JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 22, 19-29 (2006)

GOBU: Toward an Integration Interface for Biological Objects

WEN-DAR LIN, YUN-CHING CHEN, JAN-MING HO AND CHUNG-DER HSIAO*
Institute of Information Science
**Institute of Cellular and Organismic Biology*
Academia Sinica
Taipei, 115 Taiwan

In the present post-genomic era, many annotation catalogs are being created for annotating biological objects, for example, structural families, functional annotations, or expression levels in microarray experiments. Given a set of biological objects, one of the most important biological research goals is to find meaningful correspondences between annotations from different catalogs. For catalogs with linear structures, e.g., real numbers like expression levels, this problem can be solved by using computational methods. However, for catalogs with complex structures, e.g., functional annotations organized in a directed acyclic graph structure, this problem may be complicated for computer scientists, because these structures usually involve biological domain knowledge. Thus, it is necessary to have an interface that can help biologists discover important biological knowledge from these complex structures. In this paper, we propose the Gene Ontology Browsing Utility (GOBU), a Java-based software program, for integrating biological annotation catalogs under an extendable architecture that uses the Gene Ontology and a user-defined hierarchy as two main catalogs. GOBU is an OpenSource project hosted on OSSF and available at http://gobu.OpenFoundry.org/.

Keywords: gene ontology, annotation, data integration, bioinformatics, user interface

1. INTRODUCTION

In the present post-genomic era, many annotation catalogs are being created for annotating biological objects, and one of the most important goals in biological research is to find meaningful correspondences between different catalogs. For example, we may perform two different microarray experiments on the same genes, where each experiment assigns an expression level (a real number) to a gene, and thus each gene is assigned two expression levels. If we treat these genes as two-dimensional points, we may find some clustered genes of the same expression patterns. As another example, we may assign an organ (i.e., the brain, kidney, liver, etc) to a gene if this gene is working in this organ, and we may also assign functional annotations to genes, based on some public databases, like NCBI EntrezGene [3]. As a result, it may be possible to see the relationships between functions and organs, e.g., richer "signal transducer" genes in retina.

Generally speaking, we can treat an *annotation* as a property to be assigned to objects and treat a *catalog* as a set of annotations. From above examples, one can see that

Received June 15, 2005; accepted October 10, 2005.
Communicated by Kwei-Jay Lin.

some catalogs have good structures, like real numbers, so biological objects with annotations from these catalogs can be analyzed by using computational methods. However, some annotation catalogs are so complicated for computer scientists that investigations by biologists become necessary. For example, the Gene Ontology (GO) [4, 5] is a functional annotation catalog of biological terms organized in a directed acyclic graph structure. From the above, one can see that an integration interface that integrates these catalogs could help biologists obtain insights by manipulating these complex structures and obtaining immediate visual responses. Based on these insights, computer scientists could develop deeper computational tools for discovering important biological knowledge. Additionally, the integration interface should enable biologists to organize objects of interest into a hierarchical structure, based on some existing knowledge or personal preferences. For example, we could collect genes specific to organs and biologists might be interested in the functional differences between different organs.

The Gene Ontology, one of the most important functional annotation catalogs, was created and is maintained by the Gene Ontology Consortium as a controlled vocabulary for describing gene products. There are three sub-catalogs in GO, i.e., cellular components, biological processes, and molecular functions. All GO terms are organized in a directed acyclic graph structure with only one node with indegree zero, i.e., the *GO root*. Usually, GO is represented in a tree structure, in which every child GO term is a more detailed description of its parent, and the GO root is the tree root. Due to the use of this controlled vocabulary, the functional annotations between homologous genes across diverse phyla are consistent, and because of the increasing number of applications that annotated the functions of gene products with GO, we characterized GO as one of the main catalogs of our integration interface.

According to above reasoning, we have created the GO Browsing Utility (GOBU), which has the following features: (1) user-specified hierarchical data as input data, (2) user-defined data types for describing different annotations, and (3) an extendable software architecture for handling user-defined data types. In this paper, we present techniques and considerations involved in the GOBU design and one biological application.

