BioReact: Visualization of Systems Biology Network

Haoran Yu, Zixiao Zhang

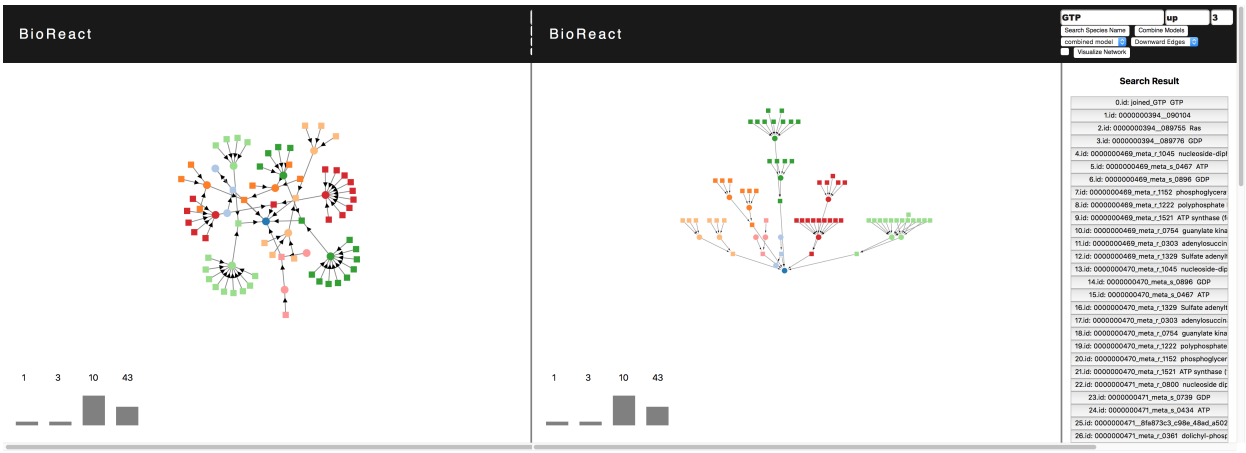

Fig. 1. The BioReact visualization. The network on the left hand side uses the force-directed layout and the one on the right hand side uses the downward edge layout. Both of the two networks are visualized using the data generated from the backend with certain filters. Users can enter the query in the panel on the right top corner. Search Result are shown in the right side bar in the list. The bar chart on the bottom is to visualize the frequency of a species's derived attribute in the displaying network.

Abstract—It is often difficult to visualize large networks effectively. In BioReact, we filter large systems biology network data by querying to select partial network as the input for visualization. Each query is parameterized by a node name, the direction of graph search, and the scope of the search. We present two layouts of the same network to clearly show network topology: a force-directed layout expands neighbouring nodes to maiximize spatial separation between nodes and links, and a downward edge layout to preserve a sense of unidirectional flow. Navigation of the network such as locating a particular node/link and linked highlighting between multiple views optimize user experience.

1 Introduction

Systems biology is one research discipline that studies cause-effect relationships between biological entities. For example, because both the biological entities `water` and `carbon dioxide` are present, the biological entity `photosynthesis reaction` can occur, which in turn causes biological entities `glucose` and `oxygen` to appear. The entity `glucose` can cause another biological entity `an isomerization reaction` to occur, which then causes the biological entity `fructose` to appear. The complex cause-effect relationships are visually represented as a directed graph, it is a conventional representation understood by all researchers in systems biology. The biological entity is drawn as a node in the directed graph, and a link is drawn between two biological entity nodes if one is the cause of the other. An example of a small directed graph is shown in Figure 2.

Researchers in systems biology are actively discovering new cause-effect relationships and modifying existing relationships between biological entities, and making a publication on the discovery. Using a small example to illustrate this idea, cell biologists have discovered a new entity A, which is a protein molecule, after performing laboratory experiments with nematode worm cells, but they do not know what its function is. Systems biologist begin to study the function of A, and after months of hard work, discovered that the entity A and a known entity B together cause the entity C reaction to occur, and causes the production of entity D in nematode worm cells. This discovery is published in a scientific journal. Using another small example to illustrate this idea, researchers knew that the presence of entity A and entity B together caused the entity C reaction to occur five years ago, but recently systems biologists discover that for entity C reaction to occur, entity D is also needed. This modification of relationships will also be published as a journal article.

