

A thick dark blue vertical bar runs along the left edge of the page. A blue arrow-shaped banner points to the right from this bar, containing the date '19-6-2018'. In the bottom-left corner, several thin, curved lines in dark blue and light grey sweep upwards and to the right.

19-6-2018

Interactive visualization and comparison of protein function prediction models

Bpsys

Supervisors: Thomas Abeel, Stavros Makrodimitris
& Jacob Gubbens

Sven Bijleveld, Karolis Cremers, Jasper van Dalum
& Bas Remmelswaal,

HOGESCHOOL LEIDEN – TU DELFT

Inhoud

Abstract	2
Introduction.....	3
Project Goal	3
Materials and Methods	4
Dependency installation script.....	4
Start script	4
Input screen.....	5
Input files.....	5
Parser.....	5
Attribute assignment.....	5
Assigning Terms to layer.....	6
Pruning the DAG	6
Pickling.....	7
X positions algorithm.....	7
Term Order Within Layers	7
Sorting a Layer Based on Barycenters	7
2-level Barycentric Method	7
n-Level Barycentric Method	8
Horizontal positions of terms	8
Visualization	9
Performance testing.....	10
Results	11
Input screen.....	11
Visualization	11
Visualizing one model.....	11
Visualizing two models	12
Interactive features	14
Search feature	15
Conclusion and discussion.....	17
References.....	19
Appendixes	20

Abstract

Gene Ontology (GO) terms describe protein functions.^[1] These terms are organized in a Directed Acyclic Graph (DAG) hierarchy. Models can be created to annotate proteins by assigning these terms to proteins. Comparing and evaluating these models is difficult because there are more than 44.000 terms.^[3] It is important to choose the right model to annotate the genes of interest. A model can have an overall high score but if there is a model with a lower overall score but a high score for the genes of interest, this model should be used to annotate the genes. An interactive visualization and comparison tool for GO annotation models has been created to make it possible to make it less difficult to select the right model. This tool is able to visualize (a part of) the GO tree and show the differences between the models on a global and local level.

Introduction

Proteins are large biological molecules responsible for a lot of vital functions in organisms. Some of the roles that these proteins carry out are for example structural support, signalling, regulation, storage, catalytic activity and many more.^[1] Due to the interconnective nature of these functions it becomes confusing very quickly. To counter this problem the Gene Ontology (GO) project aims to make a systematic overview of these functions.

The GO project provides a logical structure for biological processes, cellular components and molecular functions. Each gene function has its own term. These terms can be assigned to a gene with the function of the term. All these terms and groups form a Directed Acyclic Graph (DAG). This graph has a non-cyclic structure; therefore, when following a path to a node from the root, you will never visit the same node twice. Nodes in this structure are GO terms, edges of these nodes are used to connect the nodes to each other. GO terms are constantly revised and added as biological knowledge increases.^[2]

Annotation of proteins can be done by predicting which GO terms fit the function of the protein and assigning these to the protein. Due to the exponentially increasing number of unannotated genes and the lack of manpower to solve this issue by experimentally annotating these genes, predictive algorithms are being developed to assign GO terms to these genes. One of the problems with evaluating the performance of these predictive algorithms is the amount of GO terms they can predict (more than 40.000) and the possible difference in performance of these algorithms between functions or groups of functions.^[3]

It is important to use the right model for the right tasks. Some models could work well for plant related genes but work poorly for all the other genes while other models might work better overall but worse for plant related genes. Current visualization programs do not have the capacity to visualize the difference in performance on a global and local level of models using the GO tree.

There are a lot of tools available on the internet to visualize the GO tree. GO feat, NaviGO and Quick GO are some tools that are able to do this. The downside of these tools is that they do not offer the ability to compare models and show which model works better in a specific area.^[4-6]

Project Goal

The goal of this project is to create a tool that is able to visualize a user specified part of the GO tree and compare the scores of two models on a global and local level. There should also be a filter and zoom function to make the visualization interactive.

Materials and Methods

Dependency installation script

A bash script has been made to install all the dependencies listed in table 1. After running this script, the user is able to execute the pipeline script run.sh.

