

MULTIVARIATE HIGH DIMENSIONAL VISUALIZATION AND ANALYSIS OF MICROARRAY DATA INCORPORATING SIMULTANEOUS SPATIAL AND TEMPORAL COMPONENTS

2004/2005 Bioinformatics and Bioengineering Summer Institute
Academic Proposal

Vetria L. Byrd¹, Tarynn M. Witten² and Anthony Skjellum¹

¹Computer & Information Sciences, University Of Alabama at Birmingham

²Center for the Study of Biological Complexity, Virginia Commonwealth University

Abstract

There are several visualization tools available for scientists that allow for modeling, simulation and visualization of complex biological systems data. The functionality and features of these tools vary depending on the layer (cellular, molecular, *etc.*) of the system to be explored. My BBSI research effort will focus upon developing a different way of visualizing complex microarray datasets that have multiple variables of interest. We will use The *T. cruzi* parasite as the initial development data; however if implemented correctly, the resulting tool could be used to visualize datasets that contain more than 5-dimensions or variables of interest and may include time as well. This proposal will outline the approach we plan to take towards visualizing multivariate data.

Introduction

My research effort for the Bioinformatics and Bioengineering Summer Institute (BBSI) began with a concerted effort to find a research project with that would meet the following requirements: (1) the project would have a focus on scientific data visualization, (2) the project would be one that could be completed within the two year duration of the BBSI program and lastly, (3) the chosen project could be extended beyond the BBSI program and advance my thesis research in pursuit of my PhD in Computer Science. The ultimate approach will be one that considers a Systems Biology approach to system analysis and functionality.

Systems Biology is a field of Biology that looks at the entire system for evaluation, modeling and simulation. It is the study of complex, integrated biological systems. Unlike the traditional method of reductionist biology, where the sum of the parts is greater than the whole, the systems approach views the entire system as being greater than the sum of its parts. This approach is *integrative* because it focuses on the interaction or integration of parts in a functioning organism (Autumn 1995).

Ultimately, a systems approach will be taken; however to narrow the scope of the project my research effort will be to investigate different ways of representing complex data sets. Complex means datasets that contain more than 5-dimensions or variables of interest. The task is to develop new methods for visualizing such datasets along with developing new bio-user “friendly” interfaces. The design interface will allow users to import images and construct various data views and interactively manipulate the data. Users will also have the option to export images and numerical data contained in the images. Analysis modes of the visualization tool will incorporate analysis and statistical techniques from R Software, a freely available and widely used statistical language/software, as well as other graphics capabilities as specified by biologists.

Progress Report

The first phase of the research effort consisted of an extensive literature review of Systems Biology. A database was created to organize the resulting literature search. This database houses not only research articles but also information on visualization, simulation and modeling tools. The database will serve two purposes: (1) to allow for an organized method of recording the ongoing review of the literature and (2) to function as an online Systems Biology resource available via the Center for the Study of Biological Complexity Bioinformatics Computational Core Laboratories (BCCL) web site. While the database contains applications for modeling and simulation of biological networks, the database will be expanded to include microarray software. The goal is to review the software packages and where applicable, build on their functionality.

The interface design for the project is detailed below.

The interface design for the project will allow users to import 6 Affymetrix Gene Chips. Each image will be read onto the face of a cube (Fig. 1).



Once the images are read in, data views of the images will be constructed that will allow for interactive data manipulation. The software will include the ability to export the images and numerical data as well as incorporate other graphics and analysis modes. The *Trypanosoma Cruzi* (*T. cruzi*) microarray data will be used as initial development data. Microarray image data will be provided by the *T. cruzi* Biological Group at Virginia Commonwealth University. Users will be able to suppress or illuminate gene spots that are of a particular interest (*i.e.*, the user may be interested in viewing only those gene spots that have a illumination value at or above a predetermined value or selected by the user). The software will filter the cube and display only those gene spots that fit the user defined criteria and link the filtered data as shown in Figure 2. Additional viewing modes will be incorporated into the software to allow for the evaluation of longitudinal

(temporal) dynamics of the cells (Figure 3). Ultimately the stack will have depth “ n ”. This type of analysis and visualization of the data can be used to construct biological networks.

