Visualizing Structured Hypotheses in the Gene Ontology with Focus-and-Context

Junjie Zhu*
Department of Electrical Engineering
Stanford University

Qian Zhao†
Department of Statistics
Stanford University

ABSTRACT

Human knowledge of gene products and their attributes can be summarized in the Gene Ontology (GO). The GO can be presented as an directed acyclic graph (DAG), where each node is biological vocabulary term, and each edge represents relationship between terms, such as "is a" or "is a part of". The GO DAG includes over tens of thousands of nodes that can be elusive to display or interactive with for exploratory data analysis. Here, we introduce a novel notion of focus-and-context for data exportation in large DAGs across multiple contexts. The visualization we propose decomposes a large DAG into context representations that summarizes the whole graph and focus representations that capture fined-grained information. To deploy our visualization principles for GO analysis, we developed a web application which allows one to interpret graphical structures of the significant discoveries the experiments suggest within the GO. The application is built upon an optimized computational framework with hierarchical data structures for rendering efficiency. As a result, one can explore multiple facets of the GO via instantaneous interactions and develop new insights into how their underlying data fits in the large knowledge base.

1 INTRODUCTION

Within a specific domain in biology, search for all available information performed by human or computers can be strongly hindered by a wide variations of annotation and terminology. In particular, the annotation of genes and gene products created by decades of research can include functionally many equivalent terms or notions. The Gene Ontology (GO) [9] is a knowledge base that addresses the need for consistent descriptions of gene products. As such, GO has been developed to include three structured controlled ontologies that "describe gene products in terms of their associated biological processes, cellular components and molecular functions in a species-independent manner" [1].

The use of GO terms enables scientists to perform uniform queries across different studies and databases and interpret the genes or gene products they identify in their own experiments. For instance, some scientists may want explore the existing knowledge base to answer questions such as what biological processes are associated with human eye color? They can analyze the existing terms and annotations in the GO database, and discover previously known processes associated with their outcome of interest.

1.1 The Gene Ontology Graph

The GO can be represented as a directed acyclic grapgh (DAG). A directed graph is a collection of nodes and directed edges between nodes, and the graph is acyclic if the direccted edges do not form a closed loop anywhere in the graph. In the GO DAG, a node represents a GO term, an edge relation between the nodes. A sequence of edges is called a path if node at arrow head of previous edge is at arrow tail of the next edge. If two nodes are connected by an edge, refer the node closer to the roots as parent node, and the other child node. A node is called a root if it has no parents and a leaf if it has no children. There are three root nodes in the GO DAG: biological process, molecular function and cellular component, they are placed at top of the graph.

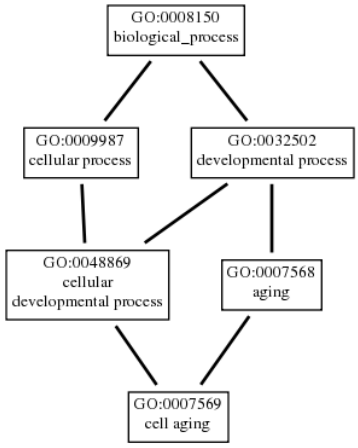

Figure 1: A subgraph of the GO DAG [6]. The node "cell aging" has two parents: "cellular developmental process" and "aging". It if a leaf node because it does not have any children. In this subgraph, all nodes other than "cell aging" are its ancestors. "Cell aging" has depth 3 and height 0.

For simplicity, we will mainly consider the biological process graph, which is a rooted DAG (e.g., a DAG with only one root). Following standard definition of rooted trees, the depth of a node refers to length of longest path to the root, and the height of a node refers to length of longest path to any of its leaves. Unlike a rooted tree, a node can have multiple parents in a DAG, making the DAG generally more difficult to visualize. The ancestors of a node is the set of all nodes reachable by recursively defining child-parent relationship; conversely, the descendants are all nodes reachable by recursively defining parent-child relationship. For example in Fig. 1, "cell aging" has two parents, "cellular developmental process" and "aging". "cellular process", "developmental process" and "biological process" are all ancestors of "cell aging".

1.2 Multiple Hypothesis Testing

The problem of determining association between a GO term and outcome of interest can be formulated as hypothesis testing on the corresponding node. The null hypothesis is H_0 : the gene ontology term is not related to outcome of interest. Alternative hypothesis H_1 is the otherwise. The p-value characterizes how extreme observed data is if null hypothesis is true. A smaller p-value provides stronger evidence against null hypothesis.