Table 1: The versions of modules and programming languages used by the application

Module	Version	Build
Python	3.6.3	hc9025b9_1
Bokeh	0.12.16	Py36
Tornado	4.4.3	Py27_0
colorcet	1.0.0	Py27h2c72f91_0
Numpy	1.14.3	py36h0ea5e3f_1
Pandas	0.22.0	py36_0
Numba	0.37.0	Np113py36ha6e7140_0
Scipy	0.19.1	Np113py36_0
cvxopt	1.1.8	Py36_4
Java	1.8.0_121	jdk1.8.0_172
JavaFX	8.0	8.0.172-b11

Start script

The script run.sh executes all the scripts in the right order. It also makes sure that the correct parameters are used. This script calls the scripts in the order shown in figure 1.

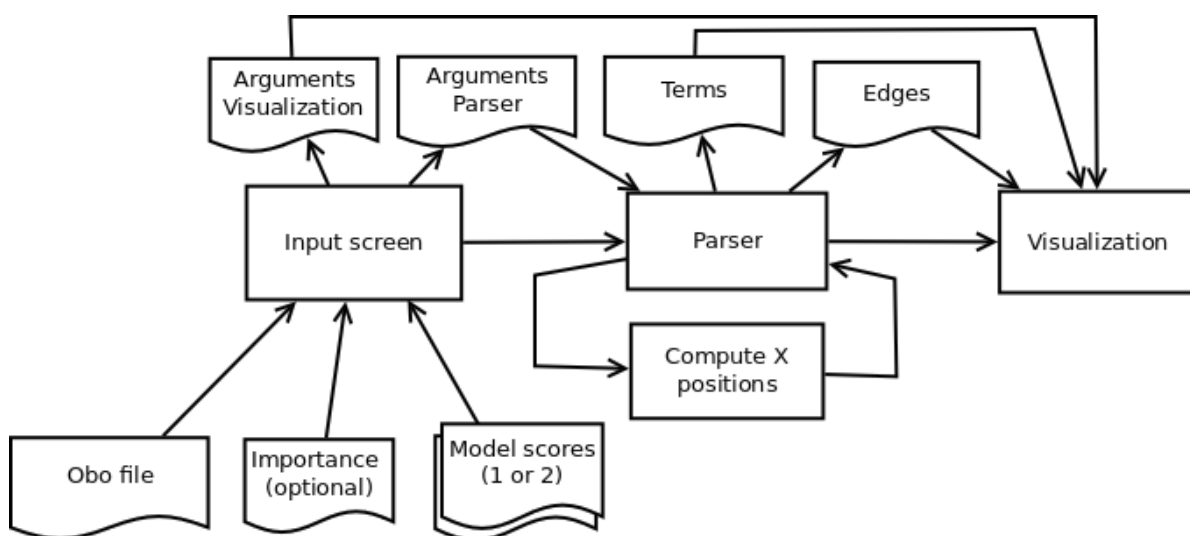


Figure 1: Flow of the visualization pipeline.

The first script that is called is the input screen. The input screen asks the user to supply the data that should be visualized. The user input is saved to two files as input for the parser and visualization script.

The second script is the parser. This script reads the obo file and converts this structure to objects. After this it determines which layer the terms should be in with the longest path algorithm. The X position algorithm is called after this step. The data that comes back from this algorithm is merged with the already present data. Two data frames are made, the first data frame contains the terms and the second data frame contains the edges. These are saved as pickle files.

The third script that is called is the X positions algorithm. This algorithm first sorts the terms on each layer in a way that it causes a minimum amount of edge crossings. Dummy nodes are added to further enhance this. After sorting actual X positions are calculated with a custom implementation of the priority layout method. The results are returned to the parse script.

The last script that is called is the visualization script. This script uses the generated files to visualize the input files selected by the user. The terms and edges are extracted from these files and visualized.

Input screen

The input screen is the first script that is called by the start script. This script has been written in Java and opens a JavaFX application. It contains a file selection widget for the obo and the importance file and one or two file selection widget(s) for the model(s). There is also a checkbox with an option to remove all the nodes that do not have a score. When all the mandatory files are selected by the user, the "Visualize" button can be used. This button saves the selected filenames to two files. One file is used as the input parameters for the parser. The second file is used as the input parameters of the visualizer.

Input files

The user is able to select three kinds of input files. The first file is an obo file. An obo file can be downloaded from Amigo2 and contains the basic structure of the GO tree. There is an obo file available with the whole GO tree, but there are also subsets of the GO tree. These subsets are called slim sets and contain the terms that are useful for annotating a specific organism.

The second file is the score file. This file is tab-separated and contains the term name in the first column and the accuracy score from the model in the second column. This accuracy in the example data was based on an AUC ROC curve, but the scores are not required to be computed in this way.