The remainder of this paper is organized as follows. In section 2, we give an overview of the main GOBU program. In section 3, we describe techniques and considerations involved in the GOBU extendable architecture design. Then we present a biological application in section 4 and discuss future works in section 5.

2. OVERVIEW

The main GOBU program contains three tree components (see Fig. 1): the *GO tree*, *user tree*, and *focused GO tree* (from left to right). The GO tree displays GO terms in the tree structure as described in previous section, the user tree displays user-specified hierarchical data, and the focused GO tree displays a summary GO information of user selection.

To build hierarchical data for GOBU, users have to identify objects of interest, i.e., objects to be assigned annotations, as *representative nodes* (*R-nodes*) and then organize them into a hierarchical structure according to existing knowledge or personal preferences. Notice that a simple list of R-nodes is still a kind of hierarchical structure that

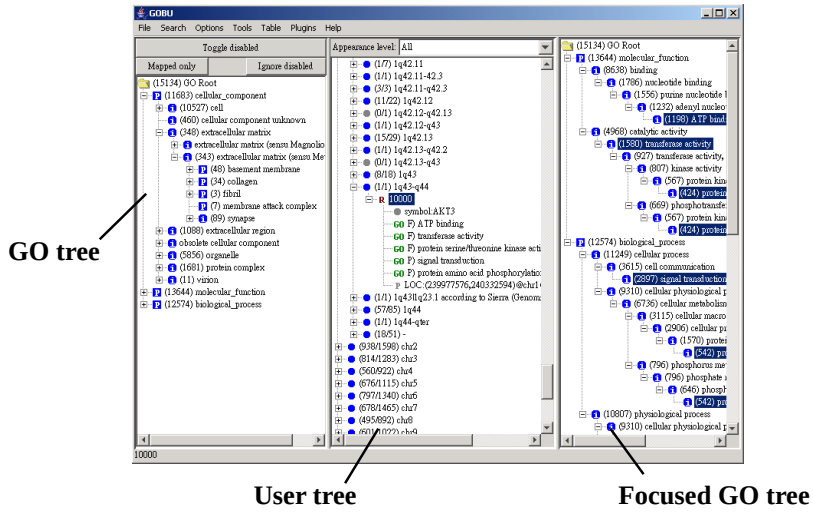


Fig. 1. The main GOBU program.

involves treating R-nodes as the child nodes of a common parent. After annotations are appended to R-nodes as their child nodes, the hierarchical data is ready for GOBU.

2.1 GO Distribution

The first thing GOBU does is to count the related R-nodes for every GO term, where we say that an R-node is *related* to a GO term if this R-node has a GO annotation that is exactly (or is a descendant of) the GO term. Recall that every child GO term is a more detailed description of its parent, so the number of related R-nodes of a given GO term is the number of R-nodes that can be described by this GO term. For convenience, we call the above counts in the GO tree the *GO distribution*. As a result, we append the GO distribution counts to GO terms, and biologists can interact with the GO tree to browse the GO distribution.

2.2 Selection Effect

When any GO term in the GO tree is selected, a *selection effect* will be applied to the user tree according to the following rules: (1) a *user node*, i.e., a node in the user tree, will be *selected* if it is a GO annotation and it is exactly (or is a descendant of) the selected GO term; (2) a user node will be selected if it has some child nodes that are selected. In GOBU, we assign the color gray to non-selected nodes and other colors to selected nodes. Additionally, the *Appearance level*, a helper control of the user tree, can help biologists do some filtering: (1) “All” means no filtering is performed; (2) “Unit” means non-selected *unit nodes* (nodes with ‘U’ icons) and their non-selected ancestors are filtered out; (3) “Representative” means non-selected R-nodes and their non-selected ancestors are filtered out. Notice that we do not filter out non-selected descendant nodes of non-filtered nodes, because biologists might be interested in studying them later (see Fig. 2 for an example).