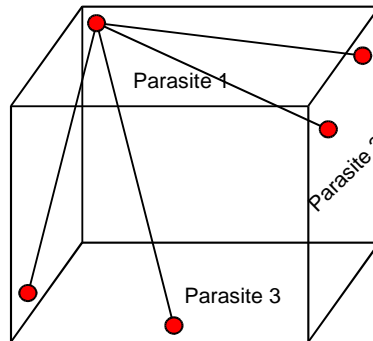


Figure 2: Filtered Cube

The image cube can be evaluated in the spatial domain by taking a cross sectional view of the cube at time i . A longitudinal view allows for evaluation in the time domain. Combining these viewing modes will allow for spatio-temporal integration of the data (Figure 4).

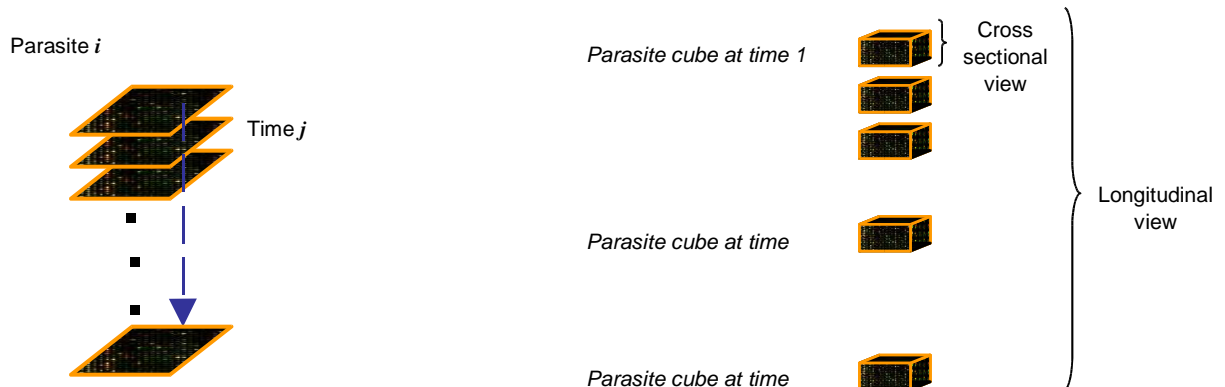


Figure 3: Temporal Stack

Figure 4: Spatio-Temporal Integration

Preliminary pseudo code for the algorithm is shown below:

```

Read in  $n$  images (cell types) in time,  $t$ 
  for each cell type 1 to  $n$ 
    for each cell time 1 to  $m$ 
      {perform actions on image  $I_{ij}$  }
    
```

The actions to be performed in this nested loop will include displaying the images, showing properties of the illumination array, allow users to cut into the sub image and manipulate it (rotate, zoom, select, etc.) as desired and automatically store the cube data in matrix format to access numerical gene markers. Additional features will be added as advised by bio-users.

Goals for Academic Year

Goals for the academic year include creating a bio-user “friendly” tool that will allow users to visualize multivariate complex data. The creation of a bio-user “friendly” tool requires continual interaction with biologist. Virginia Commonwealth faculty members, Dr. Kelly Archer (Biostatistics) and Dr. Patricio Mangué (Microbiology), will be consulted frequently to provide biological expertise. Once a prototype is developed it will be delivered to the biologist for alpha commentary.

Short term goals for the year include an ongoing review of the literature and visualization tools and development of the aforementioned interface design.

Long term goals for the year include (1) implementing the algorithm using thousands of images and incorporating parallel processing techniques as large computational demand arises, (2) write and submit the results of this research effort for publication and present this work at a national conference, and (3) shrink wrap version 1.0 of the software by August 2005.

Plan for Academic Year

Software Engineering methods and techniques will be implemented to ensure the resulting visualization tool is doable, reliable and generalizable to other scientific data visualization tasks. The most commonly used stages in the software development life cycle are problem definition, requirements analysis, specifications, design, coding, testing, and operation and maintenance (von Mayrhauser 1990). Table 1 details the software design overview for the research effort. A formal definition of each phase of the software life cycle is listed in the Appendix. Documentation found in the Appendix was taken from Section 1.3 Sequential Software Life-Cycle Models from the text *Software Engineering: Methods and Management* by von Mayrhauser.