The third file is the importance file. This file is optional and contains the importance of terms in the same order as the accuracy score file. The term importance used in the application is analogue to information content of a term. The information content of a Term can be calculated using the following function from Wyatt T. Clark and Predrag Radivojac.^[7] $i(t) = \log \frac{1}{\text{Pr}(T)}$ Where $i(T)$ is the information content of a term and $\text{Pr}(T)$ the probability of a protein is experimentally associated with a consistent subgraph T in the ontology. $\text{Pr}(T)$ can be calculated using the following formula: $\text{Pr}(T) = \prod_{v \in T} \text{Pr}(v|p(v))$ where v denotes a term in a graph and $p(v)$ is the set of parent terms of v .

Parser

Attribute assignment

Term objects are assigned attributes based on the obo, scores and importance files are stored in a dictionary. Then the parents attribute is used to log the connections as edges. Finally, the terms are assigned to layers.

Assigning Terms to layer

Assigning terms to the correct layer can be done by using a List Scheduling Algorithm. A List Scheduling Algorithm solves a multiprocessor scheduling problem by scheduling a number of causally related tasks on processors. This minimises the completion time of the calculation.

The longest path algorithm has been used to assign terms to the correct layer. This algorithm tries to find a path to the root with the maximum length. The longest path algorithm assigns a lower priority to terms with a high number of ancestors. These priorities are used as the layers in a DAG graph. All terms with the same priority are seen as one layer. A longest path algorithm described by *Healy, Patrick* and *Nikolov, Nikola S.* has been used to assign layering.^[8] This algorithm assigns terms to a layer directly when it is given a priority. This algorithm has been additionally modified to prioritise the terms with the lowest number of ancestors to improve space efficiency in the visualized graph. Pseudocode of the algorithm can be found in appendix 1. A basic workflow of the algorithm can be seen in figure 2.

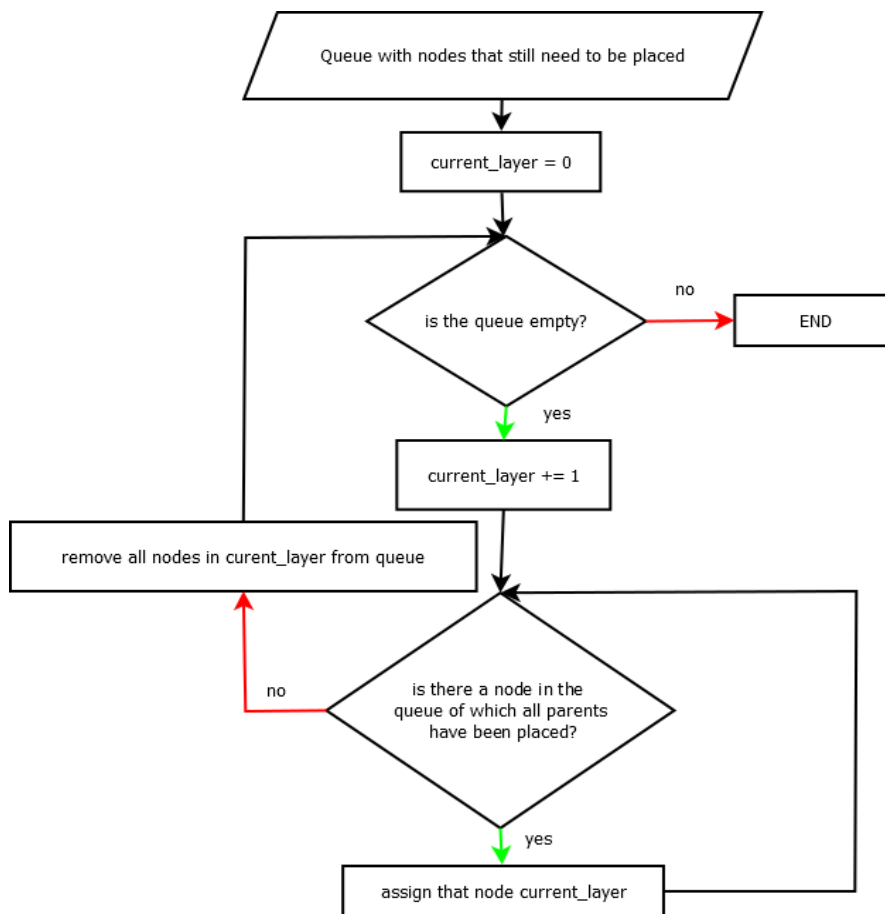


Figure 2: The workflow of the longest-path algorithm: First, a queue is made containing all terms which have not been placed. Every term in the queue is checked if it can be placed by checking if all its parents are placed. The current layer count starts at 0 and is increased if all terms that could be placed have been placed.