As the number of relationships published increases, how do researchers keep track of the known relationships? It would be too tedious to try to find all the published articles about a particular entity and find its relationships with other entities within each article. Therefore, the solution is that a dedicated group of people actively reads newly published articles and store the new relationships in the database in text format, or add the modified existing relationships to the database. If this database is up-to-date, then the researcher can retrieve all stored relationships from this database. Dedicated software then converts the text format data to directed graphs, and the researcher is able to visualize the large network of relationships between different entities.

The main problem for all types of network data, including systems biology, is that the network is too large. If a researcher wants to find a particular entity in the directed graph generated from the network data, it is certainly difficult because the graph-drawing software can place a particular entity anywhere in the network. Even if the researcher decides to find the entity by inspecting each entity in the network, the nodes and linked are so cluttered that it creates cognitive burden to the researcher. The researcher may become frustrated and confused by the large number of overlaps between nodes and links. Suppose that the researcher finally finds the entity, it may still be difficult to visualize that entitys relationships with other entities because the graph-drawing software may place these entities far apart in space.

Our solution to effectively visualize large systems biology network

Haoran Yu email: haleyyew@cs.ubc.ca
ZiXiao Zhang email: zhangzixiao03@gmail.com
Video Demo: https://youtu.be/OFnOUvHes9s

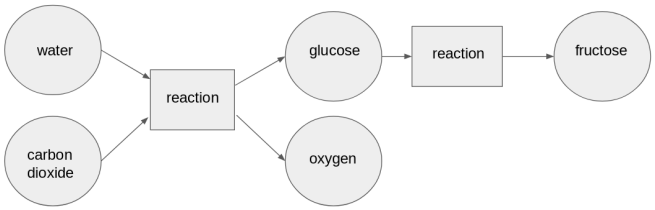

Fig. 2. An example of a systems biology directed graph. There are 7 nodes and 6 links. The 7 nodes correspond to 7 entities in the systems biology cause-effect relationships network. As will be explained later, there are 2 categories of entities: a Species entity (encoded as circles) and a Reaction entity (encoded as rectangles).

data is to implement an application that allows graph querying by entity name, filtering of data, highlighting of parts of a directed graph, and effective faceting between multiple views of the same data.

2 Related Work

One brute force approach to visualize large network data is to explicitly draw out each node and link by specifying the (x,y)-coordinates on a two-dimensional space such that there are no overlaps and clutters. If the user is interested in visualizing a small subset of large network, which is usually the case, the user first selects the nodes and links to be drawn, and then draw them out one by one. One existing tool to draw systems biology directed graphs is CellDesigner [2]. The user is presented with a blank page initially, and the user can place nodes or links anywhere on the page. When the user is satisfied with the directed graph drawn, the user can save the graph so that another user can open it later in CellDesigner to visualize the graph. While it is very convenient for the second user to visualize the saved graph, few users are willing to take the initiative to spend the time and effort to create the graphs using the network data.

A solution idiom to reduce clutter of large networks is the Topological Fisheye Views [3]. When the user is presented with the visual representation of the network, the user can focus on one specific region of the network, and the focused part of the network will be magnified. The nodes and links that were too cluttered before focusing are now more spatially separated, so that the user can clearly distinguish between nodes and links in the focused region. However, the computation is expensive to rearrange spatial positions of nodes and linked, as proposed by the algorithm. Also, since the user does not know initially where a particular node is located, spatially separated nodes and links can only be helpful to the user once that initial point of interest is identified.

A professional network data visualization tool is Cytoscape [7], it allows expert users to import network data and to perform analysis on the data. The nodes and links can be edited, such as specifying its position, its size, and color. The user can also perform query on the network to display only parts of the complete network. The interface allows annotation of nodes and links, inspect attributes associated with each node and link, and filter based on attribute values. Although this tool offers promising approach to visualize network data, its functions are not tailored for specific datasets and therefore could be difficult for a user who is not familiar with the attributes in the dataset being visualized. For example, if the user wants to filter the network data by attribute value but does not know what each of the attribute means, then the user does not know how to start.

For a better user experience, the visualization tool needs to be tailored for a specific dataset, so that all the functions in the tool have a specific meaning to the dataset. VIGOR [6] is a tool designed to visualize published works of authors. The interface permits graph querying, has multiple coordinated views of data, summarizes the data being visualized in a small panel, and allows data filtering. Our project aims to build a visualization tool mimicking multiple features in VIGOR. Our tool will also have features such as graph querying, multiple views, and filtering to alleviate the burden of users visualizing a very large systems biology network.