If a null hypothesis is true (e.g., no associations exist), its p-

*e-mail: jjzhu@stanford.edu
†e-mail: qzhao1@stanford.edu

value would be uniformly distributed on [0,1]. Thus, if one is testing a large number of null hypotheses, say 10,000, on average 10 are falsely rejected if 0.001 is used as cutoff for rejection. This phenomena is called the "multiple testing burden" and often occurs in genetics. To adjust for the multiple testing burden, multiple testing corrections are applied to these p-values [3, 14, 15, 24, 26]. For instance, the Bonferroni correction controls family-wise error rate (i.e., the probability of making at least one false rejection) at level q by setting rejection threshold at level q/N when the number of hypotheses tested is N. If hypotheses are hierarchically structured on a DAG, one might want to preserve the logical structure when testing them, for example impose that rejected hypothesis be a subgraph of the original DAG. Recently many procedures have been developed to test hypotheses on a DAG [11, 21] or a tree [4].

After multiple testing correction, a scientist would be interested in set of nodes that are "rejected", that is, statistically significant. For the purpose of this paper, we will be focusing on the set of rejected nodes and where the locate in the GO DAG. Thus, the visualization can reveal structures among the tested hypothesis and what biological terms are significant.

2 Related Work

2.1 GO-specific Visualization

Existing tools for GO DAG visualization typically perform two main features: one is query, where the tool displays what other GO terms are related to a query; the other is output of results from data, where the statistical testing results are displayed.

For instance, AmiGO2 [6] is a commonly used web-based interface to query, browse and visualize GO terms. Users can supply a list of GO terms and display them on a graph. GOLEM [25] displays a subgraph containing the node of interest. It also allows the user to query for a specific node by name, and overlay the gene enrichment analysis p-values by user. REVIGO [28] performs clustering of gene ontology terms based on semantic similarity and selects one representative for each cluster [18]. Users can select one of four methods to calculate similarity matrix between GO terms. A GO term is them shown in its location in the semantic graph in a scatterplot. Node size is expression level in background, its color encodes significance level. An interactive graph is also available where the user can drag nodes and magnify the graph.

Despite numerous options to create GO graphs, to our knowledge, all these tools are limited to local information of the GO DAG, but do not indicate how local rejections fall in context of global DAG structure.

2.2 Large-scale Graph Visualization

Broader graph or network visualization techniques can potentially handle DAGs to handle GO visualization. The focus-and-context principles have been successful in other data applications, such Table lens [22], yet limited applications of focus-and-context have been proposed for large graphs (e..g, those with $> 10,000$ nodes). Herman et al. [13] provide for an excellent review of focus-and-context technology in displaying graphs, and here, we also summarize the general strategies that can improve display of large graphs.

The problem of visualizing a structured graph can be formulated as optimally positioning nodes and edges on a canvas to satisfy some aesthetic criteria [19, 20], such as minimum crossing between edges [2, 10]. Some traditional layouts include tree layout [23, 30], spring layout [2], and Sugiyama layout [27]. When the graph size is large, it is difficult to display the entire graph in detail. It can even more difficult to solve the prescribed optimization problem. In general, strategies one can take can be classified into two categories: clustering of nodes hierarchically and display information in accordance to hierarchy, or highlighting graphs via pre-specified objects of interest or via interactions.

Clustering can be performed with or without a distance measure. Distance measure can be content-based, such as how much information two nodes share, or by structure of the graph, such as counts of number of shared neighbors. Clusters can also be rendered by structure-based methods, such as force-directed methods, where natural clusters emerge after a few iterations. After labeling hierarchy of each node, one can choose to number of nodes to display nodes or amount of information at each resolution level. One can also zoom-in the graph in terms of content or in size of groups.

In terms of highlighting information, one method is to use alternative geometry. For example, one can project hyperbolic geometry into Euclidean geometry [17]. One can also use distortion: given focus point as origin, one can distort distance x to it as $h(x)$. Some examples include fish eye display, and bifocal and multi-focal display [16]. This process is analogous to magnifying a part of graph, see [8] for many forms of lenses and their effects. Additionally, Degree-Of-Interest (DOI) is a commonly used strategy to highlight information interactively [7]. The user can input a location of interest, for example by mouse over a region, nodes that are high in DOI will be highlighted, whereas others will be muted. DOI is calculated as compromise of intrinsic importance and relevance to user's focus.

