching algorithm developed by McGregor [30]. It detects *maximal common subgraphs* (*MCS*), containing the largest number of common nodes and edges between the two graphs. For example, Fig. 2.1 shows the MCS between the two graphs. Solving the general problem of MCS is also computationally intractable (NP-hard) [32], but by using backtracking search, MCSs are found in most realistic problems.

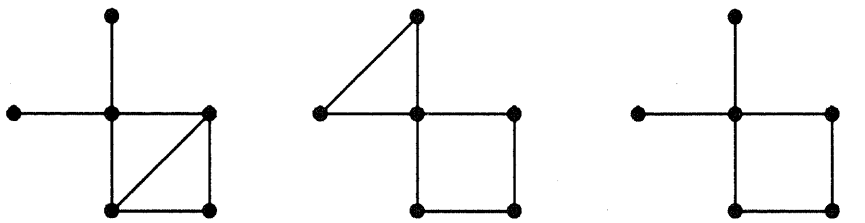


Fig. 2.1. Maximal Common Subgraph (MCS). The graph size is computed as the number of edges in this example.

Algorithm 2.1 shows how to find perfectly matching MCSs between graph G_1 and graph G_2. Let the number of nodes and edges in G_1 and G_2 be N_1, E_1 and N_2, E_2, respectively. Before starting the backtracking search, two matrices named as TRIED and MARCS are prepared. TRIED has $N_1 \times N_2$ elements corresponding to the nodes of G_1 and G_2. TRIED is initialized as all 0s (line 1), and is used to check explored node pairs in the backtracking search. For example, when node i in G_1 and node j in G_2 are temporally matched in a partially matched common subgraph, the element (i, j) in TRIED is checked as 1. MARCS has $E_1 \times E_2$ elements corresponding to the edges of G_1 and G_2. MARCS is initialized as all 1s (line 2). This implies that all edge pairs are possible to be matched. When it is found that edge i in G_1 and edge j in G_2 cannot match, the element (i, j) in MARCS is set to 0. A stack is prepared to store tentatively corresponding node pairs (line 3). The iterative variable i which means node i in G_1 is set to 1 (line 4). At the first step of this algorithm, node i of G_1 and node j of G_2 which are possible to correspond to each other are selected as a tentatively corresponding pair (line 6-7). At the next step, the value in position (i, j) of TRIED is set to 1 (line 8), and MARCS is refined on the basis of the following idea: edges connecting to node i in G_1 are not possible to correspond to edges not connecting to node j in G_2 when node i and node j are matched, and vice versa. Such elements meeting above condition in MARCS are set to 0. After the refinement of MARCS, if there is any row x whose elements are all 0s in MARCS, the edge x in G_1 is not possible to correspond to any edge in G_2. In this case, MARCS is restored to the previous state (line 19). If there is no such row, tentatively push the pair (i, j) into the stack and store MARCS as associated with i (line 11-12). If i equals N_1, all pairs in stack is one of perfectly matching subgraphs (line 14), otherwise i is incremented (line 16). If there is no untried node in G_2 to which node i of G_1 may correspond, then backtracking occurs: i is decremented, MARCS is restored to the state associated with i, and the stack is popped (line 22-24).

```
1.  Set TRIED to contain all 0s;
2.  Set MARCS to contain all 1s;
3.  Create stack S.
4.  i := 1;
5.  while ( i > 0 ) {
6.      if ( there exist any untried nodes in G2 to which node i of G1 may correspond ) {
7.          Select such node j in G2;
8.          TRIED (i, j) := 1;
9.          Refine MARCS based on the tentative correspondence (i, j);
10.         if ( there exists no column which contains all 0's in MARCS ) {
11.             Push the pair (i, j) into S;
12.             Store MARCS as associated with i;
13.             if ( i = N1) {
14.                 All pairs in S form one solution;
15.             } else {
16.                 i := i + 1;
17.             }
18.         } else {
19.             Restore MARCS to the previous state;
20.         }
21.     } else {
22.         i := i – 1;
23.         Restore MARCS associated with i;
24.         Pop S;
25.     }
26. }
```

Algorithm 2.1. Detection algorithm for perfectly matched MCS.

2.2 Semantic Similarity of Gene Ontology (SSG)

In order to find functionally similar network motifs, similarity between two proteins has to be defined. We introduce Gene Ontology (GO) annotations attached to proteins in order to determine that similarity, because GO describes all biological aspects of proteins including biological processes, cellular components and molecular functions.

GO is a hierarchical tree structure as in Fig. 2.2, which shows a fraction of GO tree. In GO tree, the deeper a term from the root of the tree, the more concrete its description. For example, a term 'protein binding' (GO:0005515) is more concrete than the term 'binding' (GO:0005488).

Lord et al. proposed *Semantic Similarity* [31] which is a statistical measure for determining quantitative similarity between two GO terms. They use the notion of *information content*, which is based on the concept that the more frequently a term is used, the less information it has. For instance, 'DNA binding' (GO:003677) is a more informative term than 'RNA binding' (GO:0003723), because the former is used 229 times in the biological context, while the latter is used 420 times (Fig. 2.2).

The ontological links of GO can also be exploited in the calculation of the information content for each concept. If the term 'DNA binding' (GO:0003677) occurs, then implicitly, the concept 'nucleic acid binding' (GO:0003676), 'binding' (GO:0005488) and 'molecular_function' (GO:0003674) also occurs, as well as any other terms which subsume it. Here, only the 'is-a' links are considered. Terms which do not have any 'is-a' link but have one or more 'part-of' links (these terms are called as *orphan terms*) are to be directly linked to the root of their taxonomic term.

The frequency of each term is shown in Fig. 2.2 (written as '*N*' in Fig. 2.2). These frequencies are counted by the GO annotation file published in Saccharomyces genome database (SGD) [33]. The frequency probability (written as '*P*' in Fig. 2.2) for each node is the value divided by the number of times any term occurs. The probability for the root node occurring must be 1. Then information content of each term is defined as the value calculated as –log score of its probability. It reflects reasonable distribution towards the frequency probabilities.

Semantic Similarity between two GO terms is defined as the information content of their common parental terms (see Eq. 2.1). For example, the Semantic Similarity between 'protein binding' (GO:0005515) and 'RNA binding' (GO:0003723) is –log(1.742) = 1.748. This is based on the idea that two terms can match each other with error probability correlating to the frequency probability of the common parental term. Because GO allows multiple parental terms for one term, there can be two or more common parental terms between any two

GO terms. In the case, the largest Semantic Similarity of common parental terms is adopted as the Semantic Similarity between them. We call Semantic Similarity of GO terms as *SSG* in order to distinguish from that of proteins mentioned in the next part.

$$sim(c_1, c_2) = -\log(\Pr(c_1, c_2)) \tag{2.1}$$

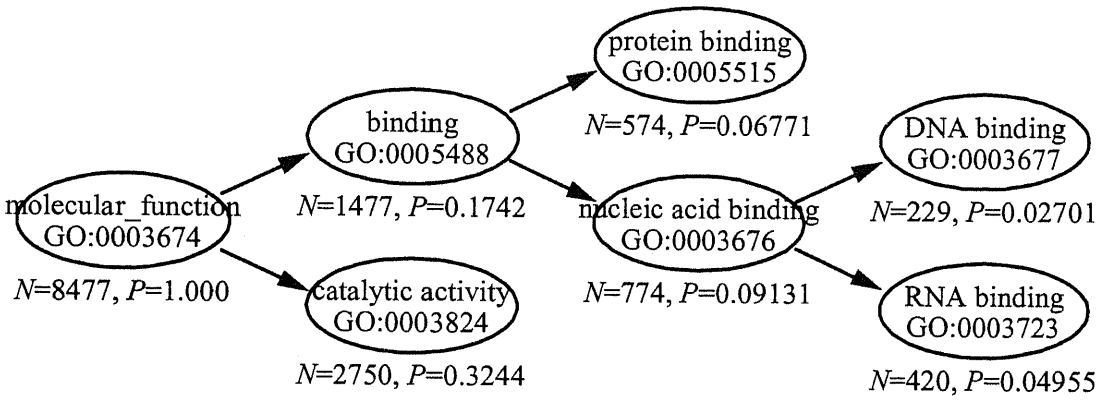


Fig. 2.2. A part of the Gene Ontology tree. Each ellipse indicates one GO term. The name of each term is in the upper line, and the ID number of each term is in the lower line. *N* indicates the frequency number of each term counted in the GO annotation file of SGD. *P* indicates the frequency probability of each term. *P* is the value *N* of the each term divided by *N* of the taxonomic term 'molecular function'.

2.3 Semantic Similarity of Proteins (SSP)

In the previous part, Semantic Similarity of GO terms (SSG) is defined. Similarity between two proteins is calculated by summing up SSGs among all of their GO terms. The similarity between proteins is defined as Semantic Similarity of proteins (called as *SSP* in this paper). As proteins are annotated with one or more GO terms, we introduce three methods for summing up SSGs: (1) simple average of SSGs (*simple-average method*), (2) maximum score of SSGs (*max-score method*) and (3) weighted average of SSGs (*weighted-average method*).

The simple-average method has already used by Lord et al. [31]. They assumed that proteins are fully annotated with GO terms, and therefore took the average SSG between all terms (see Eq. 2.2). They investigated this method by comparing with BLAST bit-score, and then it is concluded that both measures correlate to each other when measured against the 'molecular_function' aspect of GO. The simple-average method, however, has also a defect. If two proteins have dissimilar GO terms between them, the average of SSGs is decreased. Fig. 2.3 shows such a demonstration, in which (a) is the case that two proteins are annotated with only similar terms, and (b) is the case that two proteins are annotated with both similar and dissimilar terms. In the case (b), SSP between two proteins is decreased although their biological functions are considered to be close to each other.

$$sim(P_1, P_2) = \underset{c_1 \in P_1, c_2 \in P_2}{average}(sim(c_1, c_2)) \tag{2.2}$$

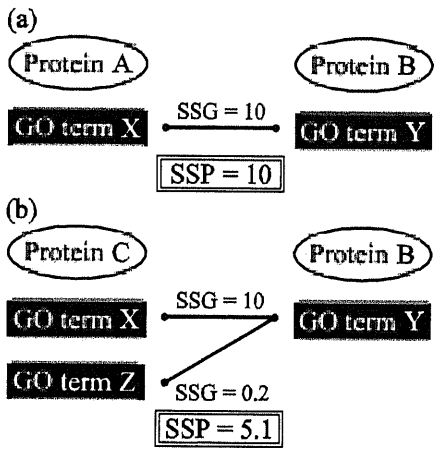


Fig. 2.3. Defect of the simple-average method. A, B and C denotes proteins, and X, Y and Z denotes GO terms annotated to proteins above them.

The max-score method adopts the maximal SSG as SSP between two proteins (see Eq. 2.3). This approach has already been used in WordNet by Resnik [34].

$$sim(P_1, P_2) = \max_{c_1 \in P_1, c_2 \in P_2} (sim(c_1, c_2)) \tag{2.3}$$

The weighted-average method is an improvement from the simple-average method. When calculating an average of SSGs, each SSG is weighted by the score of itself. This approach assumes that an SSG between two GO terms has value as large as –log score of frequency probability of its common parental term (see Eq. 2.4).

$$\begin{aligned} sim(P_1, P_2) &= \sum_{c_1 \in P_1, c_2 \in P_2} \left\{ sim(c_1, c_2) \times \frac{sim(c_1, c_2)}{\sum_{c_1 \in P_1, c_2 \in P_2} sim(c_1, c_2)} \right\} \\ &= \frac{\sum_{c_1 \in P_1, c_2 \in P_2} sim(c_1, c_2)^2}{\sum_{c_1 \in P_1, c_2 \in P_2} sim(c_1, c_2)} \end{aligned} \tag{2.4}$$

These three methods are similar to distance measurements used in hierarchical clustering. That is, the simple-average method, the max-score method and the weighted-average method corresponds to the average linkage method, the simple linkage method and the weighted-average linkage method, respectively (Fig. 2.4).

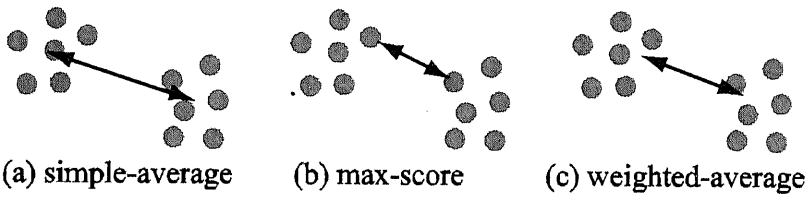


Fig. 2.4. Image diagram of methods for SSP. GO terms annotated two proteins are indicated gray circles being separated into two groups. The closer a pair of two gray circles between two groups, the larger the SSG between them. Each Arrow indicates SSP calculated by each calculation method.