3 Data and Task Abstractions

3.1 Domain Background

We make a clear distinction of the two different categories of entities in a systems biology network. A `species` is an entity that has a corresponding real object. For example, a biological molecule and a chemical ion are both species. A `reaction` is an entity that does not have a physical object associated with it, it is merely a concept to describe the consequence of the presence of species. For example photosynthesis is a reaction that describes the consequence of having both water and carbon dioxide physically present at the same time. If we do not distinguish between the entity categories, each entity is represented as a node in the directed graph, but depending on the category of the entity that the node belongs to, the nodes have different meanings in the directed graph.

A species node and a reaction node are always connected alternately in the node-link diagram. A link always connects between a species node and a reaction node, never between two species nodes or two reaction nodes. This constraint is imposed by the researchers who initially designed the systems biology network data architecture of our datatset. Using this constraint, the nodes and links in the network form a bipartite graph, where all the species nodes belong to one partition and the reaction nodes belongs in the other.

3.2 Data Abstraction

The dataset we used is the BioModels Linked Dataset stored in the European Bioinformatics Institute (EBI) database. This dataset was originally stored in Systems Biology Markup Language (SBML) format, but was later transformed to the Resource Description Framework format. By transforming the dataset to the Resource Description Framework format, data can be queried using an interfacing linking the EBI database. We have downloaded the entire dataset, which consists of 636 static files, we call each file a `model`.

There are a total of approximately 40000 nodes and 100000 links in the complete network dataset, some models contain thousands of nodes and links, while others contain only a few. The data schema for species node, reaction node, and links are different, each contains around 10 attributes. However, we have selected only a few attributes to visualize. A species node contains the `id`, `name`, and `model_id` attributes (among many that we decided not to visualize). A reaction node contains the `id`, `name`, `model_id`, as well as a `list` containing links associated with the node. The `id` attribute is a unique identifier within each mode. Note that multiple models can contain nodes with the same id. The name attribute is the formal name of the entity assigned by the researcher, it has a biological meaning, that is, the name of a species or a reaction. The model_id notes which model the node is stored in, this attribute is very useful later, when we aggregate multiple models into one in out visualization. The list attribute contains ids of links, the data schema does not provide us the source node and the target node for each link, therefore it took us some efforts to retrieve this information. The id, name, and model_id are all categorical attributes since the ordering of the values have no specific meaning. Each link has 4 attributes, `id`, `model_id`, `reference`, and `value`. The id is a unique identifier of the link in the model. The model_id notes which model the link is stored in.

The reference is the species node id, i.e. it is one end of the link. The value describes the quantity of the species participating in the reaction. For example, in photosynthesis, 6 molecules of water participates in each reaction. The id, model_id, reference are categorical attributes, and the value is a quantitative attribute.

The derived attributes for each link are the `source` and `target` node id. In order to use the data visualization library, we need to explicitly store a source and target for each link. However due to the way a link is defined in the systems biology data schema, the source and target cannot be directly retrieved from a link. We performed preprocessing and precomputation of our dataset to find the source and target for each link. While performing precomputation, we have also collected additional statistics about the dataset. For each species node, we store one derived attribute, `appear_count`, which tells us how many times does a speciess name appear in the complete network. This attribute is ordinal.

3.3 Task Abstraction

Since we want to present the network data as node-link diagrams to the researcher, we want the topology of the network to be clear. That is, we do not want overlaps between nodes and links, and we do not want to have clusters of nodes and links that are hard to discern. The only way to achieve this is to avoid displaying the complete network. Since each researcher is only interested in his or her own research domain, displaying data that are not relevant to the researchers field is unnecessary. Therefore we must allow the researcher to query for parts of the network that he or she wants to see, and only display that part of the network.