In the domain of general display for large graphs, it is difficult to find a one-size-fits-all solution because inherent graph structures can vary from application to application. Data-specific visualizations can typically render better results than generic graph visualizations because inherent characteristics of the graph or user query patterns can help guide the design to provide the appropriate information. We will take the latter approach to design an application specific to GO DAG in the following sections with some of the aforementioned principles.

3 Methods

3.1 Focus-and-Context Principle and Design

The principle of focus-and-context involves representing visual elements on multiple scales: *focusing* on a smaller collection visual elements of interest while maintaining the *context* of the background elements. Here, we formalize this principle to large DAGs and define the notion of a *context graph* and a *focus graph* using graph theoretic properties of DAGs.

Context Graph We abstract the entire DAG via a summary bar plot as the context graph (as shown in Fig. 2), where the order of the bars correspond to the levels of the hierarchy in a DAG, and the length of the bars count the number of nodes on each level. Because a DAG has well-defined root nodes and leaf nodes, the ordering of the node can have a direct interpretation, such as the maximum distance from the root or the leaves. Thus, each bar counts how many such nodes share the same distance from the root or leaves. For a large DAG with tens of thousands of nodes, we argue that it is unnecessary to display all nodes or edges because one cannot process the massive amount of information in a short amount of time, and fine-grained interaction with ten thousands of nodes and edges is prohibitive. A bar plot on the other hand can unambiguously convey precise information of how many nodes are on each level without showing the individual nodes. This representation can be particularly useful for hypothesis testing where one might ask the question of how many nodes are significant in the original DAG, because the bar representation can break down the significant nodes into their particular layer and allow hierarchical patterns to emerge.

Focus Graph Given a set of query nodes (i.e., nodes one is interested in), we display the nodes as well as their ancestors and descendants in a focus graph (as shown in Fig. 2), where all nodes and edges are displayed as a bone fide graph. The nodes in the original DAGs by definition have no relationship with query nodes, even though they may be related to the displayed ancestor or descendant nodes. Importantly, the levels of nodes in the focus graph mirrors the

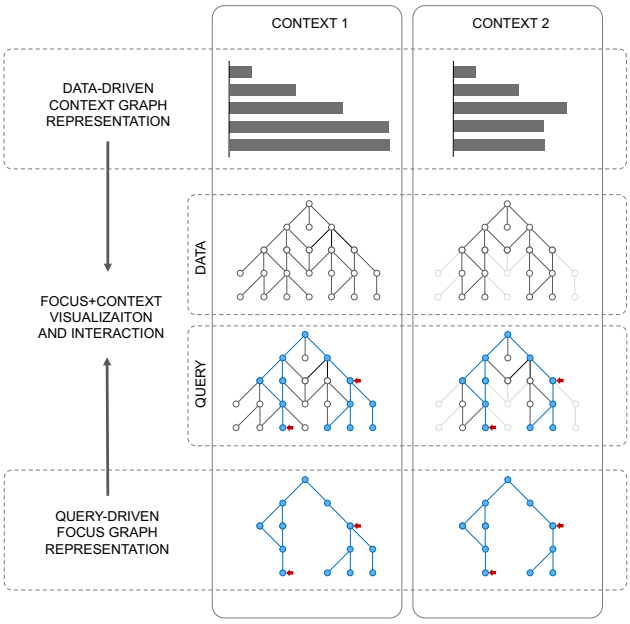

Figure 2: The large DAG is represented abstractly via the context graph, and queries and their relevant nodes (all ancestors and descendants) are concretely represented via the focus graph. The combination of the two representations is rendered for interactive visualizations. Additionally, the context graph can be a subgraph (which is also a DAG) of the original DAG, and focus graph can be generated from this particular context.

level ordering within the context graph - their original orders in the large DAG. The ancestor and descent nodes provide a local context of where the query node lies in the original DAG, and reveal the paths connecting the query nodes to the relevant roots or the leaves. As long as the query nodes are specific enough, the number of total descendants and ancestors is typically limited, and one may now be able to look at local graph structures much closer and interact with individual nodes related to the query nodes. The interaction between related nodes can be highly useful for understanding how similar concepts to the query nodes are organized and fine-grained details about these nodes.

Our interpretation of focus-and-context generalizes to multiple context graph representations of the data and enables the interesting focus graph to be viewed within different contexts (see Fig. 2). We consider two contexts: the "Total" context (Fig. 2 left) corresponds to the entire GO graph. The "Statistical" context (Fig. 2 right) removes nodes that are not tested. The Statistical context can be more meaningful when a scientist wants to investigate nodes of interest among tested nodes, because error-control methods, , such as Bonferroni correction explained in Sect. 1.2, often set threshold that depends on number of hypotheses.