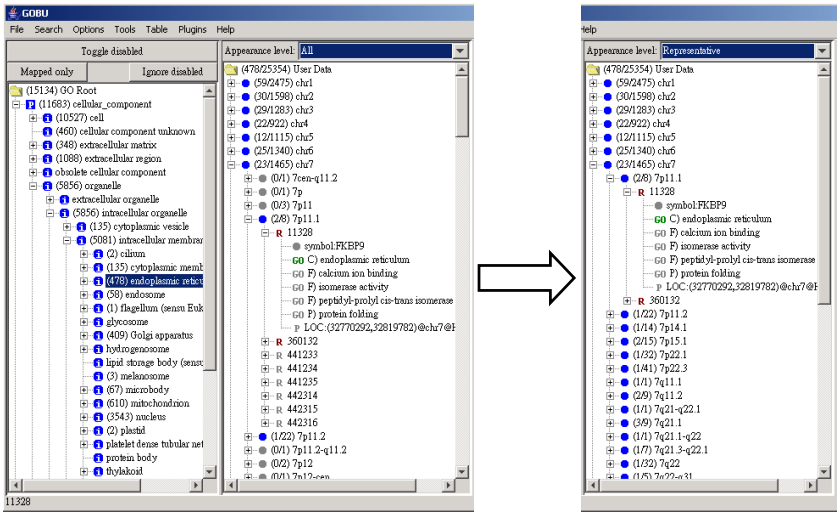


Fig. 2. An example of the selection effect and filtering.

2.3 Focused GO Tree

Recall that GO terms are organized in a directed acyclic graph structure, which means that some GO terms may have multiple appearances in the GO tree and that biologists might be interested in the parentage of a selected GO term. Thus we mimicked the DAG Viewer component of DAG-Edit (http://www.godatabase.org/dev/java/dagedit/docs/), an official tool of the Gene Ontology Consortium for editing the Gene Ontology, and created the focused GO tree to display the parentage of selected GO terms. Further, the parentage of different GO annotations in a sub-user-tree should also be shown to biologists. Thus the focused GO tree works according the following rules: (1) click on a GO term and the focused GO tree will show the parentage of this GO term; (2) click on a user node and the focused GO tree will be composed of the parentage of selected GO annotations shown under the clicked node (see Fig. 3). In either case, the selected GO terms (or annotations) will be marked “selected” in the focused GO tree.

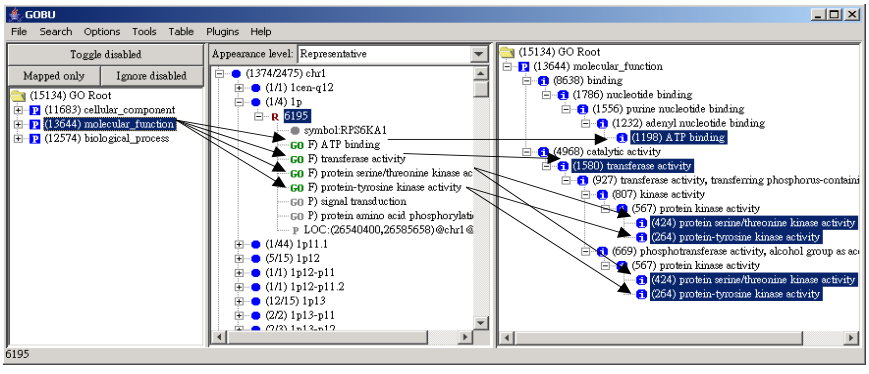


Fig. 3. An example of focused GO tree with clicking on a user node.

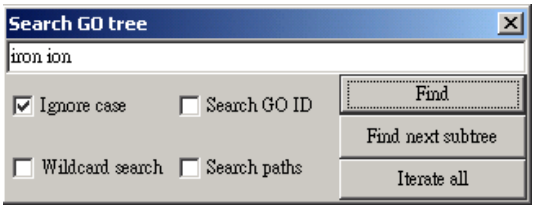


Fig. 4. Search utility.

2.4 Other Functionalities

Other functionalities include a table building utility for creating tables with GO distributions of pre-selected GO terms related to sub-user-trees and a search utility (Fig. 4) for searching nodes in a GO tree and user tree.

3. DATA FORMAT AND EXTENDABLE SOFTWARE ARCHITECTURE

3.1 Data Format

A data format for describing biological objects is necessary for an integration interface to enable biologists to manipulate biological objects with annotations from different catalogs. In this subsection, we describe the GOBU data format.