There are several microarray tools that are currently available. Three well known and widely used microarray tools will be evaluated and reviewed for functionality: PathwayAssist™, GenePix® and Cytoscape. PathwayAssist™ is a powerful software tool for biological pathway analysis. For this research effort, PathwayAssist™ will be used to evaluate how the software handles microarray data interpretation. A 20 day trial version of the software is available via the World Wide Web. The purchase of a single user license for this software far exceeds the funding for this project, so the demo version will be downloaded and evaluated for the duration of the trial period. The demo can be downloaded from <http://www.ariadnegenomics.com/products/pathway.html>. GenePix® is standalone image analysis software for microarrays, tissue arrays and cell arrays. This software will be used as a front end to read gene chip images into the program. *Cytoscape* is an open source bioinformatics JAVA software platform for visualizing molecular interaction networks and integrating these interactions with gene expression profiles and other state data (Shannon *et al*, 2003). A closer inspection of the source code

for this project will serve as a starting block for initial programming structure and functionality.

	Technique/Method	Description	Milestone
(1)	Software Requirements Specification (SRS)	Customer writes, developer validates after iteration, accepted by both – Sign Off	Early schedule given
(2)	High Level Design (HLD)	Functional and non-functional aspects of design; flow chart; data structures. Design Review, Iterate, Sign Off	Better schedule guess
(3)	Write User's Manual		Often done last
(4)	Detailed Design (DD)	Maybe some early prototyping to refine understanding; more knowledge of detail; refine schedule; Critical; Design Review, Iterate, Sign off	Best schedule given
(5-N)	Implement with milestones		
(N+1)	Develop Test Plan		Customer should accept
(N+2)	Testing	Release candidates meeting DD + User's Manual	
	Repeat until acceptable		
(∞)		Infinity	Release

Table 1 Software Design Overview

Additional plans include:

1. Learning R Statistical Language Software. R is an open source implementation of the S Language. It is freely available for Windows and UNIX and can be downloaded from the Comprehensive R Archive Network (CRAN) via the World Wide Web (WWW). Free documentation for R is online. However, learning R requires a “formidable” investment of time.
2. Review current public domain microarray tools for functionality and use.
3. Ultimately, the project will take a systems approach by not only allowing the users to import image data but also allow the user to draw multiple networks of genes on a canvas and output the set of mathematical equations needed to solve the user drawn network. A review and renewed understanding of differential equations is needed. An introductory course on differential equations will be crucial for understanding and implementing this part of the project.

V. Byrd

2004/2005 Academic Proposal

Budget

Pending

References

- Autumn, K. 1995. Chapter 1, Section 2. Integrative Biology. Ph.D. Dissertation, University of California at Berkeley.
- Bader GD, Betel D, and Hogue CWV. 2003. BIND: the Biomolecular Interaction Network Database. *Nucleic Acids Research* 31(1):248-250.
- Bader GD, Betel D, and Hogue CWV. 2001. BIND: the Biomolecular Interaction Network Database. *Nucleic Acids Research* 29(1):242-245.
- Bader GD, and Hogue CWV. 2000. BIND—a data specification for storing and describing biomolecular interactions, molecular complexes and pathways. *Bioinformatics* 16(5):465-477.
- Kelley BP, Yuan B, Lewitter F, Sharan R, Stockwell BR, and Ideker T. 2004. PathBLAST: a tool for alignment of protein interaction networks. *Nucleic Acids Research* 32:W83-W88.
- Kelley BP, Sharan R, Karp RM, Sittler T, Root DE, Stockwell BR, and Ideker T. 2003. Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proceedings of the National Academy of Science* 100(20):11394-11399.
- Lee TI, Rinaldi NJ, Robert F, Odom DT, Bar-Joseph Z, Gerber GK, et al. 2002. Transcriptional Regulatory Networks in *Saccharomyces cerevisiae*. *Science* 298:799-804.
- Rain J-C, Selig L, Sellg L, De Reuse H, Battaglia V, et al. 2001. The protein-protein interaction map of *Helicobacter pylori*. *Nature* 409:211-215.
- Salwinski L, Miller CS, Smith AJ, Pettit FK, Bowie JU, and Eisenberg D. 2004. The Database of Interacting Proteins: 2004 update. *Nucleic Acids Research* 32:D449-D451.
- Shannon P, Markiel A, Ozier O, Baliga NS, Wang JT, Ramage D, Amin N, Schwikowski B, and Ideker T. 2003. Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks. *Genome Research* 13:2498-2504.
- Sharan R, Ideker T, Kelley BP, Shamir R, and Karp RM. 2004. Identification of Protein Complexes by Comparative Analysis of Yeast and Bacterial Protein Interaction Data. *Proceedings of the Eighth Annual International Conference on Research in Computational Molecular Biology—RECOMB*, San Diego, CA, pp. 282-289.
- Uetz P, Giot L, Cagney G, Mansfield TA, et al. 2000. A Comprehensive analysis of protein-protein interactions in *Saccharomyces cerevisiae*. *Nature* 403:623-627.