3.2 Computation Framework and System Architecture

Data Structure Because the selection of context, choices of layout and interaction states can be combinatorial, we propose a new computing data hierarchy and computing framework that can efficiently support the layout rendering and interactions. In our framework of focus-and-context, a data object includes 1) primary attributes that are invariant across any context and focus representations, 2) secondary attributes that can vary under different contexts but are invariant to what the focus graph is, and 3) tertiary attributes that can vary with what is focused on during interaction. All these attributes collectively define how an object is displayed on the front-

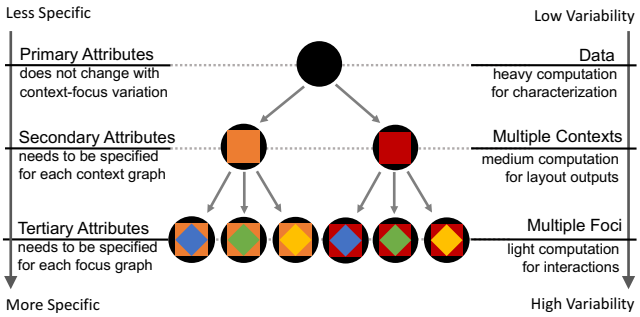

Figure 3: The data objects consist of hierarchical attributes based on whether they change with the context-focus specification. The primary attributes are invariant to the selection of context and focus; the secondary attributes are invariant to the focus; and the tertiary attributes define different ways to focus on a graph. This data structure also specifies the computation hierarchy and minimizes the redundant computation and storage.

end interface, but because of the attributes can be expensive to compute so we define data hierarchies to reduce the space time needed for computations.

As a toy example illustrated in Fig. 3, let us consider the rendering of a circle combined with a square and a diamond. As there are two ways to color the square, and three ways to color the diamond, there are a total of six different renderings of this data object. Further, each object may also require different computation time. In our application, rendering the black circle would correspond to retrieving the entire GO DAG from a database and characterizing the node properties such as how many genes are associated with each GO term. The squares would correspond to the characterization of DAG or subDAG that we use as the context graphs, and the diamonds would correspond to the focus graphs, to which most of the interactions are applied. When adding a new feature to the application, it is critical to define where the feature lies in the hierarchy and reduce the need to re-compute the same data values. As a result, the front-end display can render faster and allow for instantaneous (e.g., $< 0.1s$) interaction updates.

Core Algorithms To support multiple contexts and focus graphs, we developed a customized layout algorithms to generate different context graphs and focus graphs.

We use algorithm 1 as a sub-routine to parse though levels of a DAG. The inputs into the sub-routine is a set of nodes of interest, a relationship that is desired (either "parent" of "child"), and a set a context set of nodes. For instance, we initially traverse through the entire DAG given the data, and then compute the level of each node in the context of the entire data. To compute the depth of each node, the input node set would be the root node, the desired relation would be "child" and the context set would be all nodes in the DAG; to compute the height of each node in this same DAG, the input node set would be the set of leaf nodes, the relation would be "parent", and the context nodes would also be the full set. When the context changes, this process can be repeated to characterize node height and depths in the new context. Instead of using recursion, we use a queue data structure to allow for higher efficiency during the search.

For the layout of focus graph nodes, we introduce a mean-ordering heuristic in algorithm 2 to define where nodes are positioned on each level of the DAG hierarchy. Unlike tree layout algorithms, DAG layout can be significantly more challenging, and the problem of minimizing edge crossing has been shown to be NP-hard. Thus, we adopt this heuristic to locally reduce edge crossing. In practice, when the focus graph is not too large, the heuristic sufficiently reduces the edge crossing. One alternative solution would be force simulation

Algorithm 1: Context-aware Graph Traversal with Queuing

Input: queryset, and Full DAG
Param: contextset and relation ("parent" or "child")
Output: outputmap, a map with keys being all the ancestors or the descendants of queryset restricted to only the contextset and values being the level of the node

```
outputmap ← {};
currentlevel ← 0;
queue ← Intersection(queryset, contextset);
while queue ≠ ∅ do
    currentcount ← length(queue);
    while currentcount > 0 do
        currentnode ← popleft(queue);
        outputmap [ currentnode ] ← currentlevel;
        for neighbor in get(currentnode, relation) do
            if neighbor in contextset then
                pushright(queue, neighbor);
        currentcount ← currentcount −1;
    currentlevel ← currentlevel +1;
```