We investigated performances of these three methods.

2.4 Searching Topologically and Functionally Similar Networks

To search functionally similar protein networks, Algorithm 1 needs to be modified in order to combine itself with SSP. Suppose that we want to find functionally similar MCSs between a given protein network (query) and a large protein-protein interaction network (target) (see Fig. 2.5). We define the total sum of SSPs between corresponding protein pairs in an MCS as "MCS score". Note that these SSPs can be calculated by three methods mentioned in the previous part. Eq. 2.5 is the definition of MCS score for one MCS: (i, j) represents a node in the MCS, P represents the query graph, and Q represents the matched subgraph in the target graph.

$$MCS\ score = \sum_{(i,j)\in MCS} sim(P_i, Q_j) \tag{5}$$

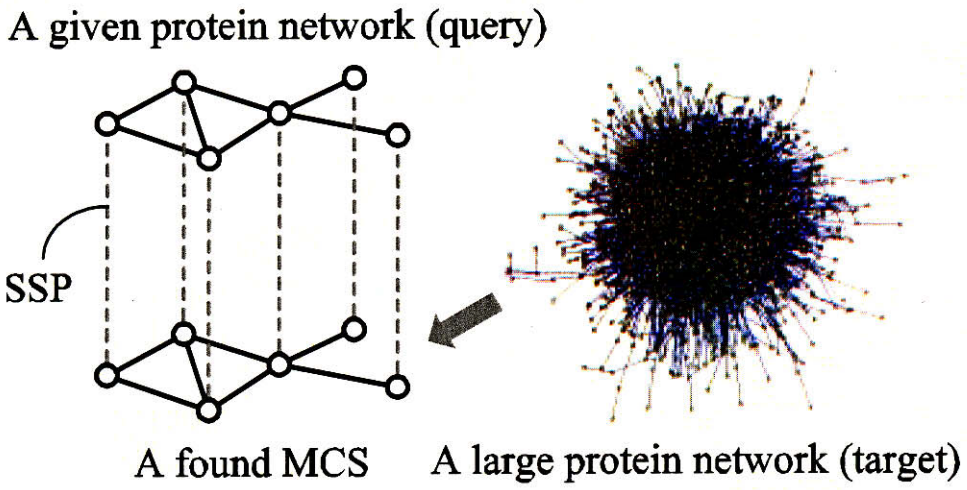


Fig. 2.5. Diagram of Modified Algorithm 1.

Accordingly, we modified Algorithm 1 so that it can deal with real-valued MCS scores (Algorithm 1'). First, TRIED contains real values instead of binaries (0 or 1) in Algorithm 1. TRIED (i, j) is to be set to $-\infty$ if node pair (i, j) does not have possible correspondence or if it has already been explored.

In initializing the TRIED matrix, the SSPs of all protein pairs between the query and the target graphs are calculated in advance and set the values into corresponding elements in TRIED. To reduce the search space, two simple bounds are applied. One is a threshold for SSP. If the SSP of a node pair (i, j) is less than a specified threshold, TRIED (i, j) is set to $-\infty$. The other is the bound for node-degrees. If the degree of node i in the query graph is more than that of node j in the target graph, then the node pair (i, j) is not achievable, and the TRIED (i, j) is set to $-\infty$.

Second, we prepare a variable *Score* to represent the sum of SSP (line 4'). When tentatively corresponding node pair (i, j) is selected in the following step, its SSP will be added up to *Score*.

We implemented an additional bounding method by keeping a potentially maximal MCS score, *maxScore*. This is the upper bound of the *Score* for current TRIED, calculated by adding the best SSP for each node k of G_1 (k is larger than i) up to the current *Score* (line 6'-a). For example, let us assume that we want to find top 10 results out of many topologically identical pathways. If *maxScore* of a candidate is less than the current top 10 MCS scores, then backtracking is invoked immediately (line 6'-b).

If the criterion of 6'-b is satisfied, a node with the best SSP among candidates in G_2 is selected (line 7'). Then the SSP of the pair (i, j) is added up to *Score*, and TRIED (i, j) is set to $-\infty$ (line 8').

When MARCS is stored or restored in association with i, *Score* is also stored or restored simultaneously (line 12', 19' and 23').

4'-a $i := 1$;

4'-b *Score* := 0.0;

6'-a Calculate *maxScore* by current TRIED;

6'-b **if** (*maxScore* > top10 solution's score ∩ there exists any untried nodes in G_2 to which node i of G_1 may correspond) {

7' Select node j which has the best SSP out of such G_2 nodes;

8'-a *Score* := *Score* + TRIED (i, j);

8'-b TRIED $(i, j) := -\infty$;

12' Store MARCS and *Score* as associated with i;

19' Restore MARCS and *Score* to the previous state;

23' Restore MARCS and *Score* associated with i;

Algorithm 2.1'. Modification of Algorithm 2.1. Only modified lines are shown here. This algorithm can find top 10 functionally similar protein networks with a query network.

2.5. Materials

We used *S. cerevisiae* interaction data in the Database of Interacting Proteins (DIP) [35] as the target network to investigate our new tool. DIP is a database that documents a number of protein-protein interactions experimentally determined by such as two-hybrid screen, protein complex purification and so forth. Therefore, it has some false-positive rate within its data set. The filename of the data is 'yeast20040704.lst', in which there are 4707 proteins and 15138 interactions (multiple interactions are removed).

Gene Ontology data was obtained from Gene Ontology Consortium's web site. The filename is 'go_200406-termdb.xml', in which there are 17,552 terms. Its creation month is 2004/06.

Gene Ontology annotation data was obtained from Gene Ontology Consortium's web site. We used GO annotation data for *S. cerevisiae* made by Saccharomyces Genome Database [33], in which there are 29,105 GO annotations for *S. cerevisiae*. Its version is 1.910, and its creation date is 2004/06/17.

Chapter 3 Results

We implemented a software tool which performs the methods described in Section 2. To evaluate our software tool, we searched MAPK cascade in yeast protein interaction network. MAPK cascade is a well-studied signal transduction cascade, and is well-conserved from yeast to mammalian. In the cascade, MAP Kinase Kinase Kinase (MEKK) phosphorylates MAP Kinase Kinase (MEK), and then MEK phosphorylates MAP Kinase (MAPK) (Fig. 3.1).

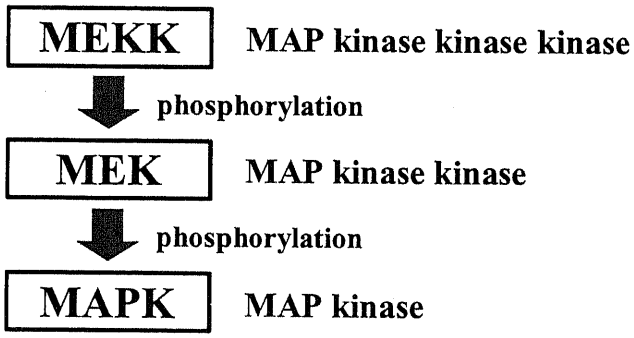


Fig. 3.1. MAPK Cascade. MAP Kinase Kinase Kinase (MEKK) phosphorylates MAP Kinase Kinase (MEK), and the activated MEK phosphorylates MAP Kinase (MAPK)

When a small protein interaction network is input as a query, our software tool outputs functionally similar MCSs with high MCS score from a target network. We first investigated the performance of three types of semantic similarities by searching known MAPK cascade. Fig. 3.2 shows four known pathways in MAPK cascade. We chose the Filamentation / Invasion pathway (STE11-STE7-KSS1) as a query network. We chose yeast protein-protein interaction networks in DIP [5] as a target network (this network contains 4738 proteins and 15129 interactions). Since there is no interaction between STE11 and STE7 in the network, only the High Osmolarity Growth pathway and the Cell Integrity pathway should be detected from the yeast protein-protein interaction network. We chose GO annotation file of SGD [13] as both protein annotation list and GO term occurrence list.

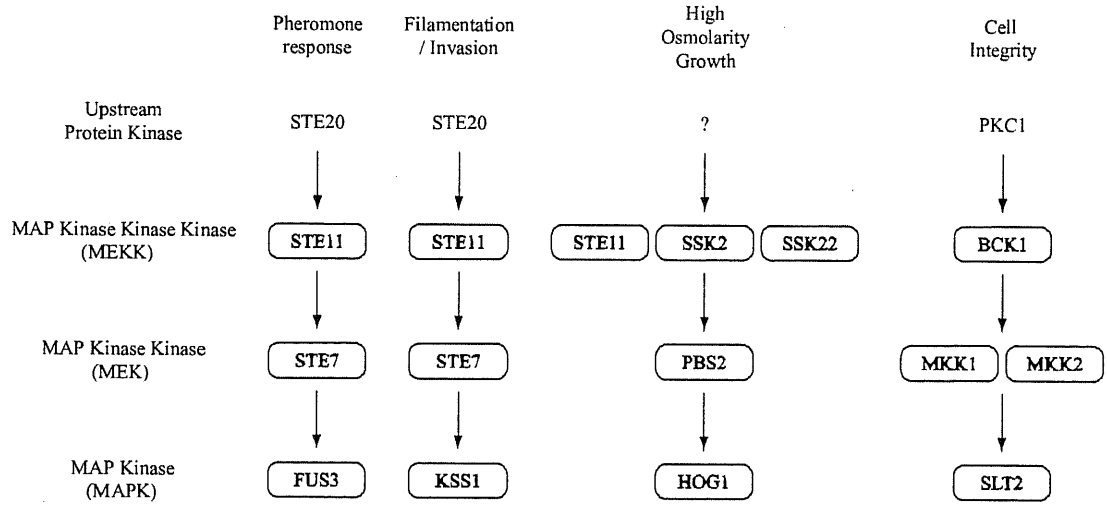


Fig. 3.2. Four known pathways of MAPK cascade in yeast. In the High Osmolarity Growth pathway, STE11, SSK2 and SSK22 phosphorylates PBS2 independently when they are activated. Similarly in the cell integrity pathway, BCK1 phosphorylates MKK1 and MKK2, and activated MKK1 and MKK2 phosphorylates SLT2 independently.

By querying Filamentation / Invasion pathway, we obtained and inspected top 100 ranking results. The Ranking results by the simple-average method, the max-score method and the weighted-average method are shown in Fig. 3.3, Fig. 3.4 and Fig. 3.5, respectively. Table 3.1, Table 3.2 and Table 3.3 shows MCS scores of pathways in MAPK cascades calculated by manual Semantic Similarity measurements.

There were only two correct answers in the 4th and 18th by the simple-average method. On the other hand, the result by the max-score method had five correct answers in the top 5 ranks. In addition, all the searched pathways ranked in top 13 were composed of proteins in MAPK cascade with their real order converted. These results indicate that this method provides the best matches within GO annotations. One drawback of this method is that the distribution of its ranking results is discrete because of reduction of information by its protocol.

The result of the weighted-average method had correct answers in the 4th, 13th, 21st, 38th and 60th ranks. Therefore, it is considered that the weighted-average method has low false-positive and low false-negative rates.

Table 3.1. Each MCS score of all the known pathways in MAPK cascade against the Filamentation / Invasion pathway is computed by the simple-average method, and written in the right-most column. The Pheromone Response pathway and the Filamentation / Invasion pathway do not exist in yeast protein-protein interaction network in DIP. Still, MCS scores of these pathways are computed and written in parentheses.

