Anindita Varshneya

BIOL 598-04: Advanced Systems Biology Research

**Test-driven development and new features improve GRNsight: a web application and service for visualizing small- to medium-scale gene regulatory networks**

Discussion

Error and warning catching, TDD, and the other user interface features mentioned above are necessary and indispensable additions to the GRNsight project. In addition to building GRNsight according to software development best practices, these features ensure that visualizations presented by GRNsight are true to the biological data uploaded by the user, and ensure that our program handles incorrectly formatted adjacency matrices without crashing. Furthermore, the user interface and customization features allow users to make modifications to the visualization that are both informative and aesthetically pleasing.

GRNsight returns informative error and warning messages

Prior to the additions that I made to the GRNsight project, with help from the entire research team, our spreadsheet parser only considered a few major edge cases, and did not have any warnings. Supplementing the previously created error list with additional error edge cases means that substantially more mistakes in adjacency matrices can be caught. Handling of these mistakes improves the integrity of the project overall, as the entire program is less likely to crash in the event that a user attempts to upload an incorrectly formatted adjacency matrix.

While error handling was meant to protect our program from crashing, the implementation of warnings ensured that the data we present to the user properly and accurately represents the biological information that the user uploaded. If a potential mistake was found in the adjacency matrix that did not cause the program to crash, we implemented a warning catch that returns a message to the user that informed them of the potential mistake. This ensures that the user knows the graph visualization may have problems, as well as what they should fix in order to completely prevent those problems from existing.

Both the error and warning handling systems are essential to communicating to the user when GRNsight does not function exactly as it is supposed to. However, these systems are only as useful as the messages they provide to the user. If the user doesn’t understand why GRNsight didn’t upload their adjacency matrix as they expected, the importance of the error and warning handling systems are diminished.

In order to convey this information to the user in the most optimal way, we used a three-part message system that tells the user: a) there was a problem when GRNsight tried to upload their adjacency matrix, b) where the problem was, and c) how the user can attempt to fix it. The purpose of the three-part error/warning message framework is to ensure that all error messages are precise and constructive, so users understand exactly where the problem is and how they can fix it. Furthermore, all messages were written so that lay-users could understand the entirety of the message. These characteristics are essential for a positive human-computer interaction as they ensure the user will be able to benefit from the message (Molich et al., 1990). Moving forward in this aspect of the project, it would be beneficial to explicitly explain to the users how the potential problems in their adjacency matrix might affect their graph visualization. For example, if a user uploads an adjacency matrix with a missing target gene name, it would be beneficial to inform the user that any regulatory relationships that influence the gene that belongs in the missing cell will not be presented in the visualization. This would not only make our error messages more comprehensive, but it will present all information in an easy to understand format for lay-users. By providing this additional information, even lay users can understand the implications of the warnings associated with their adjacency matrix.

Test-driven development (TDD) allows for quality checks in GRNsight

The use of TDD consistently, and through all server-side functionality in GRNsight ensures that all of the errors and warnings mentioned earlier are caught and treated as expected. Without properly functioning error and warning handlers, the informative messages that go with them are of no use. Furthermore, TDD ensures that all functionality internal to the program is functioning as expected. Implementation of TDD to projects such as GRNsight is essential in ensuring the quality of the program and the corresponding visualizations.

It was found in analysis of 27 different programs, made either in industry by professionals or in academia by students, that TDD improved both internal and external quality of the program (Rafique et al., 2013). Across a majority of these projects, the use of TDD also indicated a decrease in productivity. However, only one in nine academic projects reported decreased productivity upon implementation of TDD (Rafique et al., 2013). GRNsight is still in the process of optimizing the unit testing framework, so it is difficult to say whether implementation of TDD has truly slowed down the productivity of the programmers. That said, because GRNsight is mainly used for analysis of biological data, the importance of accuracy in our visualizations outweighs the importance of quick implementation of functionality. Whether or not TDD slows down productivity in the production of new functionality, it is essential to the program that all code functions as expected, and is checked for proper functioning immediately as it is being written. Test last development (TLD) offers too many opportunities for unit tests to conform to the preexisting code, and therefore does not ensure the same level of quality that TDD requires. While implementation of TDD has been a long process, it has been essential for the integrity of the visualizations produced by GRNsight, and conforms GRNsight to software engineering best practices.

Additional features and modifications in graph visualizations allow for customization

The two modifications affecting graph visualizations that I added to GRNsight are centering of arrowheads to edges, and menu options that allow users to hide or show weights. Centering arrowheads to the edges was a purely aesthetic decision. Arrowheads were centered to each edge weight separately because using the same centering parameter across all edge weights resulted in off-center arrowheads at thicker edge weights. This adjustment was an important addition to the GRNsight graph visualization because one of the key aspects of GRNsight’s design is that graphs are quick and simple to produce, and are visually appealing.

Additionally, a new feature was added that allows users to hide and show weights in weighted graphs using menu options in the sidebar and in the format menu of the program. By default, weights are hidden until mouse over, at which point they appear in a tooltip at the bottom right corner of the pointer. Users can select to either always show weights, never show weights, or return to default settings and only show weights upon mouse over of the edges. When presenting and analyzing networks, it is often beneficial to see both graphic displays of the regulatory relationships (i.e. via color coded edges with varying thicknesses) as well as numerical weights. Allowing users to always show weights means they can easily interpret their graphs without having to mouse over each individual edge. Additionally, hiding all weights can be beneficial if the user chooses to not have extra data on their network.

Another important reason for users to be able to always show weights is because it allows users to compare networks fairly. If two networks are generated using GRNsight, in order to compare the graphs with each other, the user must be able to look at the numerical values of each edge to understand how the different regulatory relationships differ from each other. A new feature that is currently under implementation is customization of the normalization factor. While users can currently compare numerical weight data using the “always show weights” setting, this does not modify the thickness of the edges in the network itself.

Allowing users to adjust the normalization factor means users can select a range of edge weights encompassing both networks they wish to compare, and edge thicknesses will be decided according to that range of edge weights. If a user inputs the same normalization factors into our user interface, the edge thicknesses will automatically adjust according to that factor. If multiple graphs are adjusted to the same normalization factors, the edge thicknesses of those graphs will become entirely comparable. Customization of normalization factors will be achieved using both minimum and maximum edge weights from all comparable networks. The range of edge weights will be compartmentalized into 12 edge thicknesses using D3.js, and the resulting visualization will use edge weights according to the newly compartmentalized weight-to-edge thickness settings.

Expanding the error and warning handling system, implementing TDD, and adding additional front-end features were necessary to improving GRNsight. These additions not only conform to software engineering best practices, but also attempt to improve user experience overall. The expanded error and warning handling system encompasses several new edge cases, and ensures that GRNsight runs correctly, and that visualizations on GRNsight are created accurately. TDD encourages student programmers on the project to code new unit test prior to coding new functionality, which promotes internal and external quality of GRNsight code. Finally, adjustments to the visualization via aesthetic modifications and customization of the appearance of edge weights mean users can create network graphs for various different uses, and even compare several networks more easily. Moving forward, customizable normalization factors will continue to improve GRNsight, and will be a new feature in v2.0.1.

**References:**

Molich, R., & Nielsen, J. (1990). Improving a human-computer dialogue. *Communications of the ACM*, *33*(3), 338-348. doi: 10.1145/77481.77486

Rafique, Y., & Mišić, V. B. (2013). The effects of test-driven development on external quality and productivity: A meta-analysis. *IEEE Transactions on Software Engineering*, *39*(6), 835-856. doi: 10.1109/TSE.2012.28