The most basic element of the GOBU data format is the technique used to describe user nodes. In GOBU, there are five classes of user nodes, some of which were described in previous section:

1. Representative nodes (R-nodes): These nodes are used to represent objects to be assigned annotations. These nodes are indicated by “R” icons.
2. GO annotations: These nodes are used to represent GO annotations of R-nodes. These nodes are indicated by “GO” icons.
3. *Property nodes* (*P-nodes*): These nodes are used to represent user-defined data types, i.e., annotations not from GO. These nodes are indicated by “P” icons.
4. *Unit nodes*: These nodes are used to represent the unit relationships of R-nodes. These nodes are indicated by “U” icons.
5. *normal nodes*: Other nodes that are indicated by dot icons.

In the GOBU data format, we use a single string to represent a user node; thus it is important to have a mechanism for encoding the type string and the content in a single string, and vice versa. Our solution is to use a colon (:) as a field separator of an input string; that is, the string “aa:bb” means type string “aa” and content “bb”. Additionally, for the inclusion of colons into type strings, we introduce the use of the escape character (anti-slash, \) into string processing, where characters exactly following escape characters are processed without any special meaning. Thus, we created a Java class called EscapeStringTokenizer, which produces “processed” type strings and “unprocessed” contents. For example, the string “GO:0004567” means type string “GO” and content

"0004567," and the string "G\O\::0\0\04567" means type string "GO:" and content "0\0\04567." The following table shows specifications for type strings and the contents of all classes of user nodes.

class	*type string*	*content*	*Example*
R-nodes	RP	any	RP:LL.1542
GO annotations	GO	integer	GO:0008372
P-nodes	PR	any	PR:LOC:(240773309,240816925)@chr1@Human
Unit nodes	UN	any	UN:group1
normal nodes	otherwise	any	name:CYMP

For P-nodes, the "unprocessed" contents are processed by EscapeStringTokenizer again to detect user-defined type strings and contents. For example, the string "PR:LOC:(240773309, 240816925)@chr1@Human" means a P-node of the user-defined data type "LOC".

To organize nodes into a hierarchical structure, we can use two file formats:

1. list format: every line indicates a tree path to be added to the user tree root, and every path is encoded by means of node strings with <TAB> as internal edges;
2. tree format: every line contains exactly one node string with *depth* leading space characters, and every node should be a child of the last node with *depth*–1 leading space characters.

```
UN:group6<TAB>tax9606<TAB>RP:33350932<TAB>conf: 100.0
UN:group6<TAB>tax9606<TAB>RP:33350932<TAB>GO:0005868
UN:group6<TAB>tax9606<TAB>RP:33350932<TAB>GO:0000166
UN:group6<TAB>tax9606<TAB>RP:33350932<TAB>GO:0005524
UN:group6<TAB>tax9606<TAB>RP:33350932<TAB>GO:0017111
UN:group6<TAB>tax9606<TAB>RP:33350932<TAB>GO:0042623
UN:group6<TAB>tax9606<TAB>RP:33350932<TAB>GO:0003777
UN:group6<TAB>tax9606<TAB>RP:33350932<TAB>GO:0007018
UN:group6<TAB>tax9606<TAB>RP:33350932<TAB>GO:0007052
UN:group6<TAB>tax9606<TAB>RP:33350932<TAB>PR:LOC:(100421045,100507307)@chr14@Human
```

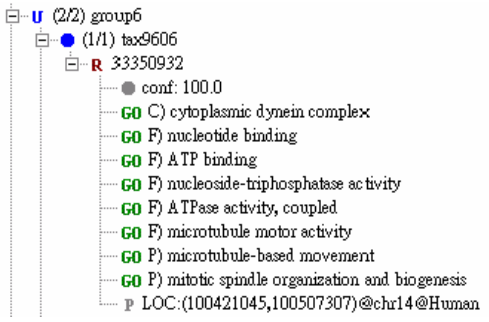


Fig. 5. An example of the list format data and its corresponding user tree.