	MEKK	MEK	MAPK	MCS score
Pheromone Response	STE11	STE7	FUS3	(6.78)
Filamentation / Invasion	STE11	STE7	KSS1	(6.99)
High Osmolarity Growth	STE11	PBS2	HOG1	5.13
	SSK2			4.18
	SSK22			4.28
Cell Integrity	BCK1	MKK1	SLT2	6.30
		MKK2		5.84

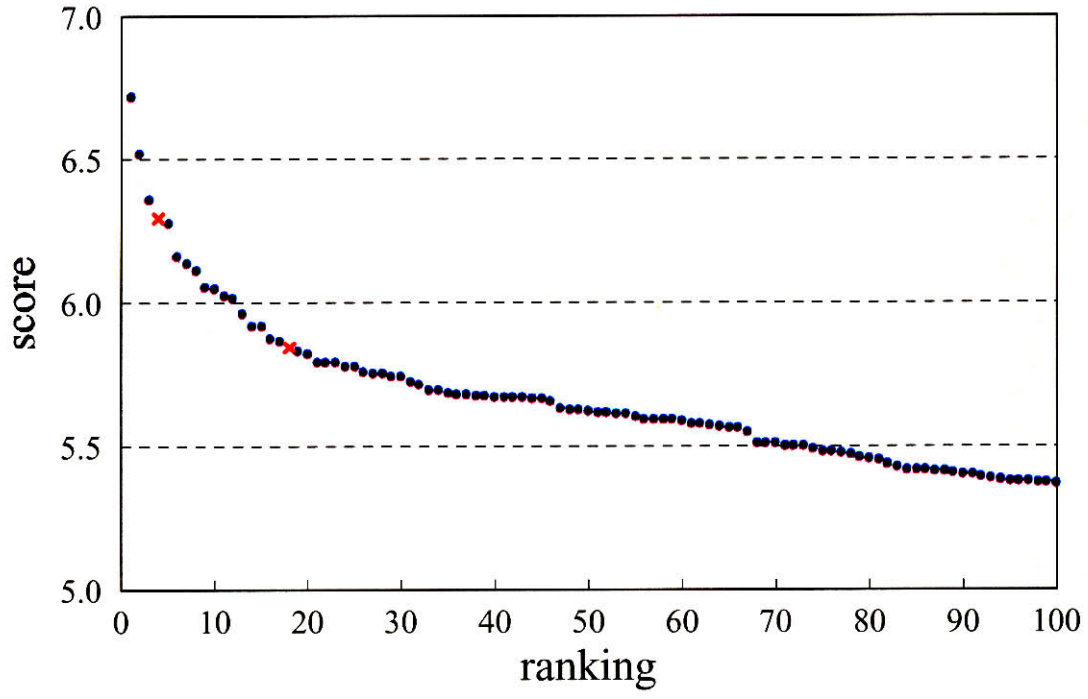


Fig. 3.3. Ranking of the top 100 results in the case that the simple-average method is used to calculate Semantic Similarities between proteins. Blue points are incorrect answers and red crosses are correct ones. Two known MAPK pathways are ranked in the 4th and 18th.

Table 3.2. Each MCS score of all the known pathways in MAPK cascade against the Filamentation / Invasion pathway is computed by the max-score method.

	MEKK	MEK	MAPK	MCS score
Pheromone Response	STE11	STE7	FUS3	(22.52)
Filamentation / Invasion	STE11	STE7	KSS1	(22.52)
High Osmolarity Growth	STE11	PBS2	HOG1	22.52
	SSK2			22.52
	SSK22			22.52
Cell Integrity	BCK1	MKK1	SLT2	22.52
		MKK2		22.52

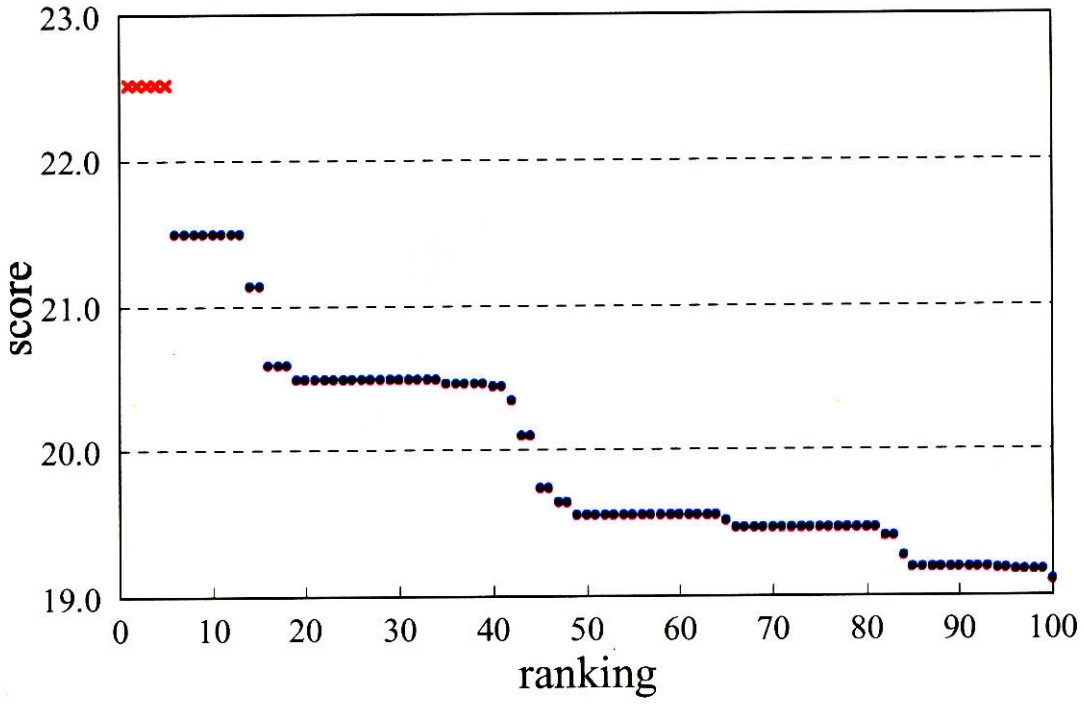


Fig. 3.4. Ranking of the top 100 results in the case that the maximum-score method is used to calculate Semantic Similarities between proteins. Three known MAPK pathways are ranked in the top 5.

Table 3.3. Each MCS score of all the known pathways in MAPK cascade against the Filamentation / Invasion pathway is computed by the weighted-average method.

	MEKK	MEK	MAPK	MCS score
Pheromone Response	STE11	STE7	FUS3	(15.96)
Filamentation / Invasion	STE11	STE7	KSS1	(15.72)
High Osmolarity Growth	STE11	PBS2	HOG1	15.19
	SSK2			15.01
	SSK22			15.67
Cell Integrity	BCK1	MKK1	SLT2	14.45
		MKK2		14.71

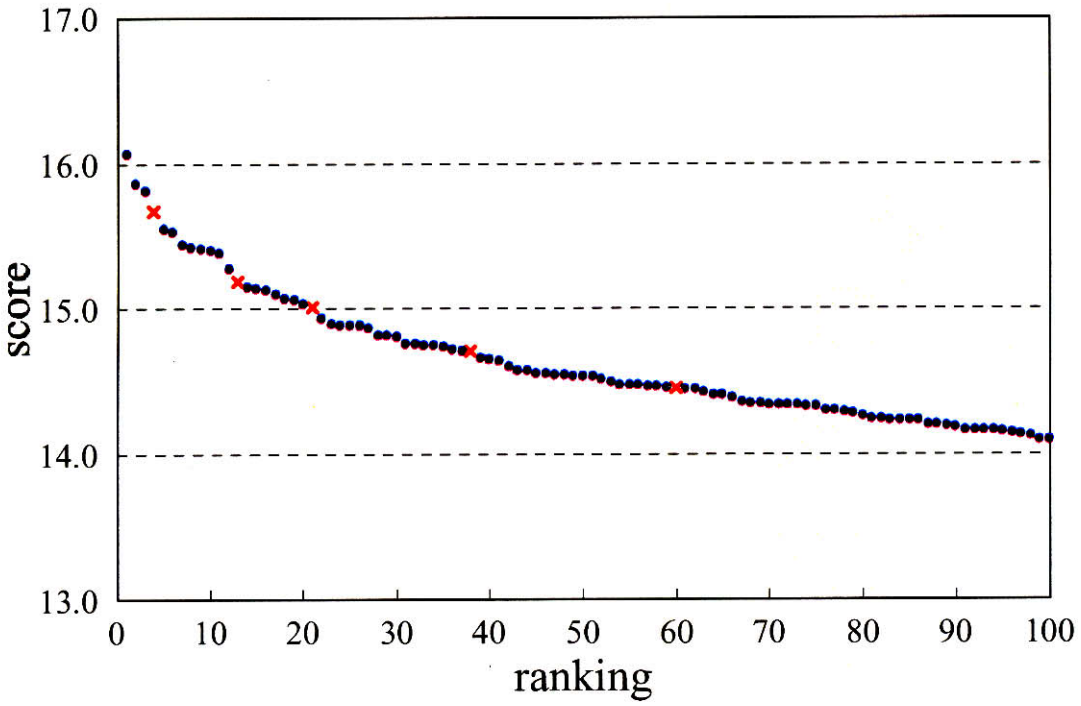


Fig. 3.5. Ranking of the top 100 results in the case that the weighted-average method is used to calculate Semantic Similarities between proteins. Three known MAPK pathways are ranked in the 4th, 13th, 21st, 38th and 60th respectively.

For evaluation of our approach, we compared the performance of our software tool with that of PathBLAST. PathBLAST uses yeast protein-protein interaction network in the DIP database as the same as our software tool. We posted the Filamentation / Invasion pathway as a query into the PathBLAST server, and then obtained the top 100 results in the order of similarity scores measured by PathBLAST (Fig. 3.6). All of the known pathways in MAPK cascade should be detected by PathBLAST because PathBLAST allows a gap in pathway alignment. However, only one correct answer (BCK1-MKK2-SLT2) was found in the 2nd rank. The detected pathways ranked around the top rank are mainly composed of proteins not-associated with MAPK cascade. These results indicate that PathBLAST may have high false-positive and high false-negative rates.

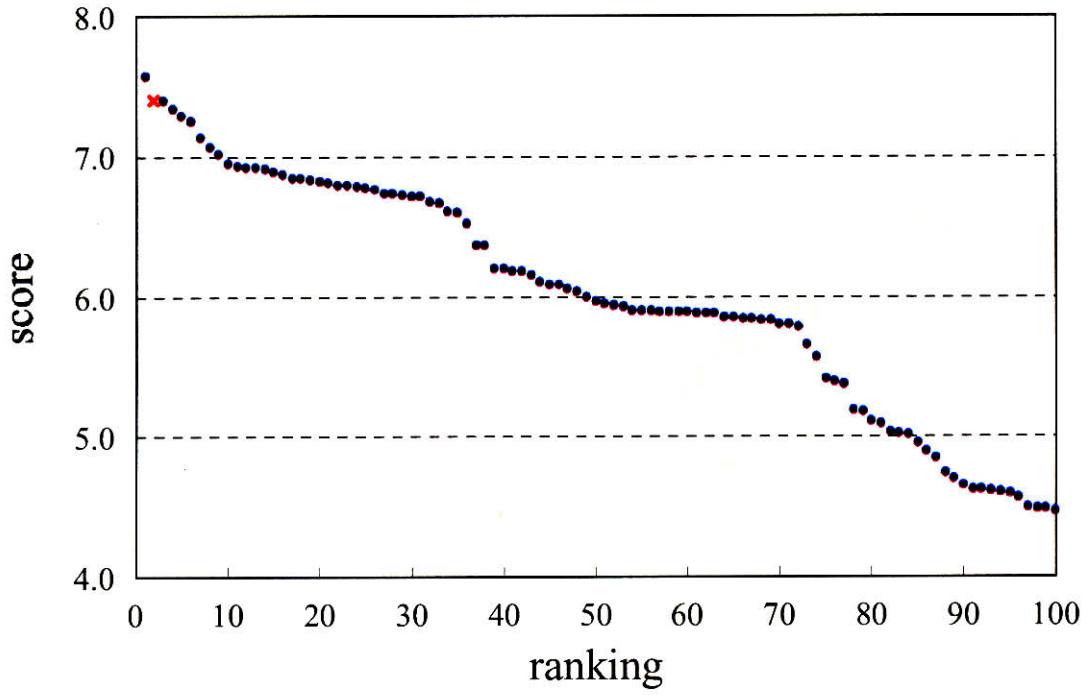


Fig. 3.6. Ranking of the top 100 results by PathBLAST. Only one known pathway of MAPK cascade is ranked in the 2nd rank.

Chapter 4 Discussions

According to the above results, it is found that the max-score method and the weighted-average method are better than the simple-average method for detecting functionally similar network components. By the max-score method, correct answers were obtained in the top ranks, but instead large amount of information of GO annotations were ignored. By the weighted-average method, the results had several false-positives with little loss of information. It was also demonstrated that our software tool using these two methods provides better performance than PathBLAST.

As an application of our approach, combining the max-score method and weighted-average method seems to improve the performance of our software tool. Generally, there is no information about correct answers in searching biologically similar modules, while the ranking result of max-score method provides the significance of correctness to each result of the weighted-average method because of its high correctness.

A superior point of our software tool is that it uses only GO annotations (no amino-acid sequences) when calculating protein similarities and it is applicable to genetically uncharacterized functions (proteins). For example, Ito et al. found a possibly existing protein-protein interaction network which has a biological function of autophagy [11]. Our software tool can search such a virtual network given a target protein pattern.