To query the network, we define additional parameters that the researcher must specify before querying. The parameters are the `name` of the species, the `scope` of the partial network, and the `direction` of the query. The name is the point of interest in the network, when the researcher searches for a name, all species nodes having that name are returned. However, it is useless to visualize single nodes, the researchers goal is to find out the relationships of the queried entity with other entities in the network. Therefore we need to display all adjacent nodes connected to the queried node via a link. The scope parameter specifies how far in the relationships cascade does the researcher wants to see. For example, the scope is 1 if the queried node and its immediate neighbours are being visualized. The scope is 2 if the queried node, its immediate neighbours, as well as the immediate neighbours neighbours are being visualized. To explain the concept of scope in another way, a queried entity and all entities that directly cause the queried entity (defined `EU`) as well as all entities that are directly caused by the queried entity (defined `ED`) form the scope 1 visualization. For scope 2, all entities caused by EU and ED as well as all entities that cause EU and ED, are included in the visualization as well. The direction parameter filters out parts of the entities set returned within the scope. The direction parameter takes values `up` and `down`. If up is specified, only the entities set EU and the queried node are returned. If down is specified, only the entities set ED and the queried node are returned. To put it in simple terms, if up is specified, we only see all entities that cause the queried entity within the scope. Similarly, if down is specified, we only see all entities that are caused by the queried entity within the scope. The parameters that we define for querying is due to the network being a directed graph, which stores the cause-effect relationships. Naturally, as the set of nodes are returned, all links that connect the nodes are also returned.

Once the partial network is being displayed to the researcher, we permit navigation of the network. For example, browsing for the information of each node or link, locate a particular node or link in the partial network, and explore each nodes neighbours. To locate a series of consecutive links, we locate the path from the first node to the last node in the set of consecutive links. For example, the researcher may want to know for a given queried node, how does the queried node's corresponding species undergo a series of reactions to become a different species? To use a concrete example, suppose that species A cause reaction B, reaction B cause species C, species C cause reaction D, and reaction D cause species E. We can locate the path from A to E via the consecutive links linking between species and reaction nodes in between. For a systems biologist, this is an analysis task that can be achieved using our visualization tool.

Table 1. What-Why-How analysis of BioReact

What: Data	Around 40000 nodes and 100000 links in the complete network, distributed across 600 models. Species Node and Reactions node with attributes name, id and etc., links with attributes source, target, value.
What: Derived	The frequency that a species name appear in the network. The source and target of a link.
Why: Task	Node-link diagram. Query for specific node by name. Browse, locate and explore parts of the network. Show the topology of the network.
How: Encode	Circle and rectangle for species and reactions. Line with arrow to denote the direction; color and saturation for different group. Stroke width of the line for value of reactants. Label by click for specific information. Bar chart for derived statistics.
How: Reduce	Query; Filter; Dynamic aggregation.
How: Manipulate	Navigate: scroll, drag and translate. Select: check details, change node size and show path.
How: Facet	Linked Highlighting: button on the side bar, ordinal bar chart.

4 Solution

In this section we describe our visualization solution and analyze it using the What-Why-How framework from [5], Table 1 provides the summary. Our visualization design can be generally divided into 3 parts: network, interactions and bar chart.

4.1 Networks

Since we want to clearly present the topology of the network, it would be desirable to display multiple layouts of the same partial network, and let the researcher decide which layout is better to perform the analysis tasks. We presented two techniques for drawing directed graphs, each produces a different style of layout. The first technique is physics simulation to maximize spatial separation of nodes and links within a constrained space. The technique that we applied borrows ideas from circle packing in Wang et al. [9], which attaches a spring to the node and pulls the node towards a center while avoiding collision between nodes. We call this layout the `force-directed layout`.

The second technique we applied is IPSEP-COLA [1], it draws Sugiyama-style directed graphs [8]. In an older technique proposed by Gansner et al. [4], determining the optimal layout of a directed graph is broken into four steps. The first step partitions each node into different levels, where each level is separated by a fixed length of y and all nodes in the same level have the same y-coordinate. The optimal partition is computed by minimizing the length of the link connecting two adjacent nodes, subject to the constraint of a minimum length. The second step assigns the order of nodes in each level, minimizing the edge crossings between levels. The third step specifies the (x,y) coordinates of each node, while maximizing the possibility of drawing straight edges between levels. Finally, the edges are drawn. For each edge, define bounding boxes and the edge has the freedom to move anywhere within the bounding boxes. The IPSEP-COLA technique uses a different approach, it uses gradient projection to minimize a stress value, where the stress value is how far away the current layout deviates from an optimal layout subject to a number of predefined constraints such as minimizing edge crossings. We call this layout the `downward edge layout`.