Pruning the DAG

When the option "Remove nodes without score" has been selected on the input screen, any term not having an accuracy score in the first score file is deleted. This makes the visualization process a lot faster.

Pickling

Finally, when all algorithms have run, the term and edges are converted to data frames and saved in pickled files which are given to the visualization script.

X positions algorithm

The X positions algorithm determines the X positions of the terms. This is done by sorting the terms in a layer so the terms are in the right order. X values are bound to the term with the priority layout method. When this process has finished the computed X positions are returned to the parser.

Term Order Within Layers

To improve the readability of the layered graph the number of crossings between edges needs to be reduced. From several ordering methods, the barycentric method was selected because of its relatively simple implementation and because it is relatively fast. It orders the layer in $O(|V_i| \log |V_i|)$ where V is the number of vertices in the layer.^[8, 9]

Sorting a Layer Based on Barycenters

In the barycentric method, a procedure that gets used a lot is the sorting of a layer, based on barycenters. The barycenter of a term is the average position of all terms above or below that term. These barycenters are calculated for one layer based on either the children or parents and sorted to try and reduce the number of edge crossings. This procedure can be seen in figure 3.

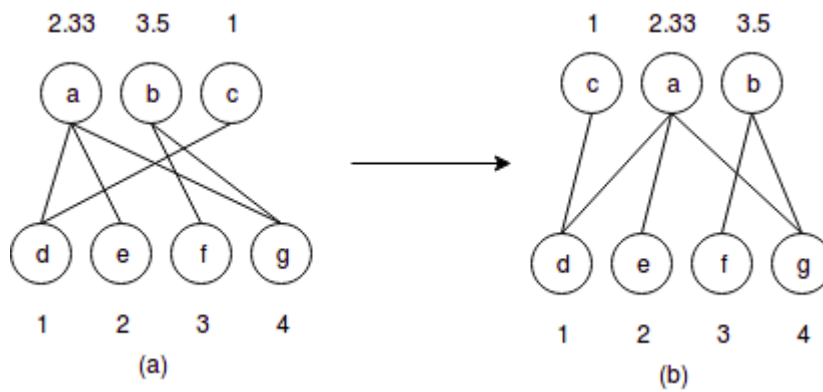


Figure 3: Sorting a layer based on barycenters in this case the upper layer is sorted based on the positions of the children. Values at the top indicate the barycenters of upper terms, values at the bottom indicate positions of the lower terms. (a) before ordering, (b) after ordering

2-level Barycentric Method

First, we will consider the barycentric method for solving a layered DAG with two layers. The method consists of two phases, phase 1 which will order the terms based on their barycenters and phase 2 where the order of terms with the same barycenter is reversed to try and obtain a better solution, phase 2 will use phase 1 as a subroutine.

Phase 1

In the 2 layer case each layer has only one layer to base their positions on. Because of this the layers can be sorted in turn.

The order in which this process is done is not relevant for this step but it is going to be relevant for phase 2. If the upper layer is sorted first (where all positions are based on the children) it is called up-down (the direction indicates the direction of the edges which it is sorted by) whereas if it's done the other way around it is called down-up.

In this phase the up-down (or down-up) process is repeated until the same configuration of terms arises again or all the layers are already sorted based on their barycenters in which case phase 1 is irrelevant. During this process the same configuration of terms usually appears after two iterations

Phase 2

In this phase the terms in the same layer that have the same barycenters will be reversed. This phase is divided into two sub phases: An up and a down phase where barycenters based on different connections are considered. In the up phase the barycenters that are based on the parent connections that have the same values are reversed. Then phase 1 is repeated up-down to first sort the layer where the barycenters have changed by the reversal. For the down phase barycenters based on the children are reversed and phase 1 is executed down-up.

n-Level Barycentric Method

For a n-layered DAG much of the same concepts for ordering a 2 layered graph are used but changed slightly.