On the other hand, there are two problems in our software tool. The first problem is that the accuracy of search results depends on the reliability of GO. Well-studied proteins tend to contain more GO annotations than not-well-studied proteins. For example, there are a lot of GO annotations in proteins of MAPK cascade. This causes SSGs between GO terms of MAPK cascade to decline unexpectedly. In order to avoid this problem, augmenting GO annotations through prediction has a beneficial effect by reducing the bias mentioned above.

The second problem is that our tool does not use confidence scores for protein-protein interactions. PathBLAST introduces confidence score to each protein-protein interaction, because protein-protein interaction data in DIP have a lot of false-positives. On the other hand, our strategy relies on the MCS algorithm, whose notion of maximality does not easily suit the integration of edge-weights. The resolution of the above problems is ongoing in our laboratory.

Bibliography

[1] Gene Ontology Consortium, "Creating the Gene Ontology Resource: Design and Implementation", *Genome Res.*, 2001, vol. 11, pp. 1425-1433.

[2] B. P. Kelley, B. Yuan, F. Lewitter et al., "PathBLAST: a tool for alignment of protein interaction networks", *Nucleic Acids Res.*, 2004, vol. 32, pp. W83-W88.

[3] M. C. Gustin, J. Albertyn, M. Alexander et al., "MAP Kinase Pathways in the Yeast Saccharomyces cerevisiae", *Microbiol. Mol. Biol. Rev.*, 1998, vol. 62, pp. 1264-300.

[4] S. F. Altschul, W. Gish, W. Miller et al., "Basic local alignment search tool", *J. Mol. Biol.*, 1990, vol. 215, pp. 403-410.

[5] L. F. Wu, T. R. Hughes, A. P. Davierwala et al., "Large-scale prediction of Sacharomyces cerevisiae gene function using overlapping transcriptional clusters", *Nat. Genet.*, 2002, vol. 31, pp. 255-265.

[6] S. Ghaemmaghami, W. K. Huh, K. Bower et al., "Global analysis of protein expression in yeast", *Nature*, 2003, vol. 425, pp. 737-741.

[7] A. H. Tong, G. Lesage, G. D. Bader et al., "Global mapping of the yeast genetic interaction network", *Science*, 2004, vol. 303, pp. 808-813.

[8] A. Kumar, S. Agarwal, J. A. Heyman et al., "Subcellular localization of the yeast proteome", *Genes. Dev.*, 2002, vol. 16, pp. 707-719.

[9] T. I. Lee, N. J. Rinaldi, F. Robert et al., "Transcriptional regulatory networks in Saccharomycs cerevisiae", *Science*, 2002, vol. 298, pp. 799-804.

[10] P. Uetz, L. Giot, G. Cagney et al., "A comprehensive analysis of protein-protein interactions in Saccharomyces cerevisiae", *Nature*, 2000, vol. 403, pp. 623-627.

[11] T. Ito, T. Chiba, R. Ozawa et al., "A comprehensive two-hybrid analysis to explore the yeast protein interactome", *Proc. Natl Acad. Sci. USA*, 2001, vol. 98, pp. 4569-4574.

[12] A. C. Gavin, M. Bosche, R. Krause et al., "Functional organization of the yeast proteome by systematic analysis of protein complexes", *Nature*, 2002, vol. 415, pp. 141-147.

[13] Y. Ho, A. Gruhler, A. Heilbut et al., "Systematic identification of protein complexes in Saccharomyces cerevisiae by mass spectrometry", *Nature*, 2002, vol. 415, 180-183.

[14] R. J. Cho, M. J. Campbell, E. A. Winzeler et al., "A genome-wide transcriptional analysis of the mitotic cell cycle", *Mol. Cell.*, 1998, vol. 2, pp. 65-73.

[15] T. R. Hughes, M. J. Marton, A. R. Jones et al., "Functional discovery via a compendium of expression profiles", *Cell*, 2000, vol. 102, pp. 109-126.

[16] A. J. Enright, I. Iliopoulos, N. C. Kyrpides et al., "Protein interaction maps for complete genomes based on gene fusion events", *Nature*, 1999, vol. 402, pp. 86-90.

[17] E. M. Marcotte, M. Pellegrini, H. L. Ng, "Detecting protein function and protein-protein interactions from genome sequences", *Science*, 1999, vol. 285, pp. 751-753.

[18] R. Overbeek, M. Fonstein, M. D'Souza et al., "The use of gene clusters to infer functional coupling", *Proc. Natl Acad. Sci. USA*, 1999, vol. 96, pp. 2896-2901.

[19] T. Dandekar, B. Snel, M. Huynen et al., "Conservation of gene order: a fingerprint of proteins that physically interact", *Trends Biochem. Sci.*, 1998, vol. 23, pp. 324-328.

[20] M. Pellegrini, E. M. Marcotte, M. J. Thoompson et al., "Assigning protein functions by comparative genome analysis: protein phylogenetic profiles", *Proc. Natl Acad. Sci. USA*, 1999, vol. 96, pp. 4285-4288.

[21] M. A. Huynen, P. Bork, "Measuring genome evolution", *Proc. Natl Acad. Sci. USA*, 1998, vol. 95, pp. 5849-5856.

[22] L. H. Hartwell, J. J. Hopfield, S. Leibler et al., "From molecular to modular cell biology", *Nature*, 1999, vol. 402, pp. C47-C52.

[23] V. Spirin & L. A. Mirny, "Protein complexes and functional modules in molecular networks", *Proc. Natl Acad. Sci. USA*, 2003, vol. 100, pp. 12123-12128.

[24] J. J. Han, N. Bertin, T. Hao et al., "Evidence for dynamically organized modularity in the yeast protein-protein interaction network", *Nature*, 2004, vol. 430, pp. 88-93.

[25] S. S. Shen-Orr, R. Milo, S. Mangan et al., "Network motifs in the transcriptional regulation network of Escherichia coli", *Nat. Genet.*, 2002, vol. 31, pp. 64-68.

[26] R. Milo, S. Shen-Orr, S. Itzkovitz et al., "Network Motifs: Simple Building Blocks of Complex Networks", *Science*, 2002, vol. 298, pp. 824-827.

[27] T. I. Lee, N. J. Rinaldi, F. Robert et al., "Transcriptional regulatory networks in Saccharomyces cerevisiae", *Science*, 2002, vol. 298, pp. 799-804.

[28] B. P. Kelley, R. Sharan, R. M. Karp et al., "Conserved pathways within bacteria and yeast as revealed by global protein network alignment", *Proc. Natl Acad. Sci. USA*, 2003, vol. 100, pp. 11394-11399.

[29] C. von Mering, R. Krause, B. Snel et al., "Comparative assessment of large-scale data sets of protein-protein interactions", *Nature*, 2002, vol. 417, pp. 399-403.

[30] J. J. McGregor, "Backtrack search algorithms and the maximal common subgraph problem", *Software - Practice and Experience*, 1982, vol. 12, pp. 23-34.

[31] P. W. Lord, R. D. Stevens, A. Brass et al., "Investigating semantic similarity measures across the Gene Ontology: the relationship between sequence and annotation", *Bioinformatics*, 2003, vol. 19, pp. 1275-1283.

[32] M. R. Garey & D. S. Johnson, "Computers and intractability: a guide to the theory of NP-completeness", 1979, W.H. Freeman & Co.

[33] S. S. Dwight, R. Balakrishnan, K. R. Christie et al., "Saccharomyces genome database: underlying principles and organization", *Brief Bioinform.*, 2004, vol. 5, pp. 9-22.

[34] P. Resnik, "Semantic similarity in a taxonomy: an informationbased-measure and its application to problems of ambiguity in natural language", *J. Artif. Intelligence Res.*, 1999, vol. 11, pp. 95-130.

[35] I. Xenarios, L. Salwinski, X. J. Duan et al., "DIP: The Database of Interacting Proteins. A research tool for studying cellular networks of protein interactions", *Nucleic Acids Res.*, 2002, vol. 30, pp. 303-305.

Appendix GO Viewer and its Library

GO Viewer is a Java-based package for (1) browsing Gene Ontology (GO) tree and (2) searching GO terms or annotations. This package also includes utilities associated with GO: loaders of GO files, data structures associated with GO, and programs that compute similarity between two GO terms and gene products (i.e. proteins or ORFs). The tool is freely available and users can easily import this package into their own programs. This tool is freely available.

A.1 Motivation

Gene Ontology [1] is an ontology aimed to create consistent descriptions of gene products (i.e. proteins or ORFs) in different databases. Indeed, GO has been used in many biological databases in the world (e.g. SGD [2], FlyBase [3] and WormBase [4]). In order to make use of GO, many tools have been developed and distributed at the Gene Ontology Consortium's web site (http://www.geneontology.org/). Most of these tools are designed mainly as two types of application software : (1) a web-based application for browsing GO terms (e.g. AmiGO : http://www.godatabase.org/cgi-bin/amigo/go.cgi) or (2) a specialized program for the analysis of biological data including gene expression data and so on (e.g. GoMiner [5]). They cannot be customized and imported into users' programs easily due to the complicated architecture. Therefore, we present a simple and user-friendly package which can be customized and imported at users' will. This package includes (1) GO Viewer for browsing GO terms or gene products and (2) utilities associated with GO: loaders of GO files, data structures associated with GO, and programs computing similarity between two GO terms and gene products.

A.2 Browsing GO Terms And Gene Products

GO Viewer is a useful application for browsing GO terms and gene products (Fig. A.1). Its main window shows hierarchical view of GO. Users can select a term, and then can open the detail window of the term. The detail window contains the definition of the term, annotated products, tree view and graphical view. Users can search any term or product in the search window by substring match or exact match. Simultaneously, users can customize species, data sources, and evidence code in the filter window. After search has finished, the result

window opens and users can select any term or product to see its detail. The gene product's detail window contains product's definition, annotated terms, tree view and graphical view.

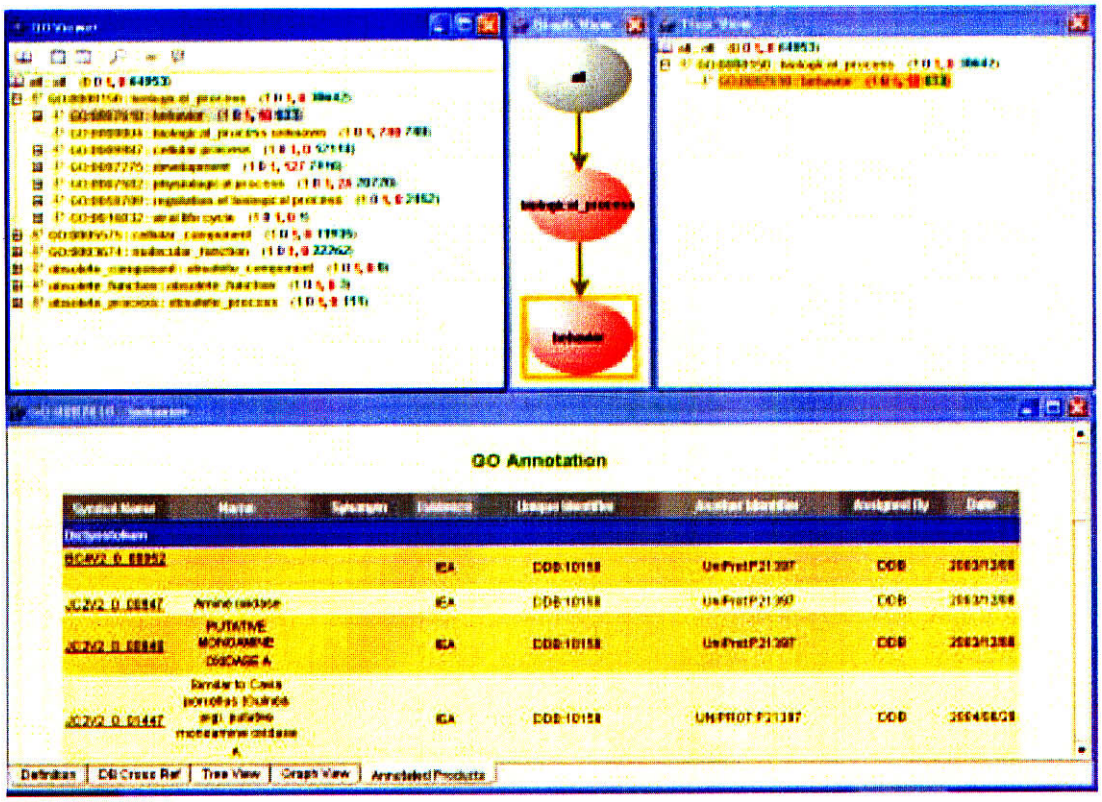


Fig. A.1. Graphical User Interface of GO Viewer

A.3 References

[1] Gene Ontology Consortium, "Creating the Gene Ontology Resource: Design and Implementation", *Genome Res.*, 2001, vol. 11, pp. 1425-1433.