Our design choice of including both the force-directed layout and downward edge layout is justified below. We filter the data by users query species name and the species name must be the focus in the visualization. The force-directed layout is to put the node in the centre and make the structure radial, with distance to denote the relationships between other nodes and the center node. The downward edge layout is to make the network into a hierarchical model such as a tree in order to show lineage and parent-child relationships. Also, this layout has arrows of edges pointing in the same direction, thus creating a strong sense of flow from top to bottom. This organized layout also allows clear visualization of paths.

Since nodes have 2 categories: species and reactions, we use the circle and rectangle mark to represent the species and reaction nodes respectively. Links not only indicate the flow direction between nodes, but also give out the quantity information in the reaction. Intuitively, we use the stroke width of the link to denote the quantity. Following the standards in systems biology, we add fixed size arrow mark to the link to avoid the overlap problem (that is, arrow size increase as the width of the line rise due to inheritance).

The data has an attribute model_id, thus we use colour and luminance to group nodes belonging to different models (because the number of the models in the query result is usually below ten). The nodes in the same group will be clustered together. The force-directed layout looks like a flower with each group as the petal, while the downward edge has groups of leaves connected to the root node. Both of the two layouts face the problem of overlap. The downward edge layout face the problem more severely in the horizontal axis when number of nodes per level becomes larger.

4.2 Interactions

In a network with large number of nodes that look similar, it is easy for the user to feel confused without any highlighting. Given that the list of node name is shown in the search result side bar, we use linked highlighting to reflect changes to a node in the network panel corresponding to clicking the node name in the side bar. Once an item in the result list is clicked, its color is changed to blue. While it is not reasonable to use the colour channel again for a node, we consider the usage of size instead. The purpose of the interaction is to help user find out certain species or reactions location in the network. We extend the the radius from 4px to 8px for the circle node as the species name is clicked in the list. And we extend the width and the height of the rectangle node from 8px to 15 px as the reaction name is clicked in the list. In the reverse interaction, clicking a node will enlarge the node and also link the change of colour of the item in the result list.

To ensure that full details of each node is displayed in the network visualization, we designed a pop-out label triggered by the clicking of a node to show the id, name, model_id and whether the node is a species node or a reaction node. To mark the path between a pair of nodes, we mark the path with colour red which is in contrast with the original link colour grey. In general, the interaction in the visualization assists the user on building an idea of the relative location of the nodes in a large-scale network.

We retain the colour of items in the result list as we switch one layout to the other, if we are still visualizing the same network data (i.e. the same model). The reasoning is that the users memory of highlighted items in the result list must be retained because he or she wants to make comparisons between the two layouts of the same network data.

4.3 Bar Chart

We encode the derived attribute appear_count as a bar chart, the number of times a speciess appears in multiple models is the bin, and the height of a bar is accumulated for all nodes satisfying the appear_count value. For instance, the bar with attribute value of 2 as the label and a height of 10 means that 10 species in the current visualized network exists in 2 models. As a bar is clicked, all species items that belong to the category are highlighted in red in the sidebar.

5 Implementation

We divided our work into two distinct categories, implementing the backend engine for answering queries and implementing the frontend interface for data visualization.

5.1 Backend Engine (Haoran)

The dataset is stored in XML format, however the JSON format is required to effectively import the data to a web service. The dataset was parsed using python scripts. The parsed data serve as the objects stored in our pseudo-database. We build indexes of the data by computing statistics - for each species name, find all models that contain the species name, and create a key-value pair using the species name as the key and the list of models as the value. To answer a query of a specific species name, find the key-value pair in the index. From the values in the key-value pair, we are able to determine which JSON models to search, whereas if we did not have an index we need to search over 600 models for every query.

Once a queried node is located in a model, perform a breadth first search in the direction specified by the parameter, direction. For example, if direction is up, then reverse the direction of all the arrows and perform breath first search using the modified graph. The scope parameter limits the depth of the search, once the search reaches the specified depth, stop the search and return all nodes and links traversed so far. To answer a query of path between two nodes, construct an adjacency matrix using the partial network currently being visualized, and then run Dijkstras single-source shortest path algorithm. The algorithm return the shortest path between the pair of nodes. If there is no path between a pair of nodes, return an error which is caught at the frontend script.

To aggregate multiple partial networks returned by the query into a single network, we simply aggregate the N query species nodes present in each of the N models returned in the query into a single node. For each link connected to the original species node, change the links connection from the original species node to the new aggregated species node. This operation guarantees that if all the nodes were connected to a models partial network, they are still connected in the combined network.