Algorithm 2: Focus Node Ordering in Hierarchical Levels

Input: focusgraph
Param: contextset and relation ("parent" or "child")
Output: outputmap, a map with keys being input nodes and values being their order in their level

```
outputmap ← {};
previouslevel ← 0;
foreach node in focusgraph do node.position ← 0 ;
while previouslevel < maxlevel do
    currentlevel ← previouslevel +1 ;
    currentnodes ← getnodes(focusgraph, currentlevel) ;
    foreach node in currentnodes do
        neighbors ← get(node, relation) ;
        node.score ← mean(neighbors, position) ;
    Order position of currentnodes from 0 by their score;
    previouslevel ← currentlevel;
```

layout, but we decided on a static layout over a dynamic graph layout because we want to preserve the node count and reproducible node ordering when visualizing queries and their corresponding focus graphs. The DAG software library we implemented is portable and extends to the use case for GO analysis where we can construct the DAG from the real data, interpret the genes that are associated with each node in the DAG and perform hypothesis testing on each node. Similarly, we create javascript libraries to input the hierarchical data structure that we define to render focus and context graphs.

System Architecture Our system architecture (shown in Fig. 4) is built upon this data hierarchy and prioritizes computations required for each attribute the data object to be rendered. In particular, we used Python to manage the back-end data communication with the GO database and perform heavy statistical and graph computations on tens of thousands of nodes. The primary attributes (such as statistical testing results and the genes associate with each node) are typically independent of layouts so we can cache these attributes as pre-computed results. Multiple context representations can also be computed as different instances of the corresponding secondary attributes: the children/parents and height/depth of each node in the context graph. These secondary instances can be cached in a light-weight fashion along with the primary attributes, when the queries are used to render layouts of different focus graphs.

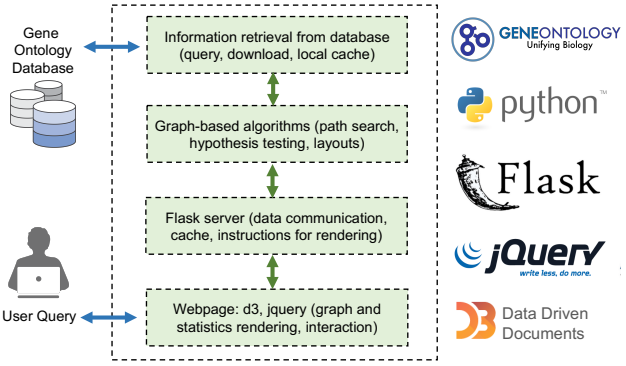

Figure 4: The integrated software system consists of four main components: 1) communication with the gene ontology database; 2) construction of a graph data structure and relevant algorithms; 3) a server that links the Python back-end with the rendering on the front-end web interface; 4) a web interface that renders the layouts, supports animation and interaction.

Within this integrated system, we utilized state-of-the-art Python packages [29] to query and parse GO ontology databases, and incorporate Flask, a Python micro-framework [12] to build the web application. All of the user interactions, (which we will detail in the next section) are programmed using javascript packages: jquery.js and D3.js [5] to support light-weight front-end computations on the web-browser. Because existing packages lack the infrastructure to support multiple contexts and focus graphs, we created our own DAG library to define the data hierarchy and perform the core algorithms.

4 Results

4.1 Front-end Display

We developed a web application: AEGIS (Augmented Visualization of Error-control Methods on Graphs with Interactive Simulations), and discuss its details of the rendered web page in the this section (see Fig. 5 as an initial display). The web page is divided into three sections: a control section, a main display section, and a query section.

The control panel is located on the top of the web page. A user can switch between the "Total" context (i.e., entire GO graph) and the "Statistical" context (the subgraph of tested nodes as explained in Sect. 3.1) by choosing the corresponding radio input. Similarly, a user can select node layout according to depth or height. Additionally, a user can scroll through the simulation bar to select results from different experiments, or outputs from different statistical procedures.

Beneath the control panel is the main display section. It is comprised of an information board and a display canvas. The information board displays GO ID, name and number of genes for the node selected by a mouse. The canvas entails the focus graph on the left and the context graph on the right. The design of visual encodings in the focus and context graphs embody our focus-and-context notions. Focus nodes are highlighted by a different color compared to context nodes that are often muted by light grey. The current queried nodes are highlighted by a yellow border. Tested nodes are colored purple and rejections magenta. The focus graph displays all ancestors and descendants of queried nodes. The context graph displays the summary of each layer in a bar chart. Color encodings are matched in the two graphs.