[2] S. S. Dwight, R. Balakrishnan, K. R. Christie et al., "Saccharomyces genome database: underlying principles and organization", *Brief Bioinform.*, 2004, vol. 5, pp. 9-22.

[3] R. A. Drysdale, M. A. Crosby, W. Gelbart et al., "FlyBase: genes and gene models", *Nucleic Acids Res.*, 2005, vol. 33, pp. D390-D395.

[4] N. Chen, T. W. Harris, I. Antoshechkin et al., "WormBase: a comprehensive data resource for Caenorhabditis biology and genomics", *Nucleic Acids Res.*, 2005, vol. 33, pp. D383-D389.

[5] B. R. Zeeberg, W. Feng, G. Wang et al., "GoMiner: a resource for biological interpretation of genomic and proteomic data", *Genome Biol.*, 2003, vol. 4, R28.

A Manual for GOViewer

Note of GOViewer : pluggable viewer for Gene Ontology
version 1.0 (2005-1-20)

Hiroyuki Oka
oka@cb.k.u-tokyo.ac.jp
Department of Computational Biology
Graduate School of Frontier Sciences
The University of Tokyo, Japan

Contents

1. Introduction and Preliminaries

1.1 What's the GO?

The Gene Ontology (GO) is an ontology aimed to create consistent descriptions of gene products (proteins or ORFs) in multiple databases. The detail of GO is written in Gene Ontology Consortium's web site (http://www.geneontology.org/).

1.2 What's the GOViewer?

In order to make use of GO, many tools have been developed and distributed at Gene Ontology Consortium's web site (http://www.geneontology.org/). Most of them are designed mainly as the two types of application : (1) a web-based application for browsing GO terms or (2) a specialized program for the analysis of biological data including gene expression data and so on. They cannot be customized and imported into user's programs easily due to the complicated architecture. Therefore, we present a simple and user-friendly package which can be customized and imported as users want.

1.3 The feature of GOViewer

GOViewer can do these things:

(1) To read both OBO and XML files of GO data.

(2) To show GO tree as both Tree View and Graph View.

(3) To show data of gene products as GO annotations attached to GO terms.

(4) To show details of both GO terms and gene products.

(5) Users can import their own programs easily.

GOViewer has been developed with the assumption that users can browse and get GO data and can import their own programs. Therefore, GOViewer is one of the best packages for users aiming such things.

1.4 The GOViewer environment

GOViewer is a Java based, cross-platform application. The version of Java is required 1.4 or greater. If you don't have Java in your PC, Please get Java from Sun's web site (http://www.sun.com/).

1.5 Getting start of GOViewer

Please download the compressed file of GOViewer from our homepage (http://www.cb.k.u-tokyo.ac.jp/aritalab/oka/goviewer/). After decompressing the file, following files will be extracted:

GOViewer.jar	(The GOViewer application package)
run.bat	(an executable file for Windows)
run.sh	(an executable file for Unix)

According to your PC's platform, you should execute different operation to start GOViewer:

(1) In the case of Windows
Double-click on the "**run.bat**" file to start GOViewer.

(2) In the case of Unix
Execute the "**run.sh**" file on the shell window to start GOViewer.

(3) In the case of any other platforms
Execute following command on the shell window:

java -Xms128m -Xmx128m -jar GOViewer.jar

After starting GOViewer, the window shown in Fig. 1.5.1 will be seen on your display.

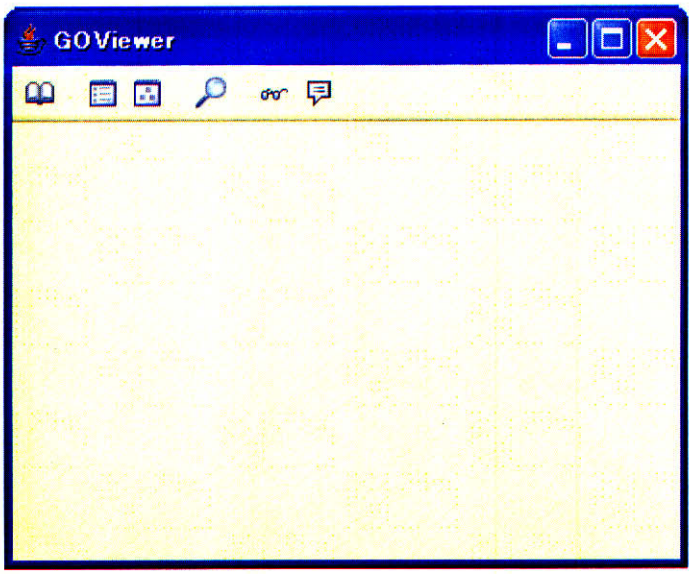


Fig. 1.5.1. The beginning window of GOViewer

At first, loading GO data is needed for showing GO tree on the main window. GO data can be obtained at Gene Ontology Consortium's web site. At this moment (2005-1-20), GO data can be downloaded from http://www.geneontology.org/index.shtml#downloads. GOViewer can parse OBO format files and XML format files, not GO format files.

When GO data is prepared, please click the leftmost button ( ) on the toolbar. By clicking this button, the dialog shown in Fig. 1.5.2 will be displayed, in which you can select GO data file to open. At first, please select the format of GO data at "Files of Type:" box. Next please select the GO data file at the central panel of the dialog. When XML format is selected, DTD file is not required to specify. GOViewer ignores DTD file for both ease and simple. GOViewer can handle uncompressed files, ZIP compressed files and GZIP compressed files. When GO data file is selected, please click the "Open" button. The GO tree will be displayed on the original window as shown in Fig. 1.5.3.

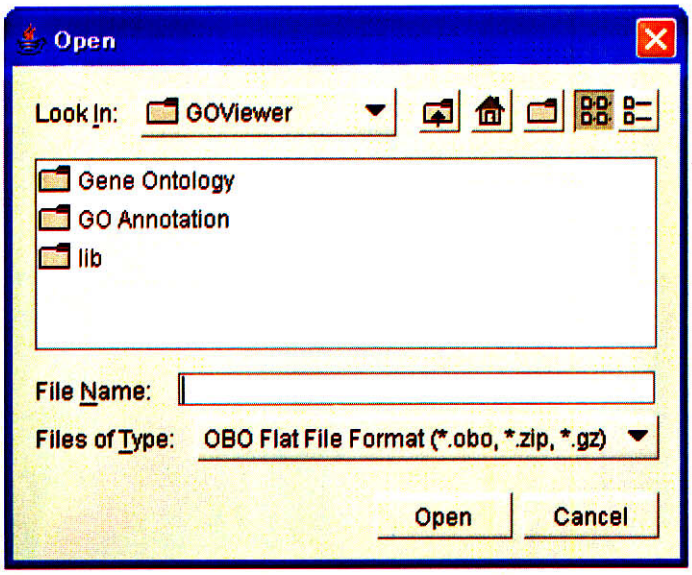


Fig. 1.5.2. The Open Dialog for GO data

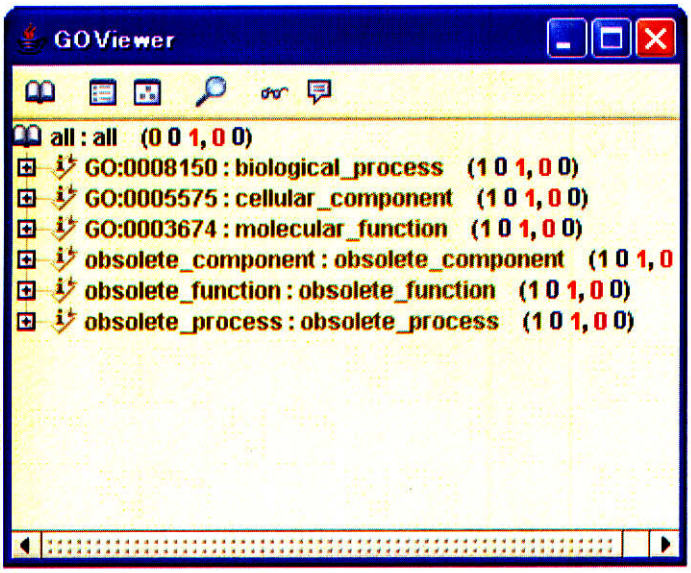


Fig. 1.5.3. The main window after reading GO data

2. Functions of GOViewer

2.1 The Tree View and the Graph View

Tree View and Graph View are made for browsing GO tree. In this section, how to use both views is explained.

2.1.1 The Tree View

Tree View is a tree-formed representation of GO tree. The Tree View shown in Fig. 2.1.1.1 is a Tree View displayed in the main window. You can browse and can get most of information about Gene Ontology from Tree Views in the main panel. When the mouse pointer is over a GO term, detailed information of the term will be displayed as Fig. 2.1.1.2.

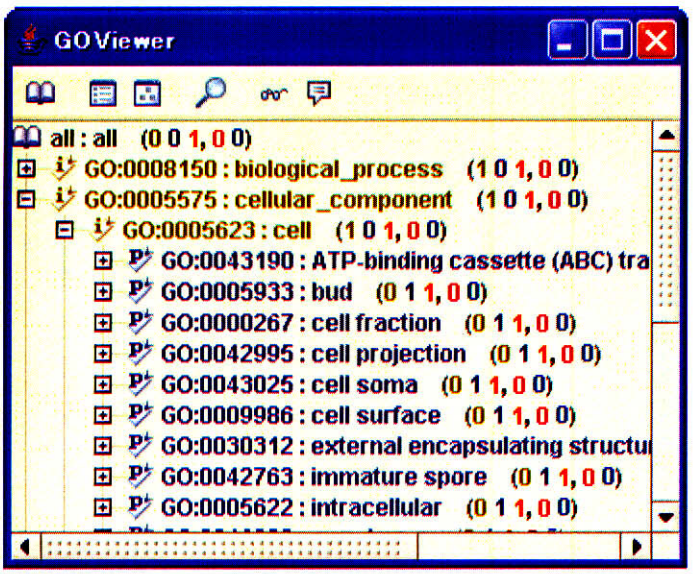


Fig. 2.1.1.1. A Tree View in the main window

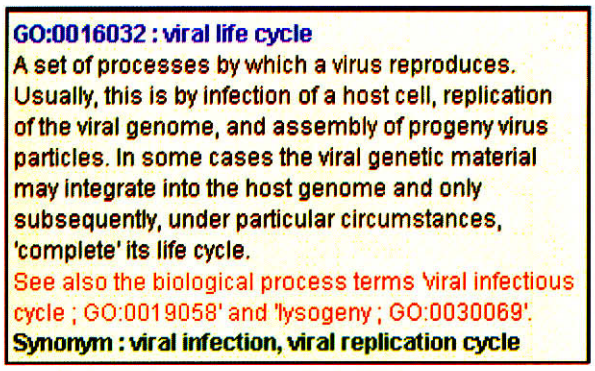


Fig. 2.1.1.2. Detailed information of the GO term "GO:0016032 : viral life cycle"

Tree Views are displayed in these panels and windows: the main panel of GOViewer mentioned above, the TreeView Window, detailed windows of GO terms and detailed windows of gene products (mentioned later). The denotion of GO terms in Tree Views is explained in Fig. 2.1.1.3.

