Mihir Samdarshi

March 1, 2019

BIOL 588 – Advanced Systems Biology Research

**Refactor of code to the Model-View-Controller paradigm improves** **GRNsight: a web application for visualizing small- to medium-scale gene regulatory networks**

Discussion

*Addition of new features*

The new, highly centralized architecture of GRNsight allows developers to add new features in a highly intuitive and modular fashion. The way that the Model-View-Controller is implemented “has a profound impact on how easy the code is to develop, maintain and reuse” (Hansen & Fossum, 2005). Thus, the way that MVC was implemented in GRNsight was made such that harmful interactions were minimized, and that minimal code was situated in locations extraneous to the MVC architecture. Most code that faces the client-side is situated in three files, which constitute each component of MVC. Thus, adding new features simply requires the addition of the HTML into the static template files, a handler for the feature into setup-handlers.js, a new application state to grnstate.js, and code to execute the user’s input and update the view, located in update-app.js. Furthermore, the entire separation of Model, View, and Controllers enable a developer to create “pluggable views/controllers”; in other words, the complete separation of all components allows a user to change the functionality or appearance of a different component of a feature without the necessity to completely refactor all code associated with a feature due to its appearance completely in one function. However, in the same vein, some modularity can also be hindered by the strict MVC architecture. It is significantly harder to reuse view and controller code, because both are tied to the specific model. Thus, a developer must take extra and often superfluous steps to ensure full code compatibility if there are any changes to the model, or in order to ensure reusability of model or controller code (Veit & Hermann, 2003).

*Central Store for Application State*

The addition of a central state for GRNsight in the form of an object creates a cross-reference that is easily accessible by both the code itself, and developers (Hansen & Fossum, 2005). This central state, in the form of an object in GRNsight, stores most of the information regarding the application state, save for the position and physics of the actual nodes and edges displayed in the simulation. There are many advantages to having this central store of information.

Firstly, having a central state store creates much less of an opportunity for unintended state conflicts. Conflicts between application sub-states often create bugs that, especially in front-end code, may affect a user’s experience. According to Ceaparu et al. (2004), errors in an application are a leading cause for end-user frustration. A few bugs that were present under GRNsight’s previous architecture that were removed with the conversion of the architecture to MVC. Some of these bugs occurred when a user changed features like normalization factor of a graph’s edges while a graph was in a grid layout configuration. Under the previous architecture, the graph would revert back to a force graph layout. This was resolved after the transition to the MVC architecture, as the separate controllers for these features referenced different parameters within the central state store. However, when bugs do occur under the MVC architecture, having a central state store makes diagnosis of the bugs straightforward. After transition of the application to the new architecture, a bug arose where, even upon clicking on the “Lock Force Graph Parameters” input box, the Force Graph Parameter sliders would not be set. Diagnosing and creating a solution involved simply looking for all functions that relied on that parameter of the central state store, finding the conflict, and removing the conflict.

Finally, having a central state store allows ease of use for future development (Leff & Rayfield, 2001). In addition to allowing for minimization of potential state conflicts that could arise from new features, the central state store allows for easier developer access to code and for easier modularity. Under the previous architecture, blocks of code that might have controlled the same feature were interspersed throughout various files or even in different parts of the same file. Under MVC, this code is now guaranteed to be contained between three files, and functions are organized according to the specific feature that they may control or interact with. Functions that are shared between multiple of these other functions are located as close to the top of the file as possible. Furthermore, the transition of GRNsight to ECMAScript 2015 (ES6) allows for the importation of code from different locations, allowing code to be much more organized and increasing potential modularity. Components of the MVC architecture are separated into modules located in separate files because ES6 allows each file to nonetheless interact with each other. Webpack additionally allows for this modularity to continue to work across all platforms, including browsers that may not yet support the new ECMAScript featureset.

*New Features*

The three modifications to visualizations that were added allow users expanded functionality to modulate their graphs and enhance overall user experience and accessibility. These additional features allow users to set a threshold at which to visualize edges as gray or colored, view those gray edges as dashed lines, and to allow users to normalize edge weight visualization against a user inputted value.

Allowing users to set a threshold at which to visualize gray edges allows users the opportunity to display edges of importance. Users control this feature via a slider that corresponded to a percentage of the largest magnitude edge weight. By default, the visualization factor is set such that edges with an edge weight magnitude of less than 5% of the largest magnitude edge weight are visualized as gray. Movement of the slider increases the percentage by one percent increments, up to a maximum of a hundred percent and a minimum of one percent. The default of five percent was chosen as these relationships tend to be fairly insignificant in a small- or medium-scale gene regulatory network. However, this feature was added because allows users to be able to decide which edges are significant and share those with peers. Since edges are colored blue and red to correspond to activation or repression relationships, setting an edge weight visualization percentage allows viewers to pay attention to those colored edges rather than the thin, gray relationships.

The addition of the feature that allowed users to control the visualization of gray edge weights was motivated primarily because a member of the LMU Bioinformatics and Biomathematics research groups was color-blind. The color-blind member of the research group pointed out that it was difficult for him to distinguish between thin magenta/red edges and gray edges. Red-green color blindness affects approximately 8% of Caucasian males, 5% of Asian males, and 4% of African males worldwide (Birch, 2012); in order to make GRNsight available to a larger portion of the scientific community, solutions were considered to allow for more accessible viewing options. Two solutions were ultimately chosen; the first to change the edge weight coloring from cyan and magenta to blue and red, and the second to display gray edges as dotted lines, so that even without color users would be easily able to distinguish thin edges from gray edges (Okabe & Ito, 2002).