Phase 1

For the n-layered DAG the 2-layer phase 1 cannot just be repeated down or upward because this only considers two layers at a time without accounting for these changes in the layers above (or below). To consider all layers both the up and down phases will go through all layers. The down phase now loops through layers 2 ... n sorting all layers based on their children and the up-phase loops through the layers n+1 ... 1 sorting based on the parents. This process is repeated until the same configuration appears or all terms are ordered by barycenters

Phase 2

Phase 2 is almost exactly the same for the n layer case, the only difference is that all the barycenters in a certain direction are changed. After the down (or up) phase, phase 1 is executed in the up-down (or down-up) procedure.

Horizontal positions of terms

The last step of the X position algorithm calculates the horizontal positions of the terms. These calculations can be done with a theoretical or a heuristic method. The theoretical method uses Quadratic Programming(QP) to calculate the horizontal position with the following minimization

formula $f = cf_1 + (-c)f_2$. The heuristic method uses Priority list to calculate the horizontal position, this non-optimal method has a reduced computing cost compared to the QP method. Therefore, the heuristic method was chosen to calculate the horizontal positions.

Priority method

Calculating the horizontal positions with the priority method can be divided in four steps. The first step gives each term an initial position per layer by aligning all the terms to the left. The gap between each term is 1.

The second step is to determine the layering order by three phases: Down, up and final. During the down phase the layers are ordered from top to bottom; During the up phase the layers are ordered from bottom to top; During the final phase the layers are ordered starting at the layer with the maximum number of terms and ends at the bottom.

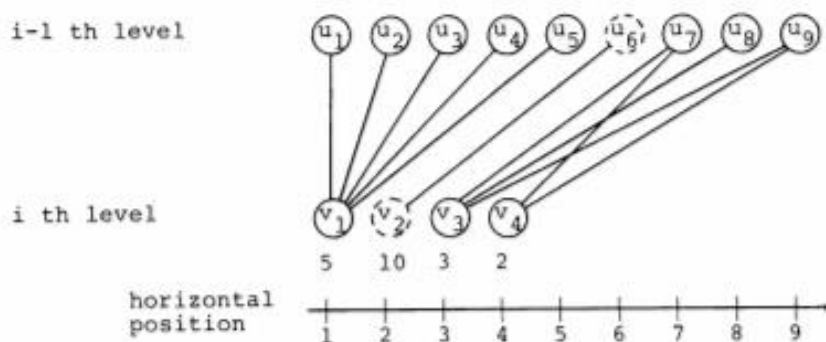


Figure 4: An example of two layers with terms and dummy nodes. The terms with a dashed circle are dummy nodes and the normal circles are terms. In the i th layer there is also a given priority. Source: *Handbook of Graph Drawing and Visualization; Chapter 13 p412*

The third step is to give each term a priority. The priority is calculated by counting the number of connections with its children when going up and with its parents when going down. To clarify this, see term V_1 in figure 4. You can see that V_1 has a priority of 5, because it has 5 connections to its parents. The priority of the terms V_3 and V_4 in figure 4 are calculated the same way. However, the term V_2 in figure 4 is an exception because the connection of V_2 is a dummy to dummy connection. Therefore V_2 gets a higher priority than any other term with different connections, so in this case it will be higher than 5. This exception exists to maintain the readability in the GO tree for edges that go through multiple layers.

the last step is to improve the position by the priority list and ordering layers so the distance between the upper and lower barycenters are minimized. This improvement has these three restrictions:

- The first restriction ensures that the positions are integers and the values of the position are not the same in any other position of the same layer.
- The second restriction ensures the order that was created by the barycentric method will be maintained. This will reduce the crossing between vertices and will improve the readability of the GO tree.
- The last restriction ensures that if a term is placed, its position cannot be changed. The only terms which the position can change of, are the terms with a lower priority than the term to be placed. The only restriction to these changes is that they are as small as possible.

To calculate the X positions with the priority layout method, a quicksort like method was implemented to solve the problem. It takes the first item from the priority list and tries to place it. Now the problem is divided into two parts: the space to the left of the node that was just placed and the space to the right, the queue is split up and the same process is repeated.

Visualization

A Bokeh server has been used to visualize the data.^[10] The Python script can be called with two parameters. The first parameter must be specified; This is the number of models that the user wants to visualize. The second parameter should only be specified if an importance file is available. These parameters are retrieved from the parameter file made by the input screen script.

After reading the parameters the files produced by the parser are read with Pandas which produces two data frames. These data frames are converted to a column data source which enables Bokeh to visualize the data in an interactive manner.