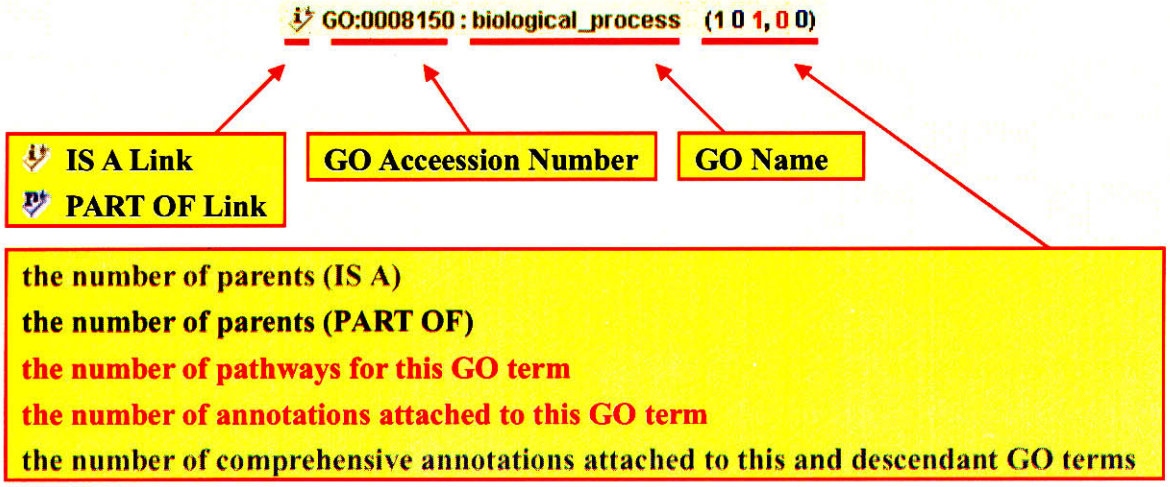


Fig. 2.1.1.3. Denotion of GO terms

When a GO term on Tree View is right-clicked, a pop-up menu shown in Fig. 2.1.1.4 will be popped. If "Show Detail" is selected, a detail window of selected GO term shown in Fig. 2.1.1.5 will be opened. If "Select Same Terms" is selected, the nodes representing same GO term in the Tree View will be selected. If "Open AmiGO" is selected, AmiGO's web site (http://www.godatabase.org/cgi-bin/amigo/go.cgi) will be

opened on a new Internet Browser Window.

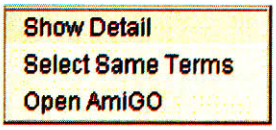


Fig. 2.1.1.4. a pop-up menu of Tree Views

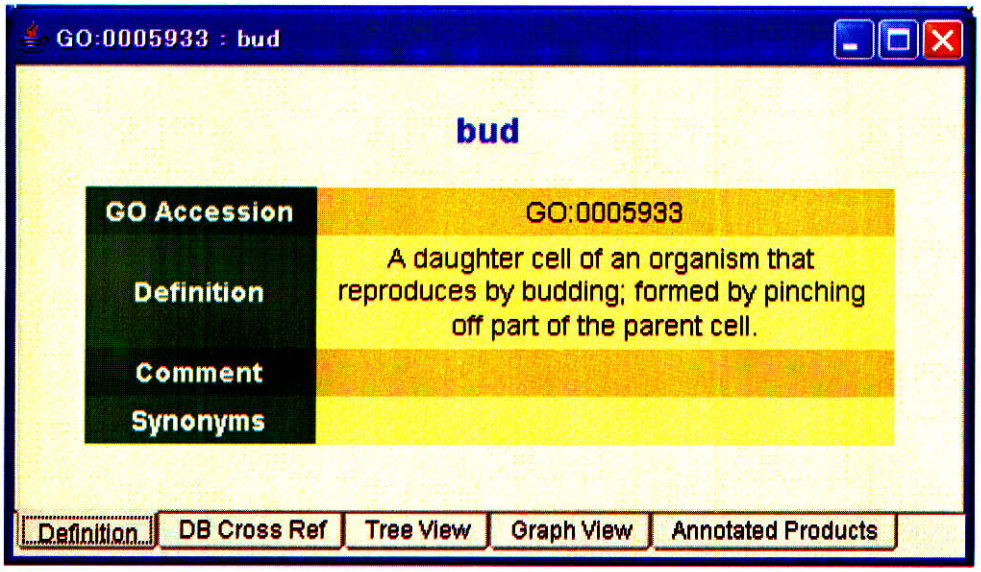


Fig. 2.1.1.5. a detail window of GO term "GO:0005933 : bud"

2.1.2 The Graph View

Graph View is a graph-formed representation of GO tree (shown in Fig. 2.1.2.1). In Graph Views, GO terms whose aspect is "molecular_function" are represented as blue ellipses, GO terms whose aspect is "biological_process" are represented as pink ellipses and GO terms whose aspect is "cellular_componens" are represented as green ellipses. "IS A" links are represented as khaki arrows, and "PART OF" links are represented as dark-blue arrows. Graph Views are seen in following windows: the Graph View window, detailed windows of GO terms and detailed windows of gene products. When the mouse pointer is over a GO term, detailed information of the term will be displayed as shown in Fig. 2.1.2.2.

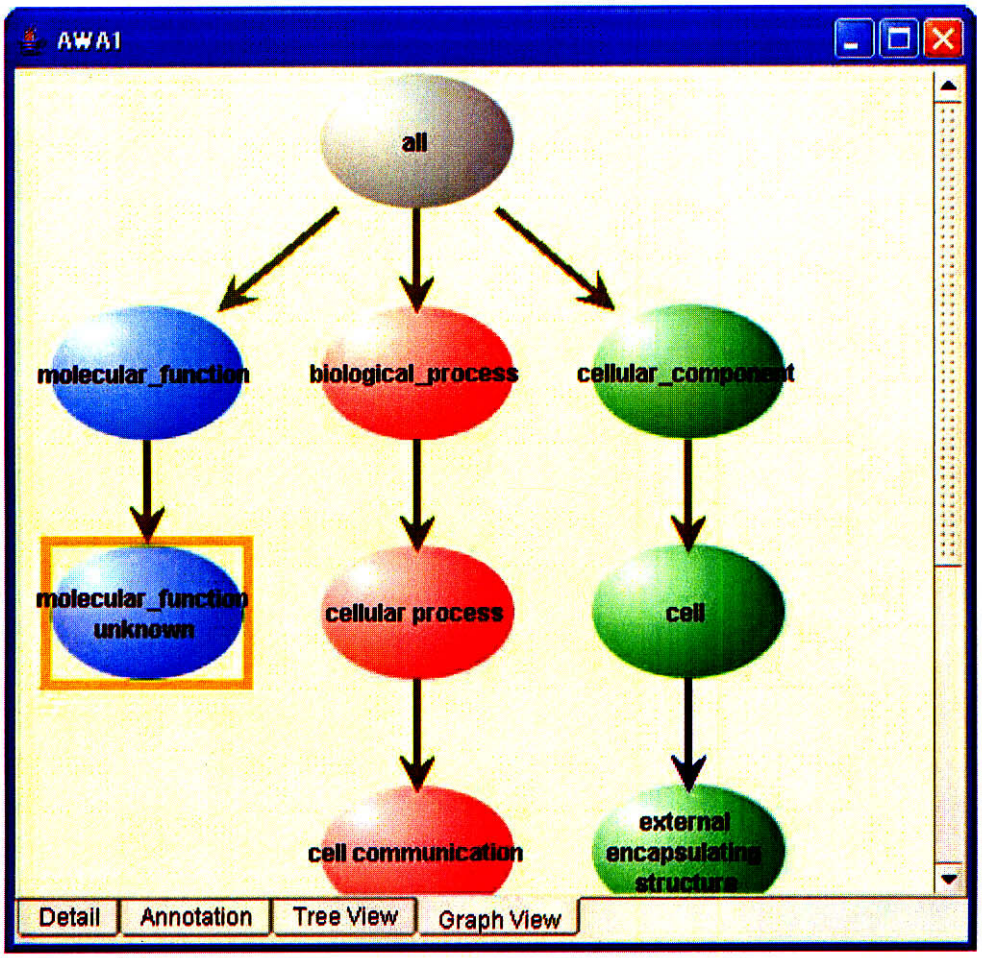


Fig. 2.1.2.1. A Graph View of gene products named "AWA1"

GO:0016032 : viral life cycle
A set of processes by which a virus reproduces. Usually, this is by infection of a host cell, replication of the viral genome, and assembly of progeny virus particles. In some cases the viral genetic material may integrate into the host genome and only subsequently, under particular circumstances, 'complete' its life cycle.
See also the biological process terms 'viral infectious cycle ; GO:0019058' and 'lysogeny ; GO:0030069'.
Synonym : viral infection, viral replication cycle

Fig. 2.1.2.2. Detailed information of "GO:0016032 : viral life cycle"

When a GO term on Graph View is right-clicked, a pop-up menu shown in Fig. 2.1.2.3 will be popped. If "Show Detail" is selected, a detail window of selected GO term shown in Fig. 2.1.2.1 will be opened. If "Open AmiGO" is selected, AmiGO's web site (http://www.godatabase.org/cgi-bin/amigo/go.cgi) will be opened on a new Internet Browser Window.

Fig. 2.1.2.3. A pop-up menu of Graph Views

When the right button of mouse is clicked on the background of Graph View, a pop-up menu like Fig. 2.1.2.4 will be popped. If "Zoom in" is selected, the view will be zoomed in. If "Zoom out" is selected, the view will be zoomed out.

Fig. 2.1.2.4. A pop-up menu of Graph Views

2.2 The ToolBar

There are 6 buttons on the ToolBar (shown in Fig. 2.2.1) upper the main window.

Fig. 2.2.1. The ToolBar of the main window.

2.2.1 Open Button 

By clicking this button, the dialog as shown in Fig. 2.2.1.1 in which you can select GO data file will be displayed.

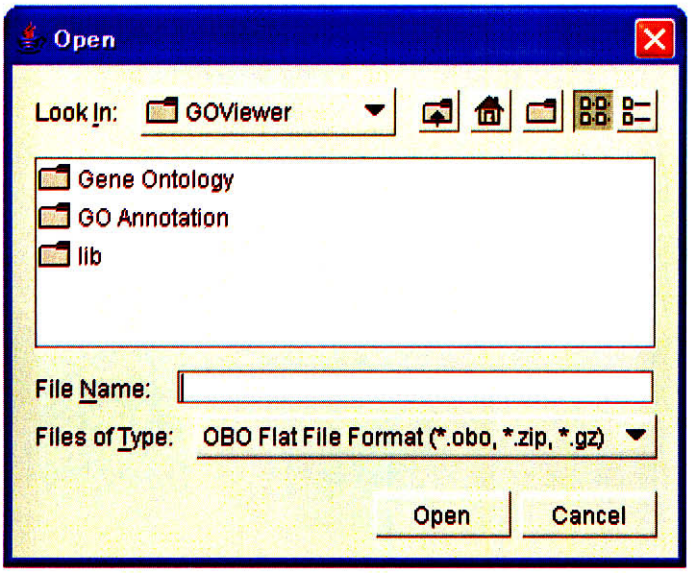


Fig. 2.2.1.1. The Open Window for GO data

At first, please select the format of GO data at "Files of Type:" box. Next please select the GO data file at the central panel of the dialog. When XML format is selected, DTD file is not required to specify. GOViewer ignores DTD file for both ease and simple. GOViewer can handle uncompressed files, ZIP compressed files and GZIP compressed files. When GO data file is specified, please click the "Open" button. The GO tree will be displayed on the original window as below. The URL of the selected GO data will be saved in the file "config.dat". When you start up GOViewer in the next time, GOViewer reads this selected file automatically.

2.2.2 Tree View Button

By clicking Tree View Button, a new window which shows Tree View of a selected term on the main window will be opened. This Tree View represents all the pathways for the selected term. This function is convenient in the case that you want to see Tree

View of GO terms continually.

2.2.3 Graph View Button

By clicking Graph View Button, a new window which shows Graph View of a selected term on the main window will be opened. This Graph View represents all the pathways for the selected term. This function is convenient in the case that you want to see Graph View of GO terms continually.

2.2.4 Search Button

By clicking Search Button, a new window shown in Fig. 2.2.4.1 for searching GO terms or gene products will be opened. You can search by two types of methods : substring match and exact match.

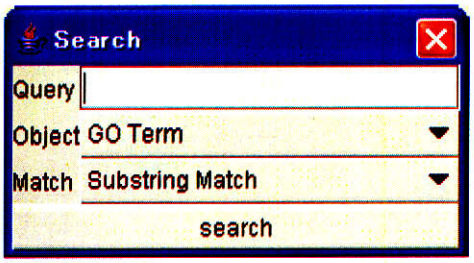


Fig. 2.2.4.1. Search Dialog

Query	Please Input query text
Object	Please select searched object : GO terms or gene products
Match	Please select searching method : substring match or exact match

2.2.5 View Option Button

By clicking View Option Button, a new window shown in Fig. 2.2.5.1 in which you can configure denotation of GO terms in Tree Views will be opened.