Finally, the user can input queries in the query bar at bottom of the web page. The query bar supports multiple inputs; it also provides a suggestion list of GO terms that matches current input letters to assist

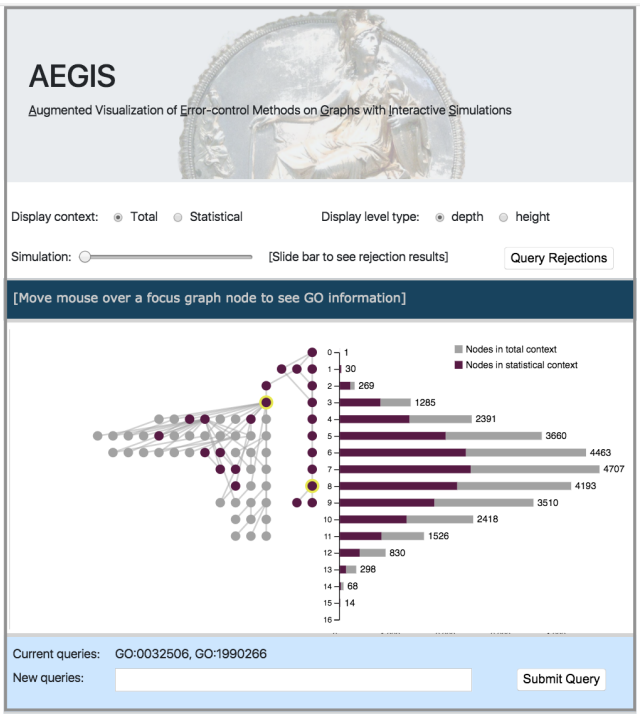

Figure 5: Initial display of AEGIS. The webpage is divided into a control panel, the main display section and a query bar, from top to bottom. In the control panel, a user can switch between contexts as explained in Fig. 2, select node layout and trial number to visualize result. The main display section is composed of a information board that shows GO ID, name of GO term and number of genes in the GO term at current mouse location, and the main display canvas.

user input. Matched nodes are simultaneously colored orange in the display canvas. The query bar is implemented by the "Awesomeplete" widget; and the user can submit queries by clicking the "Submit Query button. Further more, if a user would like to query all the rejected nodes, he can simply click "Query Rejections" button in the control panel before submit, which sets all the current rejections as new focus nodes.

4.2 A case study

We outline a possible data exploration process a user can perform with the application. Upon opening the web page, the user can see a subgraph of all related nodes to the queried nodes, colored in purple if tested and light grey otherwise Fig. 5. In the context graph, the nodes in each layer in the entire GO are colored in light grey, and tested nodes are superimposed as purple bars.

The user can switch to "Statistical" context to visualize results of statistical testing Fig. 6. Upon doing so, nodes not included in tests moves to the right of the graph and fades away, in the order of their layers. They can scroll through the simulation bar to compare among trials, and might notice one child of queried node is rejected several times (colored in magenta). Now, they can move the mouse cursor to the node to look up its information Fig. 7. Upon doing so, the node will be highlighted in dark blue and all related nodes light blue. He can reset focus to this node via the query bar Fig. 8.

4.3 Performance

Despite the size of a DAG with tens of thousands of nodes, our computation framework and system architecture described in Sect. 3.2 offers the advantage of minimal time and space required for interactions with the full-scale data. As a result, our application does

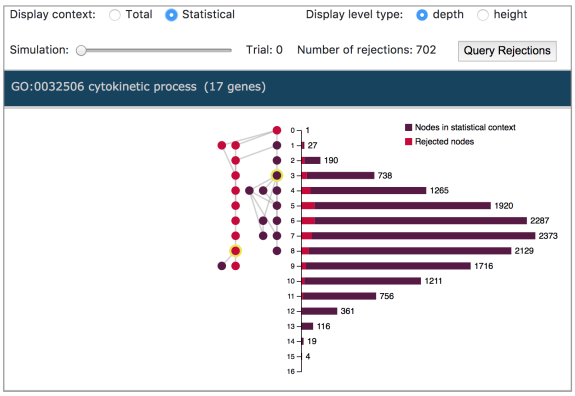

Figure 6: Display under the "Statistical context". Compared to the "Total context", nodes that are not tested are removed in the "Statistical context". Rejected nodes are colored magenta and other tested nodes purple.