The main visualization consists of terms (nodes) and edges (connections). The edges are drawn with the multiLine glyph where the first and last x/y values are the location of the terms that it connects and the other values are dummy nodes. The terms consist of one or two VBar glyphs, this depends on the number of models. The VBars are given a colour from a grayscale colour mapper, this colour depends on the corresponding score. The application also draws a third set of VBar glyphs when two models are visualized. These glyphs correspond to the difference between the scores from the first and second model. These nodes are only shown if the user presses a button to show the differences between models.

Some tools are created that can be used to interact with the visualization. Zoom tools, hover tools and tap tools are added and configured to work with this type of data. Most of these tools use the default behaviour, some tools are modified. The first tap tool is adjusted so it opens the Amigo2 page for a clicked term. The second tap tool is adjusted so it shows all its descendants, ancestors and the path between those terms. Two recursive functions are used to find these terms. All the selected terms are added to their respective textboxes underneath the graph.

On the right side of the graph a table is added with all the drawn terms. It contains columns with the term name, scores, difference and importance. Terms that are selected in the graph will also be selected in the table.

Underneath the table a search function is added. This search function filters the term name, term description, score and importance column when the search button is pressed. Values that do not match the search query are deselected in the term data frame.

If an importance file is added the application creates two statistics plots. The first plot contains the model(s) and difference plotted against the importance. A smoothing line is added which is determined with the Savitzky-Golay filter.^[11] An interactive legend is added to this plot. When one of the models in the legend is unselected, it hides all the terms. The second statistics plot is always added. This plot contains a boxplot where the distance from the root node is plotted against the accuracy score.

Performance testing

To test the performance of the application a small dataset with 1724 terms and the biological process subtree (28.984) were visualized. The time for each step to complete was measured.

Results

Input screen

A user-friendly input screen has been created for the user of the tool to provide the necessary files. The user can choose if one or two models should be visualized. If the user chooses to visualize only one model, the input field for the second model disappears. There is also the filter option "Remove terms without score". This option removes the terms if the model did not predict the term. This makes the visualization process a lot faster if the obo file is big and the amount of predicted terms small. The user can click on the visualize button when the mandatory files are provided. A screenshot of the input screen can be found in figure 5.