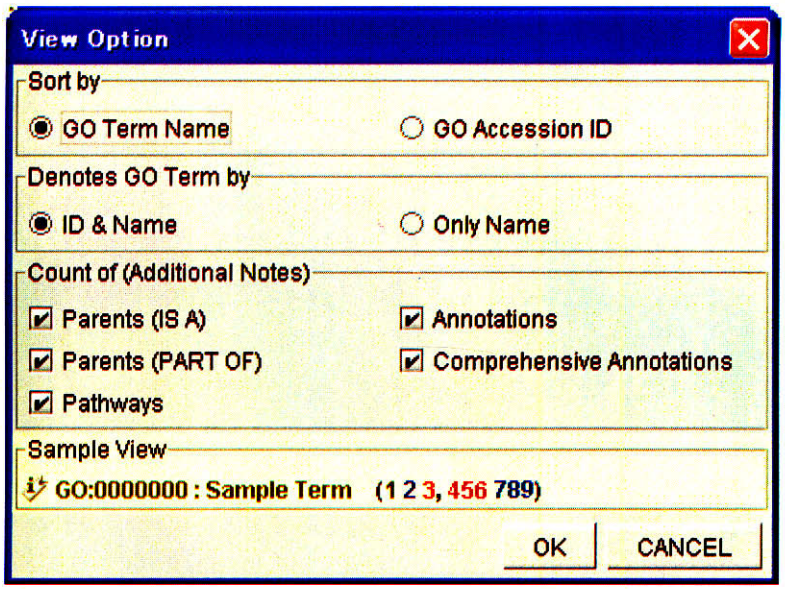


Fig. 2.2.5.1. View Option Dialog

Sort by	Please select the method for sorting GO terms	
	GO Term Name	sorting GO terms in the order of alphabet
	GO Accession ID	sorting GO terms in the order of GO Accession ID Number
Denotes GO Term	Please select the denotation of GO terms	
	ID & Name	GO Accession ID + GO term's name
	Only Name	only GO term's name
Count of (Addit.)	Please check the items you want to see	
	Parents (IS A)	the number of parents (IS A)
	Parents (PART OF)	the number of parents (PART OF)
	Pathways	the number of pathways
	Annotations	the number of annotations
	Compre. Annot.	the number of comprehensive annotations
	Sample View	A sample view of GO term based on the configured denotation

2.2.6 Filter Option Button

By clicking Filter Option Button, a new window in which you can configure annotation files, species and evidence codes will be shown. This configuration is called as "Filter". Gene products dealt with by GOViewer are added up and screened by this configuration. After configuring the filter, please click "OK" button, then both the configuration and annotation count of each GO term will be saved in the file "config.dat". When you start up GOViewer in the next time, GOViewer reads this configuration file and sets the same configuration.

2.2.6.1 Configuration of Annotation Files

By clicking the "Filtered Annotation Files" tab, the panel shown in Fig. 2.2.6.1.1 will be displayed. Users can choose the set of annotation files to use in GOViewer.

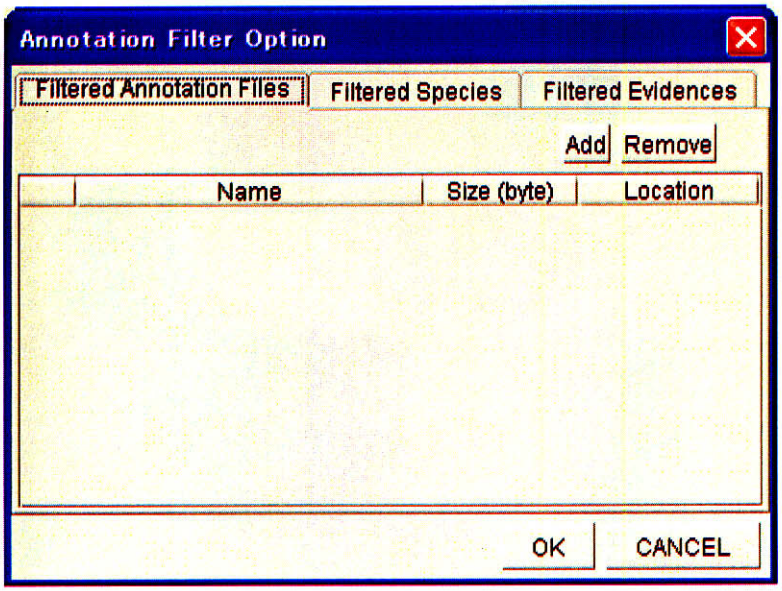


Fig. 2.2.6.1.1. "Filtered Annotation Files" tab of Annotation Filter Option Dialog

At first, please prepare annotation files, which can be downloaded at the following URL at this moment.
http://www.geneontology.org/GO.current.annotations.shtml

When annotation files are prepared, please click "Add" button. Then a file-open dialog shown in Fig. 2.2.6.1.2 will be opened.

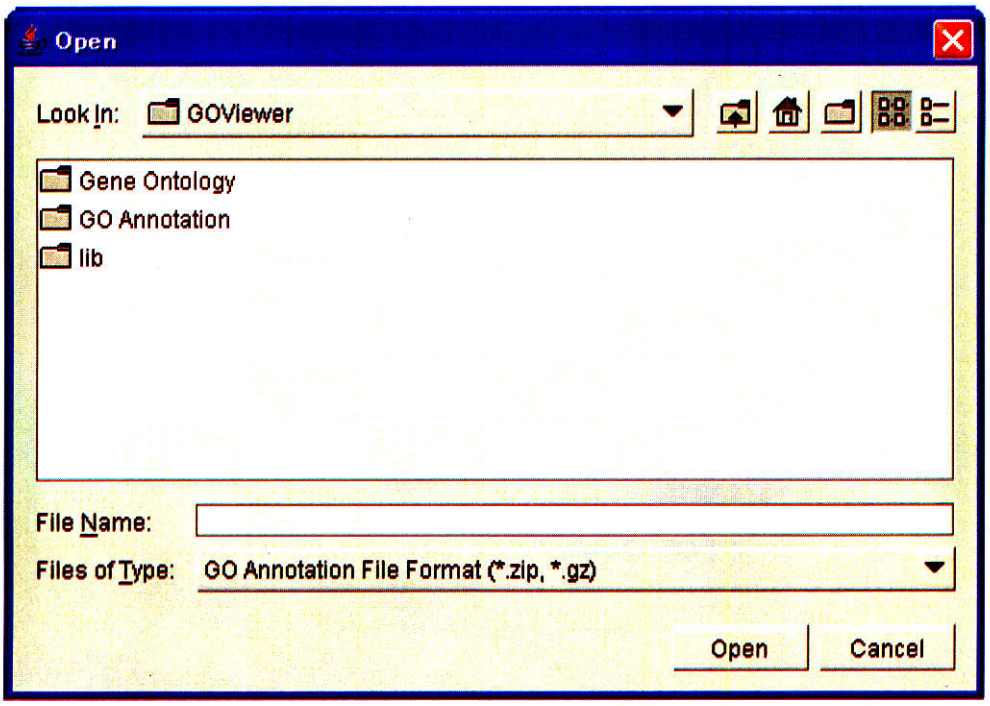


Fig. 2.2.6.1.2. A File Open Dialog for GO annotation files

GOViewer can handle ZIP compressed files and GZIP compressed files, not uncompressed files. You can select multiple numbers of files in this dialog. After selecting annotation files, please click "Open" button. Selected files will be added up into the table list of the original window as shown in Fig. 2.2.6.1.3.

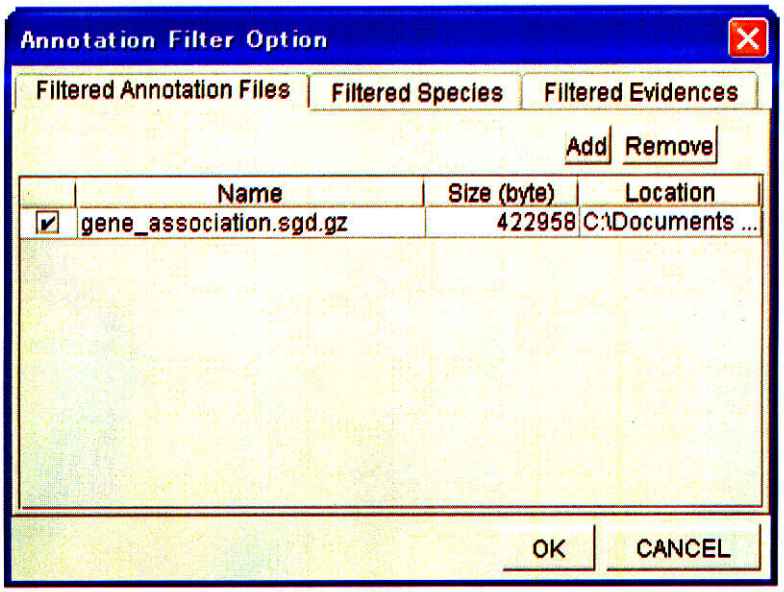


Fig. 2.2.6.1.3. The changed table-list

The leftmost checkbox in the table list in Fig. 2.2.6.1.3 means whether the annotation file will be used in GOViewer or not. When you want to remove annotation files from the table list, please select the annotation files in the table list and click the "Remove" button.

2.2.6.2 Configuration of Species

There are a lot of annotations belonging to various species in annotation files. In the panel shown in Fig. 2.2.6.2.1, you can choose the set of species to use in GOViewer if the "All Species" button is selected. Only the species checked in the table list are used in GOViewer. You can check any species in the table list. If there is no species users want to check, please click "Add" button, then a new window shown in Fig. 2.2.6.2.2 in which you can add a new specie will be opened.

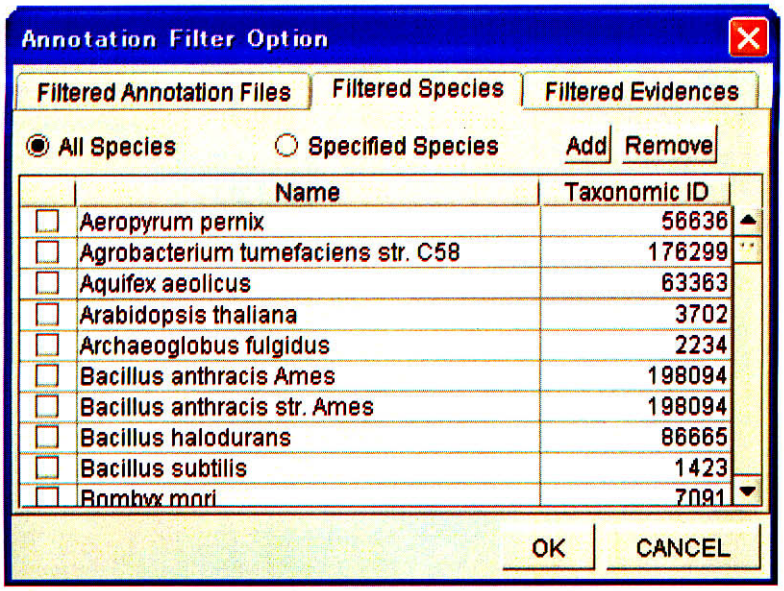


Fig. 2.2.6.2.1. "Filtered Species" tab of Annotation Filter Option Dialog

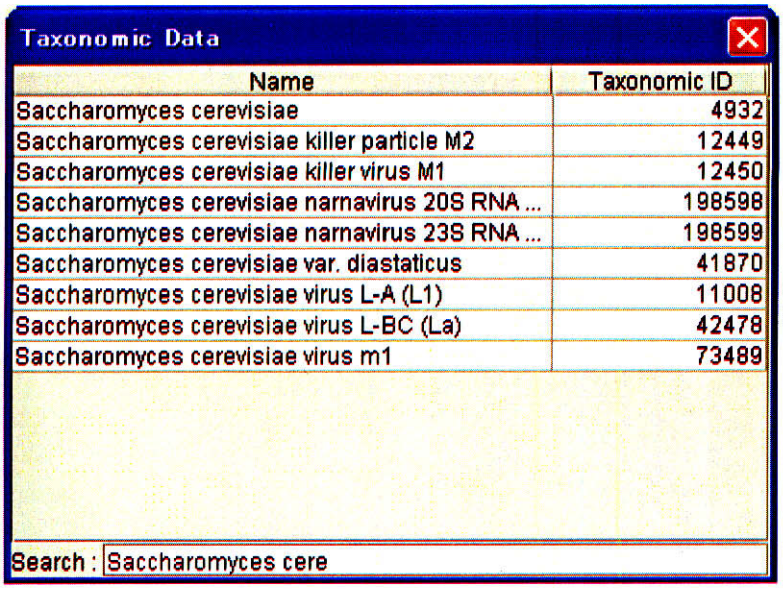


Fig. 2.2.6.2.2. The Selecting Taxonomic Data Dialog

At first, please enter the scientific name of the specie you want to add. As soon as you type its name, species whose names start with entered text will be shown. When specie you want to add is found, please double-click the specie, then the specie will be added in the table list in the original panel.