5.2 Frontend Visualization

The BioReact was implemented as a Web Application frontend using HTML5, CSS3 and Javascript. `interface.html` includes the HTML elements, CSS3 elements and Javascript code we wrote for UI construction, data processing, user interaction and visualization functions. `interface_style.css` includes part of the attributes of the UI widgets.

BioReact utilizes libraries and framework aiming to follow the trend of the frontend development, and providing a robust and aesthetic system for visualization. (i) We used `Bootstrap 3` framework to set up the responsive UI of the system, with a navigation bar at the top, a main view with 3/4 width of the window and a side view with 1/4 width on the right. Widgets like input bar and buttons were mainly self-designed. (ii) We used `jQuery` to handle the data, optimize the manipulation of the DOM , and extend the CSS selector. (iii) For data visualization, we mainly used `d3.js` to generate the SVG graph. D3 is a powerful javascript library with various existing extensions. We carefully chose some framework and implemented them with modifications to adapt to the visual encoding prototype. Here are the additional frameworks for visualization we used in the project:

`cola.js` and `cola-downwardedges.js`: cola.js is an open-source javascript library for arranging the HTML5 documents and diagrams using constrained-based optimization techniques. In this project, we chose the downward edges layout with constraints from Cola.js to make our network more organized. We modified the framework and added visual encoding to distinguish between species and reactions, as well as handling the node on-click event.

`d3v4-selectable-force-directed-graph.js` and `d3v4-brush-lite.js`: We utilized the colour encoding, selection and drag specification, so as to make our force-directed layout more interactive. We added the information label, click event, dynamic

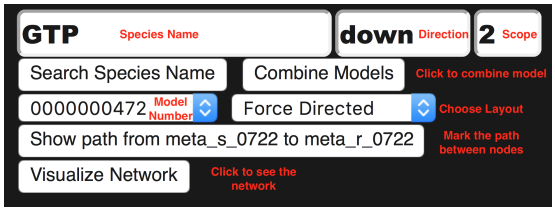

Fig. 3. The meaning of each widget in the panel.

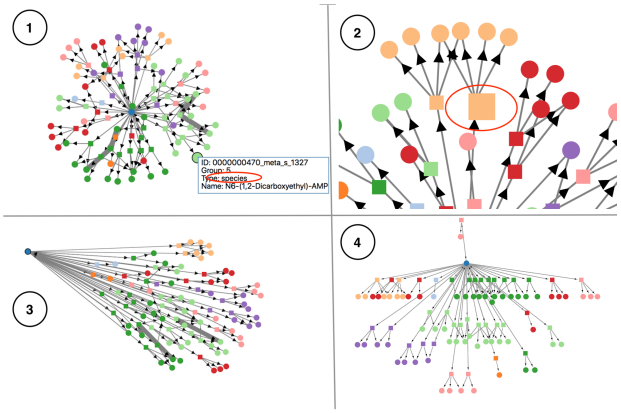

Fig. 4. There are four numbered screenshots in this figure. Screenshot 1-3 are the network with force-directed layout. 1 shows the specific detail label triggered by click event. 2 shows the size change of the node. 3 shows the appearance change by drag event. Screenshot 4 is the network shown when user switches to the downward edge layout.

shape and size change to the original code. And implemented functions to allow the interaction between the networks and other elements in the UI.

Zixiao is mainly responsible for implementing the visual encoding of the network data. Zixiao also worked on the development of the UI and part of the user interaction such as dynamic node size change. Both of the authors contributed to the general design of the interface and the integration of the codes. Haoran fixed bugs in the front-end and implemented path highlighting as well as downward edge layout.

6 Results

The proposed visualization is designed to present a more time-saving way to check the necessary information relevant to the systems biology domain. Our targeted users are mainly learners who want to look up the relevant details in the database and junior researchers who need a tool to help them set up their research plan. In addition, considering the flaws in the database, we try to make the tool helpful for administrators to do modifications on the database.

6.1 Scenario Walk-Through

Our system is a search engine facing a static database. User needs to type in the species name, direction of traversal and search scope as the query parameters. The direction of traversal take on the value of 'up' or 'down'. Species name is the name of the species the user obtained or planned to synthesize. User can choose from the two layouts of network visualization. The user can combine all the models in the dropdown list by clicking Combine Models. See Figure 3.

For the scenario of use, let's suppose that Jack is a student in biochemistry, trying to figure out what GTP can be used for. So he types in GTP, 'down' and a scope number. After the backend engines returns the result data, he can look through the combined model network or look into the single model network.