Fig. 5 shows an example of the list format data. It should be noted that the list format is more redundant but easier to be generated by means of SQL queries. Thus, we provide several utilities for file processing:

1. TreeFileMaker: this translates list format files into tree format files;
2. TreeSorter: this rearranges tree format files and places them in an easy-to-browse order; that is, sibling tree nodes are sorted according to the words and numbers in their node names.
3. AddAnnotation: given a tree format file (possibly assigned some annotations), append more annotations to objects;
4. ExtractEntrezGene and ExtractEntrezGeneGO: these extract gene and GO information from NCBI EntrezGene files and build GOBU data files for specified species.

Fig. 6 illustrates the relationships among the above file processing utilities.

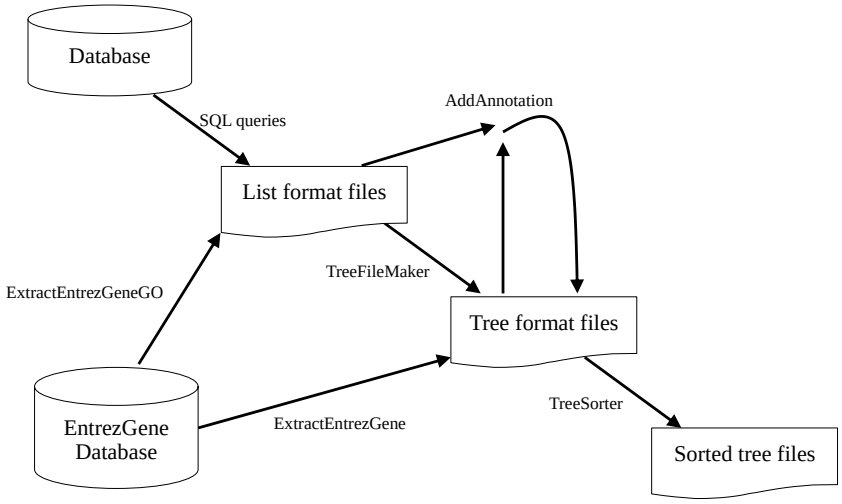


Fig. 6. The relationships between the file processing utilities and file formats.

3.2 Extendable Software Architecture

By assigning user-defined annotations to objects, we can create extension software (i.e., plugins) for handling these annotations. For example, we may define annotations that describe chromosomal localizations and create a plugin for drawing them, thus integrating one more annotation catalog (chromosomal localizations) into GOBU.

First, extension software classes must implement the Java interface GobuPlugin01 and must be placed in JAR files in the GOBU plugin subfolder. These classes will be loaded by an instance of PluginHolder when GOBU starts. While instances of these classes are obtained, additional plugin menu items (or pop-up menu items) are added for activating these plugins. At the same time, GOBU also registers their PropertyListeners

to an instance of PropertyListenerManager for proper events. Finally, there are two ways to activate plugins:

1. by selecting from a menu;
2. by clicking on tree nodes.

Note that the second approach provides a way for plugins to immediately respond to user actions.

4. CASE STUDY

In this section, we present an application of GOBU in a specific biological research project.

In this project, Zebrafish genes specific to tissues were selected through the use of statistical methods. The source tissues included the brain, fin, heart, inner ear, kidney, liver, olfactory rosettes, ovary, retina, and testis. Our cooperating biologists were interested in the following questions: (1) Is there any "hot spot" for these genes? (2) Is there any meaningful relationship between GO terms and these tissues?

To answer these questions, we first divided these genes into groups according to their organs and then assigned to these genes GO annotations by means of BLAST search [1] and chromosomal localizations by means of BLAT search [2]. In order to describe chromosomal localizations, we defined the "LOC" data type (in the form of (*start position*, *end position*) @ *chromosome* @ *genome*)). Finally, we created two extension software programs. The first one is called *Genome View*, and it is used to display the "LOC" data type in corresponding genomes. In Fig. 7, Genome View shows the chromosomal localizations of retina specific genes that are related to "signal transducer activity." Our second plugin is called *Specific Finding*, and it is used to compare the densities of related R-nodes (given a GO term) in different subsets. In Fig. 8, Specific Finding shows that the density of the retina is the highest of all tissues (given the GO term "signal transducer activity"). We need to discuss this result with our cooperating biologists.