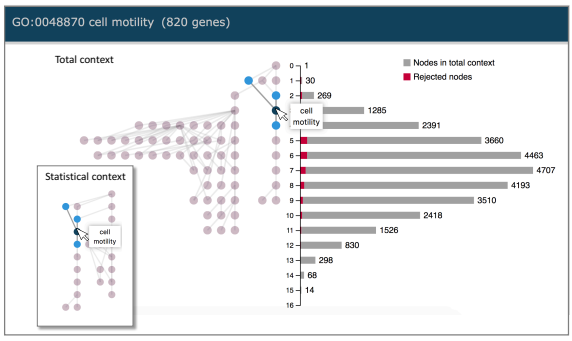

Figure 7: The node under the mouse cursor is highlighted to be dark blue; its parents and children are colored light blue; its relevant information is displayed next to the mouse location as and on the information board. All other nodes and edges are muted. The small capture on the bottom left corresponds to the same interaction under the "Statistical" context.

Figure 8: A user can input queries to reset focus in the query bar. For example, if the user inputs "cell", all GO terms that contain the word "cell" are considered to be matched. The top 10 matches are displayed in a dropdown suggestion list. Both focus and context graphs are updated simultaneously to show matched nodes in orange.

not necessary require expensive resources to perform all the computations in the background, and reduces the need to be hosted as a centralized web portal.

All the development and demonstrations were performed on a MacBook Pro laptop with a 2.8 GHz i7 processor with 8 Virtual Cores and 16GB RAM. The interaction updates in the front-end interface each take less than 0.1 seconds; the focus graph queries (by pressing the "submit query" button) are rendered within 1 second.

It is expected that certain computations in back-end, such as performing a statistical testing method can be more computationally expensive (e.g., up to 5 seconds per experiment or simulation). Nevertheless, this computation does not affect the layout - our hierarchy assigns them to primary attributes, and none of these results need to be recomputed for layout and interaction purposes. Therefore, we were able pre-compute and cache the simulation results prior to running the web server. Another computation bottlenecks in our system is data communication with the GO database and basic file parsing (which can take up to minutes depending on Internet connection), but we have created a light-weight file (of less than 10M) that includes the downloaded information needed for the entire DAG and parsing and allows for statistical testing. Again, we only need to run this computation once for this application, so one can even see it as an installation requirement.

Taken together, these features make it completely feasible to run our application off-line on a personal computer. In contrast, all the displays with interactions for GO analysis introduced in Sect. 2.1 either would require heavy local installations or constant web access.

5 Discussion

We have introduced a novel notion of focus-and-context for data exportation in large DAGs. Our setup is general for large DAGs and supports focused interactions across multiple contexts. The visualization we propose here abstracts the context into a context graph, and adaptively displays the queries in a focus graph. In order to efficiently deploy our visualization principles, we also propose a computation framework which involves data objects with hierarchical attributes.

We showcased our framework with the application of hypothesis testing on a GO DAG, the scale of which can be computationally intensive to display and prohibitive to interpret. We developed a robust back-end library that can handle this real data problem within linear time, and render our focus-and-context representations as a web application: AEGIS. Our software allows a scientist to interpret graphical structures of the significant discoveries the experiments suggest within GO. The scientist can further explore other structures in the GO DAG via our keyword search functionality and generate new hypotheses for the follow-up experiments.

The main positive feedback from scientists we received is that it is valuable to visualize output GO terms that are significant as nodes of an interactive graph instead of just a list of text. The context allowed them to understand how specific or general an output term is with respect to other terms. For those who did not have any prior knowledge about GO or even biology, we found that they were able to understand what the DAG represents, its general structure, and its components using our interactive application. Even though we initially aimed to design an appropriate visualization for domain experts, we learned that our interactive visualization can enable people outside of the field to learn the data and the application better with interactive visual aids.

6 Future Work

It will be valuable to evaluate the focus-and-context framework proposed here in other applications with large DAGs. We anticipate certain graph structures to require further improvement from our current layout: if the graph has many levels and each level contains very few number of nodes, then it might be useful to adapt solutions such as scrolling. We also notice that even for the same data we consider, when displayed in height mode, the number of nodes on the bottom can be exponentially larger than previous layers. This graph structure may require a solution that can be more effective in terms of spacial utilization, or one may consider collapsible nodes to render clearer focus graphs. While collapsible nodes has been shown to be effective in tree layouts, it still remains unclear if the same principles can be applied to certain DAGs as a node can have too many parents to allow it to be collapsed with other nodes. From the GO application point of view, one would currently need to convert data into a Python object to output the results. It will be valuable to provide an end-to-end support where a scientist can input experimental results directly via a web portal and the resulting reports can be generated via our display. Last but not least, a formal survey of how users interactions can be improved can also significantly enhance our design and enable a wider range of users to adopt our application.