In the combined model case, he can choose from two layouts. In the force-directed layout, a radial network with the node in the centre referring to GTP and all other nodes clustered, Jack can easily distinguish

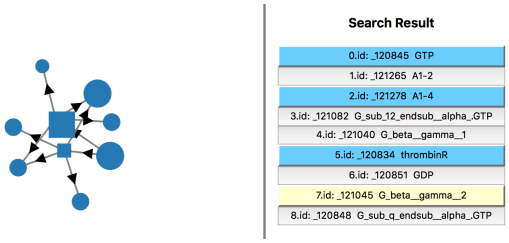

Fig. 5. In this single model view, 3 nodes (2 species and 1 reaction) were marked by the user with bigger size. One node was marked but the user finally found it useless and canceled the mark. In the meantime, the marked ones were highlighted in blue and the canceled one was highlighted in light yellow in the side bar.

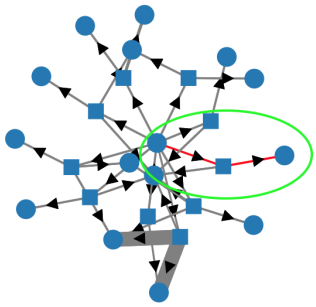

Fig. 6. Example of mark on the path using red color, indicated in the green circle.

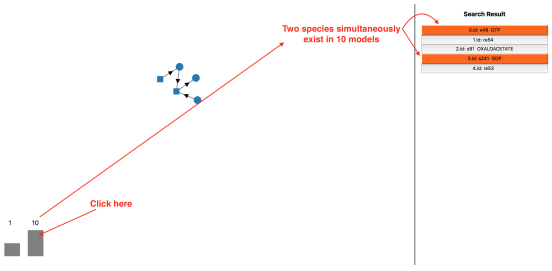

Fig. 7. The bar chart link-highlighted the species in the result list.

the difference by colour and luminance. By clicking the node in the network or the buttons on the side bar, Jack can grasp the relative location of each node in the network. Jack can zoom in, drag and translate nodes to find a better perspective for himself to observe the layout. In the downward edge layout, Jack have fewer things to do because the layout is well-organized and constrained. He can simply follow the tree downwards to find out the paths for the cascade of reactions using GTP. See Figure 4.

In the single model case, Jack inspects an individual node in the network, he can click it which makes the node bigger. If the species is meaningless for him, he simply clicks the blank area and the corresponding node will shrink back to the original size. On the contrary, he can proceed to the next species by clicking the node and gradually highlight a path that he wants to study. He then proceeds to perform research on each of the species and reaction nodes in the path he highlighted. If he finds one of the species interesting and wants to study it in more detail, he can enter that speciess name in the query and visualize the network for that species. See Figure 5.

In both two cases, Jack can mark a path between two nodes by clicking their ids in the side bar and press the Show Path button. See Figure 6. Also, by clicking a bar labelled '2' with height 3 in the barchart, Jack can check what those 3 species are in the side bar. See Figure 7.

In addition, all the entries he has already browsed were marked in light yellow to avoid repeated work in checking. Jack can switch between the two layouts, while still maintaining the highlighted items in the sidebar.

6.2 Evaluation

We preliminarily present our system to potential student users for feedback. Two Phd students from ECE BME laboratory approved our work on reducing the complexity of the network and gave positive feedback on the clarity of the topology. However, they pointed out that the downward edge layout seems better than the force-directed layout for the structure that suits their cognitive system. They also suggest to add visual encoding into the single model view, e.g. use different shapes to indicate the start node and end node. They reminded us that the overlap problem needs to be carefully tested under various circumstances, especially with the a huge number of nodes.

Two graduate student from Chemistry said that our visualization is designed properly but the data representation in our system remains to be optimized for practical use. They argued that the ids shown in the side bar may be useless in some cases because user may not be familiar with the ids. A node-link diagram for each of the specific reaction node needs to be implemented, so as to give the user a deeper idea of what each reaction is used for. For the user experience of the system, the monotony of color in single model view was also mentioned.