Finally, the last feature added in to improve GRNsight’s functionality was the option to set an edge weight normalization factor. In order to work within d3.js’s visualization limitations, edges are grouped into twelve “bins”, corresponding to edge thickness. The edge with the largest weight magnitude is placed in the being placed in the “bin” that corresponds to the largest possible edge thickness; edges are then assigned to subsequent bins based on what percentage of the largest weight their weight magnitudes are However, testing within the Dahlquist lab revealed that biologists are often trying to compare graphs of multiple gene regulatory networks in which the strongest magnitude relationship may not always be the same. Thus, the original programming of GRNsight did not facilitate accurate comparison of graphs. Setting an edge thickness normalization to a certain parameter allows users to compare graphs with differing maximum edge weight magnitudes. Thus, research that relies on the comparison of different gene regulatory networks can be easily shared.

*Future Directions*

With GRNsight now a much more modular, developer-friendly, and user-friendly software following the refactor to MVC and addition of new features, there are many possible directions that the GRNsight team could take with our software. The recent addition of the Cytoscape.js library (Franz et al., 2015) allows the potential addition of useful graph statistics. These statistics include global clustering coefficient, betweenness centrality, and degree distribution. Statistics like these allow scientists to visualize important or insignificant relationships between genes (nodes) and to determine important properties regarding significance of certain nodes or graphs. Systems biology analysis tools such as InnateDB (Breuer et al., 2012) and ConsensusPathDB are two examples of software that have already successfully implemented the functions that enable graph statistics in Cytoscape.js. Additionally, the transition of test architecture to better follow the new web-client architecture would allow GRNsight to retain a more intuitive testing process and would help to better attain bug-free software. This transition would also go hand in hand with the addition of tests specific to the new architecture in order to help GRNsight return to the Test-driven development (Beck, 2003) philosophy, which it has strayed from in recent history. Finally, in order to better increase GRNsight’s share-ability, a feature could be added to allow export of a graph visualization to an easily shareable format such as Portable Network Graphics (PNG) or Scalable Vector Graphics (SVG).

*Conclusion*

GRNsight has undergone many changes since its inception in 2014; however, it has retained the same architecture for the majority of its lifetime. During the course of this project, a major effort was undergone in order to convert GRNsight to a more sustainable and prudent architecture, termed Model-View-Controller. This architecture allows GRNsight to contain more modularity, better developer access, and less bugs. Furthermore, the addition of new features that allow users to better modulate the network graphs created by GRNsight allow users better options and more flexibility. Finally, preparation for a move to the ReactJS framework allows GRNsight to be more viable in the long-term.

References

Beck, K. (2003). *Test-driven development: by example*. Addison-Wesley Professional.

Birch, J. (2012). Worldwide prevalence of red-green color deficiency. *Journal of the Optical Society of America A*, 29(3), 313-320. DOI: 10.1364/JOSAA.29.000313

Breuer, K., Foroushani, A. K., Laird, M. R., Chen, C., Sribnaia, A., Lo, R., Winsor, G. L., Hancock, R. E. W., Brinkman, F. S. L., Lynn, D. J.; InnateDB: systems biology of innate immunity and beyond—recent updates and continuing curation, *Nucleic Acids Research*, Volume 41, Issue D1, 1 January 2013, Pages D1228–D1233. DOI: 10.1093/nar/gks1147

Ceaparu, I., Lazar, J., Bessiere, K., Robinson, J., & Shneiderman, B. (2004). Determining causes and severity of end-user frustration. *International Journal of Human-Computer Interaction*, *17*(3), 333-356. DOI: 10.1207/s15327590ijhc1703\_3

Franz, M., Lopes, C. T., Huck, G., Dong, Y., Sumer, O., & Bader, G. D. (2015). Cytoscape. js: a graph theory library for visualisation and analysis. *Bioinformatics*, 32(2), 309-311. DOI: 10.1093/bioinformatics/btv557

Hansen, S., & Fossum, T. V. (2005). Refactoring model-view-controller. *Journal of Computing Sciences in Colleges*, *21*(1), 120-129.

Kamburov, A., Stelzl, U., Lehrach, H., & Herwig, R. (2012). The ConsensusPathDB interaction database: 2013 update. *Nucleic Acids Research*, 41(D1), D793-D800. DOI: 10.1093/nar/gks1055

Leff, A., & Rayfield, J. T. (2001). Web-application development using the model/view/controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International* (pp. 118-127). IEEE. DOI: 10.1109/EDOC.2001.950428

Okabe, M., & Ito, K. (2002). How to make figures and presentations that are friendly to color blind people. University of Tokyo.

Veit, M., & Herrmann, S. (2003, March). Model-view-controller and object teams: A perfect match of paradigms. In *Proceedings of the 2nd international conference on Aspect-oriented software development* (pp. 140-149). ACM. DOI: 10.1145/643603.643618