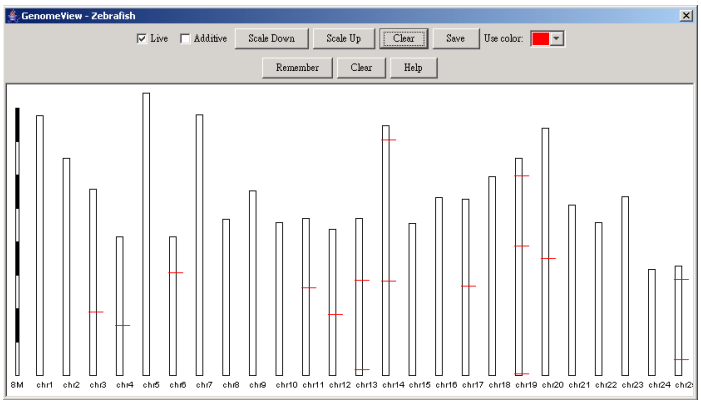


Fig. 7. Genome view plugin.

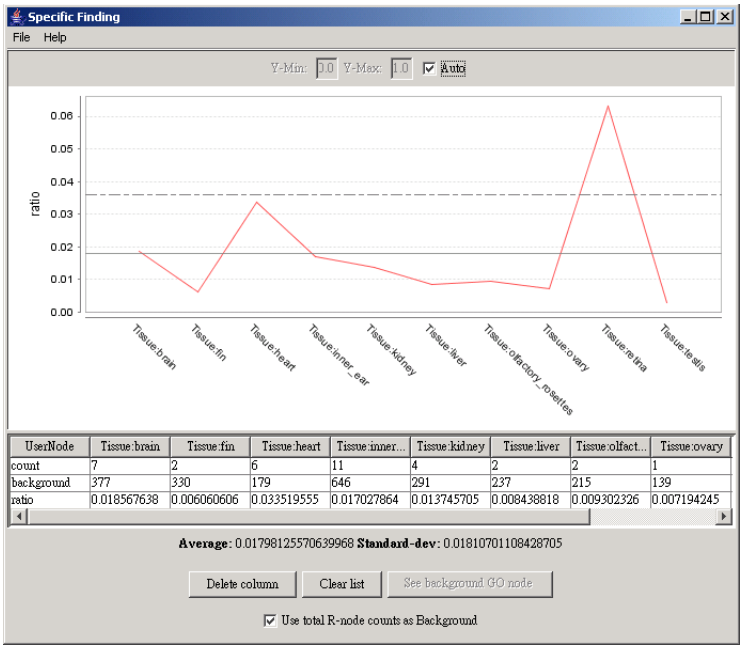


Fig. 8. Specific finding plugin.

5. FUTURE WORKS

Our next step will be to develop a plugin that (1) reads and displays annotations organized in a tree structure like GO and (2) computes counts like GO distributions for these annotations. Such a plugin could feasibly display annotations like map locations and other ontologies.

6. SYSTEM REQUIREMENTS

CPU: Pentium-4 (or equivalent); RAM: 512+ MB; Java Runtime Environment 1.4.2 (or later).

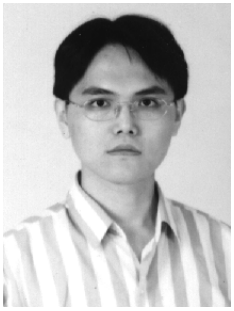
ACKNOWLEDGMENTS

The authors thank Profs. Ming-Jing Hwang and Wen-Chang Lin for helpful discussions.

REFERENCES

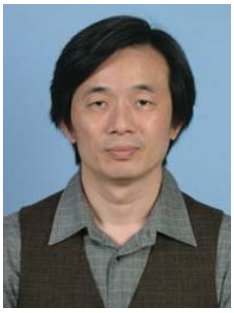
1. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, Vol. 215, 1990, pp. 403-410.