Acknowledgments

We would like to thank the CS448B staff and classmates for constructive feedback. We would also like to thank Professor Chiara Sabatti for multiple helpful discussions.

References

[1] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, et al. Gene ontology: tool for the unification of biology. *Nature genetics*, 25(1):25–29, 2000.

[2] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall PTR, 1998.

[3] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the royal statistical society. Series B (Methodological)*, pp. 289–300, 1995.

[4] M. Bogomolov, C. B. Peterson, Y. Benjamini, and C. Sabatti. Testing hypotheses on a tree: new error rates and controlling strategies. *arXiv preprint arXiv:1705.07529*, 2017.

[5] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[6] S. Carbon, A. Ireland, C. J. Mungall, S. Shu, B. Marshall, S. Lewis, A. Hub, and W. P. W. Group. Amigo: online access to ontology and annotation data. *Bioinformatics*, 25(2):288–289, 2008.

[7] S. K. Card and D. Nation. Degree-of-interest trees: A component of an attention-reactive user interface. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pp. 231–245. ACM, 2002.

[8] M. S. T. Carpendale and C. Montagnese. A framework for unifying presentation space. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pp. 61–70. ACM, 2001.

[9] G. O. Consortium et al. Expansion of the gene ontology knowledgebase and resources. *Nucleic acids research*, 45(D1):D331–D338, 2017.

[10] P. Eades and L. Xuemin. How to draw a directed graph. In *Visual Languages, 1989., IEEE Workshop on*, pp. 13–17. IEEE, 1989.

[11] J. J. Goeman and U. Mansmann. Multiple testing on the directed acyclic graph of gene ontology. *Bioinformatics*, 24(4):537–544, 2008.

[12] M. Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2014.

[13] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on visualization and computer graphics*, 6(1):24–43, 2000.

[14] Y. Hochberg. A sharper bonferroni procedure for multiple tests of significance. *Biometrika*, 75(4):800–802, 1988.

[15] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pp. 65–70, 1979.

[16] Y. K. Leung and M. D. Apperley. A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 1(2):126–160, 1994.

[17] T. Munzner. Interactive visualization of large graphs and networks, dhd thesis, stanford university, 2000.

[18] C. Pesquita, D. Faria, A. O. Falcao, P. Lord, and F. M. Couto. Semantic similarity in biomedical ontologies. *PLoS computational biology*, 5(7):e1000443, 2009.

[19] H. Purchase. Which aesthetic has the greatest effect on human understanding? In *International Symposium on Graph Drawing*, pp. 248–261. Springer, 1997.

[20] H. C. Purchase, R. F. Cohen, and M. James. Validating graph drawing aesthetics. In *International Symposium on Graph Drawing*, pp. 435–446. Springer, 1995.

[21] A. Ramdas, J. Chen, M. J. Wainwright, and M. I. Jordan. Dagger: A sequential algorithm for fdr control on dags. *arXiv preprint arXiv:1709.10250*, 2017.

[22] R. Rao and S. K. Card. The table lens: merging graphical and symbolic representations in an interactive focus+ context visualization for tabular information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 318–322. ACM, 1994.

[23] E. M. Reingold and J. S. Tilford. Tidier drawings of trees. *IEEE Transactions on Software Engineering*, (2):223–228, 1981.

[24] T. Schweder and E. Spjøtvoll. Plots of p-values to evaluate many tests simultaneously. *Biometrika*, 69(3):493–502, 1982.

[25] R. S. Sealfon, M. A. Hibbs, C. Huttenhower, C. L. Myers, and O. G. Troyanskaya. Golem: an interactive graph-based gene-ontology navigation and analysis tool. *BMC bioinformatics*, 7(1):443, 2006.

[26] B. Sorić. Statistical discoveries and effect-size estimation. *Journal of the American Statistical Association*, 84(406):608–610, 1989.

[27] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.

[28] F. Supek, M. Bošnjak, N. Škunca, and T. Šmuc. Revigo summarizes and visualizes long lists of gene ontology terms. *PloS one*, 6(7):e21800, 2011.

[29] H. Tang, D. Klopfenstein, B. Pedersen, P. Flick, K. Sato, F. Ramirez, J. Yunes, and C. Mungall. Goatools: tools for gene ontology. *Zenodo.*, 2015.

[30] J. Q. Walker. A node-positioning algorithm for general trees. *Software: Practice and Experience*, 20(7):685–705, 1990.