von Mayrhauser A. 1990. Software Engineering Methods and Management. Academic Press, Inc.

von Merling C, Krause R, Snel B, Cornell M, Oliver SG, Fields S, and Bork P. 2002. Comparative assessment of large-scale data sets of protein-protein interactions. Nature 417:399-403.

Appendix

A complete, detailed description of the Sequential Software Life-Cycle Models can be found in Chapter 1 (Section 1.3) of Software Engineering Methods and Management by A. von Mayrhauser).

1.3.1.1. Problem Design

Goal: Define the problem in user terms as precisely as possible

- The user is the ultimate arbiter of what the problem is and isn't
- The software engineer must determine how to best solve the problem abstractly, translate the solution back into its real life equivalent, and provide the user with a solution to the original problem
- Problem definition helps the user to better understand the problem

Results: A document that defines the problem in terms of the user's objectives and major constraints

1.3.1.2. Requirements Analysis

Goal: Determine the requirements and general feasibility of the solution

Objectives

- A major understanding should exist between the user and the developer of what a solution should provide
- There should be an agreement on the range of acceptability and possible tradeoffs, resulting in a set of acceptance criteria based on objectives and constraints
- A project plan should be made to implement the software with a schedule, budget and requirements as to what will be delivered at the end of each phase. If there will be several versions, a version plan is also needed. This plan indicates how and when versions will be developed and it defines their successive functionality and capabilities.

1.3.1.3. Specifications

The objectives of the specification phase are to describe what the solution looks like:

- what inputs the system is going to process,

- what functions it will perform for each input,
- what the corresponding outputs will be, and
- whether the specified system meets the requirements and whether its further development conforms to the project plan or whether the project plan must be altered

In general, specifications describe *what* the solution looks like – what the envisioned software system will do, the way it will accept and process data and provide corresponding output – but *not how* it will do all that.

This specification document often serves as a preliminary user manual.

1.3.1.4. Design

The design describes how the solution will be implemented. Design structures the system into its logically and functionally cohesive parts or modules. It also selects proper data structures and algorithms for the implementation of the input and output data and the system functions.

1.3.1.5. Coding

The objective of the coding phase is to translate the solution to actual code. The coding phase is complete when all code is written and documented, compiles error-free, and follows any coding rules and standards for the project.

1.3.1.6. Testing

The written code should be tested rigorously based on the required quality characteristics. Testing usually proceeds in several steps. As soon as code has been written, it should be tested.

Unit testing: first pieces (modules) are tested in isolation

Integration testing: modules are tested in groups to see whether they interact properly

System testing: in most instances, we must test the newly developed software system in its actual running environment.

Acceptance testing: ultimately, the software must meet the user's requirements. An acceptance test determines whether the user is convinced that the requirements are met.

Alpha testing: the software is tested in a controlled environment by some users with the developers present.

Beta testing: involves releasing a software product to some “friendly” users to find problems in a live environment; both alpha and beta testing prevent serious problems from going into the field

1.3.1.7. Operation and Maintenance

After software has been successfully installed, it must be kept operational. This is the object of the maintenance phase. Maintenance fixes bugs (*corrective maintenance*), adapts software to new environments and new and modified used (*adaptive maintenance*), and improves software (*perfective maintenance*).

Maintenance includes all activities that are necessary to keep production software operational. Maintenance handles problems and tracks them. Maintenance also ensures that multiple versions of a software product will not cause confusion and that changes are properly incorporated into software and associated documentation. This requires retesting all parts of the software that are affected by a change. This is called regression testing.

Maintenance determines when software has become obsolete or non-maintainable because of the costs of further changes are prohibitive when compared to the benefits gained from it.

Maintenance consumes a significant part of a software product’s live-cycle efforts.