Mihir Samdarshi

February 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**

Outline

1. Introduction
   1. Review what a gene regulatory network is
   2. What the Dahlquist lab is doing with yeast
      1. Cold shock experiments
   3. GRNmap
      1. Modeling of GRNs
      2. Sister project
      3. Parameter estimation
         1. Produces weighted and unweighted adjacency matrix (Dahlquist et al., 2015)
   4. What are the features of a successful visualization tool? (Pavlopoulos et al., 2015 & Saraiya et al., 2005)
      1. Automatic construction of pathways
      2. overlaying information on pathways
      3. link pathways with other data
      4. compare multiple pathways
      5. scale pathways to enable higher biological systems-level views of the pathways
   5. What does GRNsight currently do
      1. Older features (Dahlquist et al., 2016)
      2. Newer features:
         1. Edge weights
            1. Subject to physics
            2. Can be normalized against other edges
            3. Can set gray edge threshold
         2. Node coloring – for timecourse data sets
         3. Grid layout
         4. Can import and export adjacency matrices to various formats
         5. Open source
   6. Why does GRNsight do what we do
      1. Visualization of data for accessibility purposes
      2. Competeing projects
         1. Cytoscape not open source, meant for large data sets
         2. Gephi, large data sets, not biologically targeted
   7. Software architecture
      1. Test-driven development central to the core
      2. Currently haphazard
      3. Built over many years
   8. Need for a central state register to minimize conflicts between various new features
      1. Model-View-Controller is what this system archetype is called (Lee & Rayfield, 2001)
   9. ReactJS, framework developed by Facebook to deliver websites modularly
      1. Dependent on MVC paradigm
2. Materials & Methods/Results
   1. Codebase
      1. Using Webpack to bundle assets
      2. ES6 allows for more concise code and modularity in its own way
   2. Previous Architecture
      1. Code was separated by feature
      2. All three aspects of the application were contained in singular files
         1. What the user sees
         2. How the user can modulate it
         3. What the state of a feature is
         4. For example, node-coloring.js handled everything about node-coloring after the upload of the adjacency matrix
            1. Receive changes, such as compare new data set
            2. Respond to changes, such as clicking reset node coloring
            3. Display changes, including displaying new options
   3. Current Architecture
      1. Setup-handlers receives changes from user and updates central store of app state (Controller)
      2. Update-app receives changes from central store and updates view (View)
      3. Central store is an object called GRNstate (Model)
      4. Movement order
         1. Grey edges
         2. Normalization
         3. Weight options
         4. Sliders
         5. Graph Layout
         6. Node coloring
   4. ReactJS
      1. Planning move to ReactJS, although not done yet
      2. Both d3.js and ReactJS control the Document Object Model
      3. Need to find a way to minimize React and d3 clashes (in progress)
      4. ReactJS can control external parts of the website and make the website extremely modular
         1. Futureproofing for the addition of new features
   5. Other
      1. Grey edge threshold
         1. Important to be able to focus on only nodes higher than a certain weight
      2. Normalization
         1. Allows weights to be compared relative to some other weight
3. Discussion
   1. Allows for the addition of new features in a highly intuitive and modular fashion
   2. Central store for the state of the entire application creates an easy cross-reference (Hansen & Fossum, 2005)
      1. Creates less opportunity for potential state combinations to conflict, thereby creating bugs
      2. Allows for easy diagnosis and treatment of any potential bugs
      3. Allows ease of use for future development (Leff & Rayfield, 2001)
         1. No need to dig through code, very limited number of files rendering the graph
         2. ES6, Webpack and React make it simple to add new features via modularity
   3. These changes also allow front-end users a far more seamless experience than ever before
   4. Other features that I have added allow front end users more opportunities to better gain information from and share their data
   5. Future directions
      1. Finally add graph statistics into GRNsight
      2. See what bugs are remaining following the move to the MVC architecture
      3. Export graph to easily shareable format like PNG or SVG
4. Figures
   1. Display what a Model-View-Controller architecture looks like
   2. Initial vs Final files in public/js/
      1. Screenshot Github from earlier release version
   3. Annotated view of what the GRNsight app looks like
   4. ReactJS modularity example
      1. Tania Rascia’s example of what React code would look like, versus what the DOM would look like
   5. A network graph with the grey edge threshold set to 75%, 50%, and the default value
   6. A network with the normalization set artificially low, a graph with the normalization set artificially high, and a graph with no user-set normalization parameter