7 Discussion and Future Work

7.1 Limitation of Visualization

Time constraints lead to the limitation of our evaluation, but problems are quite clear given peers advice and our self evaluation. One obvious limitation lies on the interaction. Although some users may feel well to have parameters used to configure the network, others wants to avoid the compulsory configuration procedures. Another limitation is the incompleteness of the downward edge layout. Even if it is designed to be concise, we still find the necessity to provide more interactions as what we did in the force-directed layout. For example, allowing zooming in and out because zooming is especially efficient when the level number is greater than 3.

The limitation on the data is intrinsic. In our development process, we pretended to display the names of the species on the side bar. However, it is not practical because some species do not have a name in the database. Given the feedback, we display both the id and name in the result list. In the test process, we find that error or absence of a value in the database lead to the ambiguity and affect the understanding of the user.

7.2 Limitation of Browser

Due to the processing ability of the browser, the performance of our visualization falls rapidly when the parameter scope number increases. This issue exists in three major browsers, Chrome, Safari and Firefox. For instance, the graph is generated slowly and latency in the interaction is quite obvious when the level comes to 3, with 600 nodes and links. The specifications of our design are severely limited by this factor even if the quality of the visualization is the same.

One limitation concerned with the front-end limitation is that the data we used are static which is permanently stored in our local sever. However, in a more general case, there must be a backend to support the frontend. Lastly, there must be a large number of changes on the structure of the system in order to integrate them in a server.

7.3 Lessons Learned

In this project, we apply the visualization idioms in the books to provide a reasonable solution to a domain problem. One lesson we learned is carefully choose visual encoding method to avoid ambiguity. Facing thousands of data, each decision on colour encoding was complicated. So we applied efficient reduction in data size and adjusted the encoding method accordingly. The other lesson is to have a meticulous plan at the beginning. We changed our visual encoding method in the process and gave up some ideas which we used to think helpful for users. However, this costs us a lot of time and interfered with the original design of the project. Instead, we must talk to the targeted users first with multiple applicable design choices to our final design.

7.4 Future Work

As we mentioned before, the most urgent work to do is to optimize the visualization performance given different queries. So we must carry out the tests in the extreme conditions to see whether the visualization is feasible or not. For real applications, we need to allow the synchronization of the data on the sever end and front end, for example the data used in the visualization can be directly modified on the sever. And we cannot ignore our original purpose to offer a tool to correct missing data in the database. So we plan to develop an interface which allows the administrator to directly change the data from the front-end. In addition, data processing needs to fulfill the requirement of the user.

8 Conclusion

In this paper, we started from the analysis of terms and relationships of data which are commonly used in the work of systems biology and related domain. And we made a summary on the advantages and disadvantages of the current visualization tool. We emphasized the problem of the clutter when the visualization is applied to a huge dataset. Then, we proposed our solution to simplify the problem and offer the network partially according to the users' query.

In our project, we presented BioReact, a tool which provides users such as students and researcher with a partial network related to the EBI database. For the visualization, we implemented two layouts in the parallel relationships to satisfy the possible user preferences. In the visual encoding of the data, we carried out multiple trials to bring all the factors in our design to the balance. To assist the users to understand our network, we designed several interactions including linked highlighting and path highlighting. We hope our targeted user can use our visualization system in the future.

References

[1] T. Dwyer, Y. Koren, and K. Marriott. Ipsep-cola: An incremental procedure for separation constraint layout of graphs. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):821–828, 2006.

[2] A. Funahashi, Y. Matsuoka, A. Jouraku, H. Kitano, and N. Kikuchi. Celldesigner: a modeling tool for biochemical networks. In *Proceedings of the 38th conference on Winter simulation*, pp. 1707–1712. Winter Simulation Conference, 2006.

[3] E. R. Gansner, Y. Koren, and S. C. North. Topological fisheye views for visualizing large graphs. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):457–468, 2005.

[4] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993.

[5] T. Munzner. *Visualization analysis and design*. CRC press, 2014.

[6] R. Pienta, F. Hohman, A. Endert, A. Tamersoy, K. Roundy, C. Gates, S. Navathe, and D. H. Chau. Vigor: Interactive visual exploration of graph query results. *IEEE transactions on visualization and computer graphics*, 2017.

[7] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003.

[8] F. Wakamori, S. Masui, K. Morita, and T. Sugiyama. Layered network model approach to optimal daily hydro scheduling. *IEEE Transactions on Power Apparatus and Systems*, (9):3310–3314, 1982.

[9] W. Wang, H. Wang, G. Dai, and H. Wang. Visualization of large hierarchical data by circle packing. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pp. 517–520. ACM, 2006.