2. W. J. Kent, "BLAT – the BLAST-like alignment tool," *Genome Research* 12, 2002, pp. 656-664.
3. D. Maglott, J. Ostell, K. D. Pruitt, and T. Tatusova, "Entrez gene: gene-centered information at NCBI," *Nucleic Acids Research* 33, 2005, pp. D54-D58.
4. The Gene Ontology Consortium, "Gene ontology: tool for the unification of biology," *Nature Genet*, Vol. 25, 2000, pp. 25-29.
5. The Gene Ontology Consortium, "Creating the gene ontology resource: design and implementation," *Genome Research* 11, 2001, pp. 1425-1433.



Wen-Dar Lin (林文鏟) studied at the National Chiao Tung University (NCTU), from 1993 to 1998, where he received double degree from Dept. Applied Mathematics (AM) and Dept. Computer Science and Information Engineering (CSIE). In NCTU, he also received a M.S. degree CSIE in 2000 and a Ph.D. degree in AM in 2003. He then joined the Institute of Information Science (IIS), Academia Sinica as a postdoctoral fellow (till now). In IIS, he is a member of the Bioinformatics Group in the Computer Systems and Communication Lab. His research interests include discrete mathematics, algorithms, group testing and bioinformatics.

Yun-Ching Chen (陳昀慶) studied in the Department of Computer Science Information Engineering (CSIE) at National Chiao-Tung University (NCTU) from 1997 to 2001. In NCTU, he also received a M.S. degree CSIE in 2003. He then joined the Institute of Information Science (IIS), Academia Sinica as a research assistant (till now). In IIS, he is a member of the bioinformatics group in the Computer System and Communication Lab. His research interests include algorithms and bioinformatics.



Jan-Ming Ho (何建明) received his Ph.D. degree in Electrical Engineering and Computer Science from Northwestern University in 1989. He received his B.S. in Electrical Engineering from National Cheng Kung University in 1978 and his M.S. at Institute of Electronics of National Chiao Tung University in 1980. He joined Institute of Information Science, Academia Sinica, Taiwan, R.O.C. as a associate research fellow in 1989, and is promoted to research fellow in 1994. He visited IBM T. J. Watson Research Center in summer 1987 and 1988, Leonardo Fibonacci Institute for the Foundations of Computer Science,

Italy, in Summer 1992, and Dagstuhl-Seminar on “Combinatorial Methods for Integrated Circuit Design,” IBFI-Geschaftsstelle, Schloβ Dagstuhl, Fachbereich Informatik, Bau 36, Universitat des Saarlandes, Germany, in October 1993. He is a member of IEEE and ACM.

His research interests target at the integration of theoretical and application-oriented research, including mobile computing, environment for management and presentation of digital archive, management, retrieval, and classification of web documents, continuous video streaming and distribution, video conferencing, real-time operating systems with applications to continuous media systems, computational geometry, combinatorial optimization, VLSI design algorithms, and implementation and testing of VLSI algorithms on real designs.

He is Associate Editor of IEEE Transaction on Multimedia. He was Program Chair of Symposium on Real-Time Media Systems, Taipei, 1994–1998, General Co-Chair of International Symposium on Multi-Technology Information Processing, 1997 and will be General Co-Chair of IEEE RTAS 2001. He was also steering committee member of VLSI Design/CAD Symposium, and program committee member of several previous conferences including ICDCS 1999, and IEEE Workshop on Dependable and Real-Time E-Commerce Systems (DARE’98), etc. In domestic activities, he is Program Chair of Digital Archive Task Force Conference, the First Workshop on Digital Archive Technology, Steering Committee Member of the 12th VLSI Design/CAD Symposium and International Conference on Open Source 2001, and is also Program Committee Member of the 13th Workshop on Object-Oriented Technology and Applications, the 8th Workshop on Mobile Computing, 2001 Summer Institute on Bio-informatics, and Workshop on Information Society and Digital Divide.

Chung-Der Hsiao (蕭崇德) is a distinguished postdoctoral scholar of Institute of Cellular and Organismic Biology (ICOB) at Academia Sinica. His areas of interest are genome biology, developmental genetics and bioinformatics. He involved in a large- scale expressed sequence tag sequencing and annotation project on Zebrafish and Tilapia at ICOB. He received his M.S. and Ph.D. degrees in Fisheries Science from National Taiwan University.