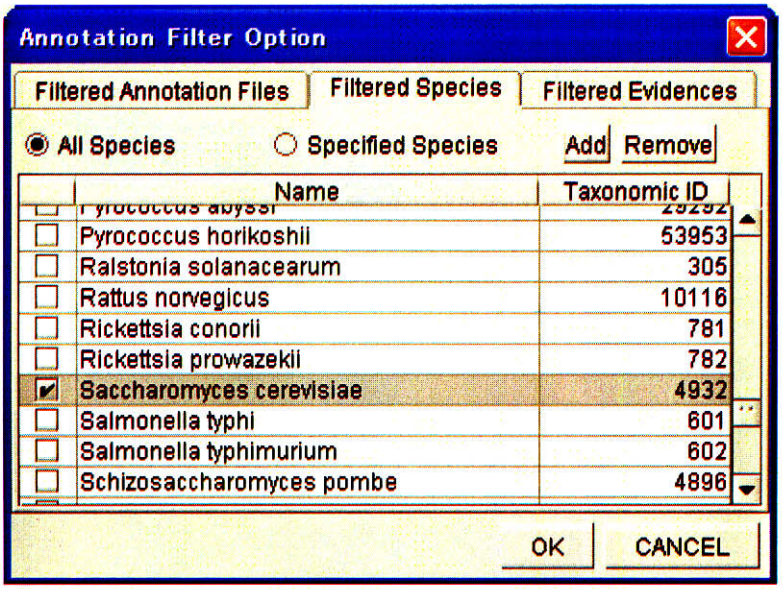


Fig. 2.2.6.2.3. The changed table-list

2.2.6.3 Configuration of Evidence Codes

Evidence Codes are the confidence tags meaning which types of experience a GO annotation was identified about a gene product. The detail of each evidence code is written in Gene Ontology Consortium's web site. You can choose the set of evidence codes to use in GOViewer in the tab shown in Fig. 2.2.6.3.1 if the "All Evidences" button is selected. Only the evidence codes checked in the table list are used in GOViewer. You can check any evidence codes in the table list.

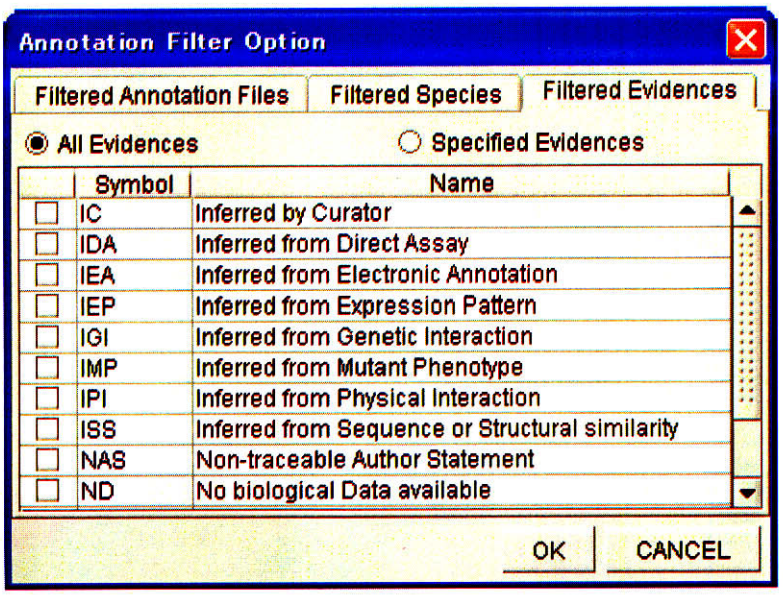


Fig. 2.2.6.3.1. "Filtered Evidences" tab of Annotation Filter Option Dialog

3 How to import GOViewer

GOViewer extends JPanel of the Swing Components of Java, so it can be used easily as a component.

```
java.lang.Object
  +--java.awt.Component
    +--java.awt.Container
      +--javax.swing.JComponent
        +--javax.swing.JPanel
          +--jp.ac.univ.jp.ac.univ.tokyo.aritalab.ontology.gene.viewer.GOViewer
```

Fig. 3.1. GOViewer's Extension Map

3.1 The libraries used in GOViewer

The libraries used in GOViewer are shown in the Table 3.1.1. These libraries are included in the package "GOViewer.jar". If you want to use each library individually, please download them at my homepage.

Table 3.1.1. The libraries used in GOViewer

go.jar	The package for handling Gene Ontology
graph.jar	The package for the data structure of Graph structure
jgraph.jar	This package is for rendering Graph structure. This package is named JGraph, and can be downloaded at http://www.jgraph.com/.
metouia.jar	The package containing a JFC's Look & Feel
ostermillerutils.jar	The package for opening the default internet browser of the PC's OS
xercesImpl.jar	The package in which a XML parser is implemented
xml-apis.jar	The package in which a set of APIs for handling XML is included

3.2 The classes in GOViewer

Fig 3.2.1 is a UML-like diagram of GOViewer. And Table 3.2.1 is Explanation of each class.

Table 3.2.1. Explanation of classes in GOViewer

Class Name	Explanation
GOViewer	The main class of GOViewer
TreeView	The class rendering a Tree View (is a JTree)
GraphView	The class rendering a Graph View (is a JGraph)
ConfigData	The class containing all the configuration
ViewOption	The class representing View Option
FilterOption	The class representing Filter Option
SearchDialog	The JDialog of searching GO terms and gene products
ResultTermFrame	The JFrame displaying searched result of GO terms by SearchDialog
ResultAnnotationFrame	The JFrame displaying searched result of gene products by SearchDialog
DetailFrame	The JFrame displaying the detail of a GO term
ProductFrame	The JFrame displaying the detail of a gene product
ViewOptionDialog	The JDialog of View Option
ViewOptionPanel	The JPanel of View Option
FilterOptionDialog	The JDialog of Filter Option
FilterOptionAnnotationPanel	The JPanel of filter of annotation files
FilterOptionSpeciesPanel	The JPanel of filter of Species
FilterOptionEvidencesPanel	The JPanel of filter of Evidence Codes

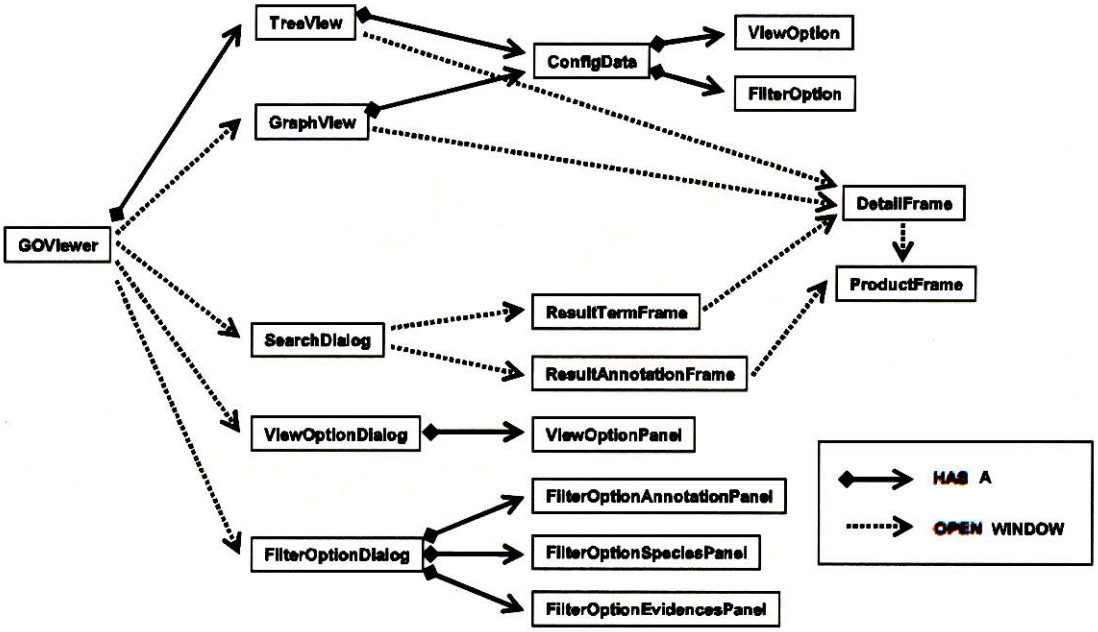


Fig 3.2.1. A UML-like diagram of GOViewer.

3.3 Size of required memory

When using GOViewer, please assign memory space of 128MB to GOViewer by using the Java VM arguments. This can be achieved by the following command:

java -Xms128m -Xmx128m -jar GOViewer.jar

If memory less than 128MB is assigned to GOViewer, processing speed of GOViewer may get slower. This is because GOViewer reads compressed files directly as extracting them simultaneously, so GOViewer requires sufficient space of memory to extract compressed files. 128MB is seemed to be sufficient size for GOViewer.

3.4 Sample codes

Code 3.4.1 is a sample code to use GOViewer as a Panel on a JFrame. Code 3.4.2 is a sample code to use GOViewer without JToolBar.

```
import java.awt.*;
import javax.swing.*;
import jp.ac.univ.tokyo.aritalab.ontology.gene.viewer.*;
```

```
public class Sample1 {

  public static void main(String[] args) {
    JFrame frame = new JFrame("GOViewer");
    frame.setSize(600, 500);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().setLayout(new BorderLayout());
    GOViewer viewer = new GOViewer();
    frame.getContentPane().add(viewer, BorderLayout.CENTER);
    frame.setVisible(true);
    viewer.loadConfig("config.dat");
  }
}
```

Code 3.4.1. A sample codes to use GOViewer as a Panel on a JFrame

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import jp.ac.univ.tokyo.aritalab.ontology.gene.util.*;
import jp.ac.univ.tokyo.aritalab.ontology.gene.viewer.*;
import jp.ac.univ.tokyo.aritalab.ontology.gene.viewer.preferences.*;
import jp.ac.univ.tokyo.aritalab.ontology.gene.viewer.serialize.*;

public class Sample2 {

  public static void main(String[] args) {

    String url_of_GOfile = "C:/GOViewer/Gene Ontology/200409/go_200409-termdb.xml.gz";
    String format_of_GOfile = "XML";        // "XML or OBO"

    // Creating View Option
    ViewOption view = new ViewOption();     // View Option
    view.setViewFlag(new boolean[]{true,    // Denotion of GOTerm
                                   true,    // Parents (IS A)
                                   true,    // Parents (PART OF)
                                   true,    // Pathways
                                   false,   // Annotations
                                   false}); // Comprehensive Annotations
    view.setComparator(new GOTermComparatorByName()); // Sort by Name

    // Creating Filter Option
    FilterOption filter = new FilterOption();
    // Creating Annotation Data
    Vector annotationData = new Vector();
    annotationData.add(new Vector(Arrays.asList(new Object[]{new Boolean(true),
                       "gene association.sgd.gz",
                       new Integer(422958),
                       "C:/GOViewer/GO Annotation/gene_association.sgd.gz"})));
    filter.setAnnotationData(annotationData);
    // Creating Species Data
    filter.setSpeciesAll(false);
    Vector speciesData = new Vector();
    speciesData.add(new Vector(Arrays.asList(new Object[]{new Boolean(true),
                       "Saccharomyces cerevisiae",
                       new Integer(4932)})));
    filter.setSpeciesData(speciesData);
    // Creating Evidence Codes Data
    filter.setEvidencesAll(false);
    Vector evidenceData = new Vector();
    evidenceData.add(new Vector(Arrays.asList(new Object[]{new Boolean(true), "IDA", ""})));
    filter.setEvidencesData(evidenceData);

    // Creating Annotation Counts Data (empty)
    Hashtable hashtable = new Hashtable();
```

```
    // Creating Configuration Object
    ConfigData config = new ConfigData(url_of_GOfile, format_of_GOfile, view, filter,
hashtable);

    // Creating GOViewer
    JFrame frame = new JFrame("GOViewer");
    frame.setSize(600, 500);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().setLayout(new BorderLayout());
    GOViewer viewer = new GOViewer();
    frame.getContentPane().add(viewer, BorderLayout.CENTER);
    frame.setVisible(true);

    // Set ToolBar Invisible
    viewer.getToolBar().setVisible(false);
  }
}
```

Code 3.4.2. A sample code to use GOViewer without JToolBar

When GOViewer is used in your program, you may often want to get out the selected GO terms in GOViewer. GOViewer has a TreeView as a JTree, so you can get JTree of GOViewer by the line 1 of Code 3.4.3. By using this JTree object, you can get selected GO terms by the line 2 of Code 3.4.3.

```
    JTree tree = viewer.getTreeView();
    TreePath[] paths = tree.getSelectionPaths();
```

Code 3.4.3. A code to get selected GO terms in GOViewer