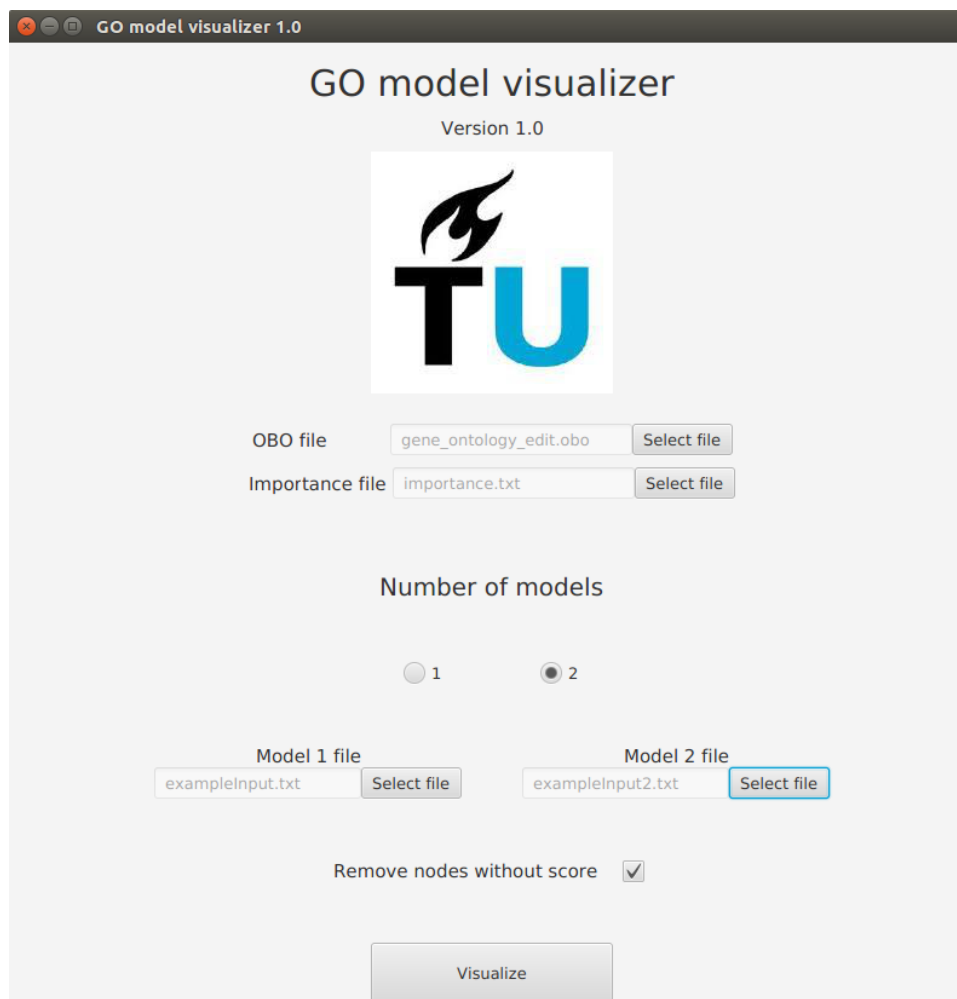


Figure 5: Input screen where the user can supply the files that should be visualized.

Visualization

After clicking on the visualize button the visualization process starts. The tool reads the input files and computes the locations for the terms and edges. The term and edge locations are saved in two files and loaded by the visualization application.

Visualizing one model

The visualization of one model can be seen in figure 6. Terms are visualized as rectangles. The prediction accuracy is high when the bar is black and low when the bar is white. The lines between the terms are the edges. The visualization with two models contains more options: The prediction accuracy difference can also be visualized.

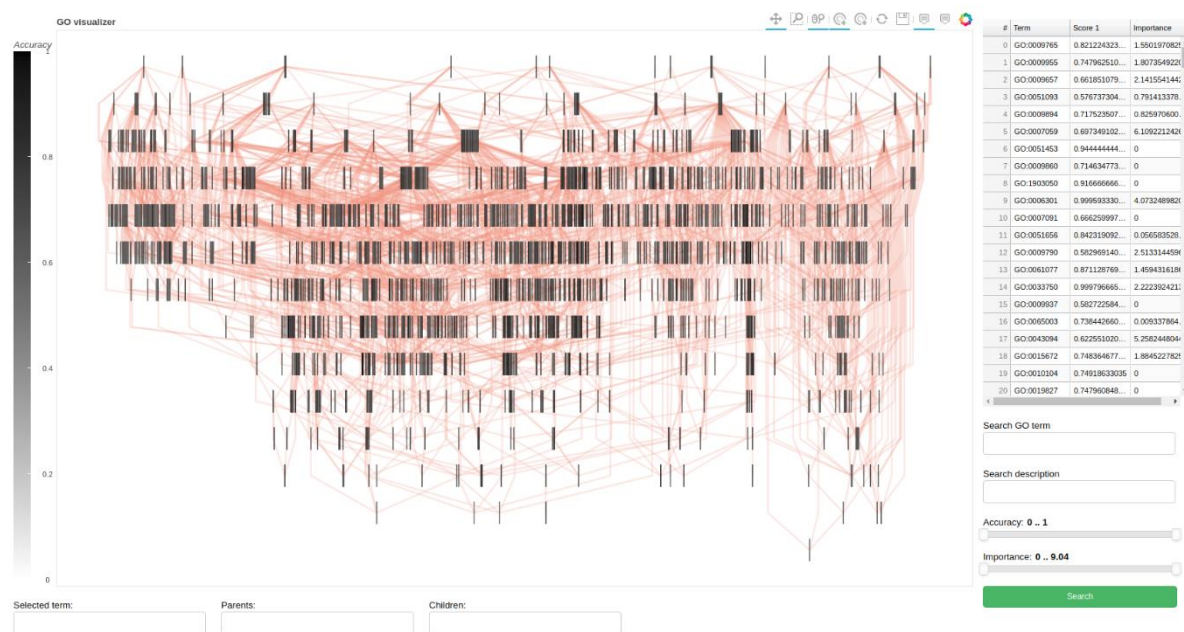


Figure 6: Visualization of one model.

Under the GO tree visualization two statistical plots are shown. These can be seen in figure 7. The plot on the left contains the information content plotted against the prediction performance with a regression line. This plot will not be shown if no importance file has been provided. On the right is a boxplot with the GO layer plotted against the prediction performance. The GO layer corresponds to the distance to the root. The orange circles are the outliers.

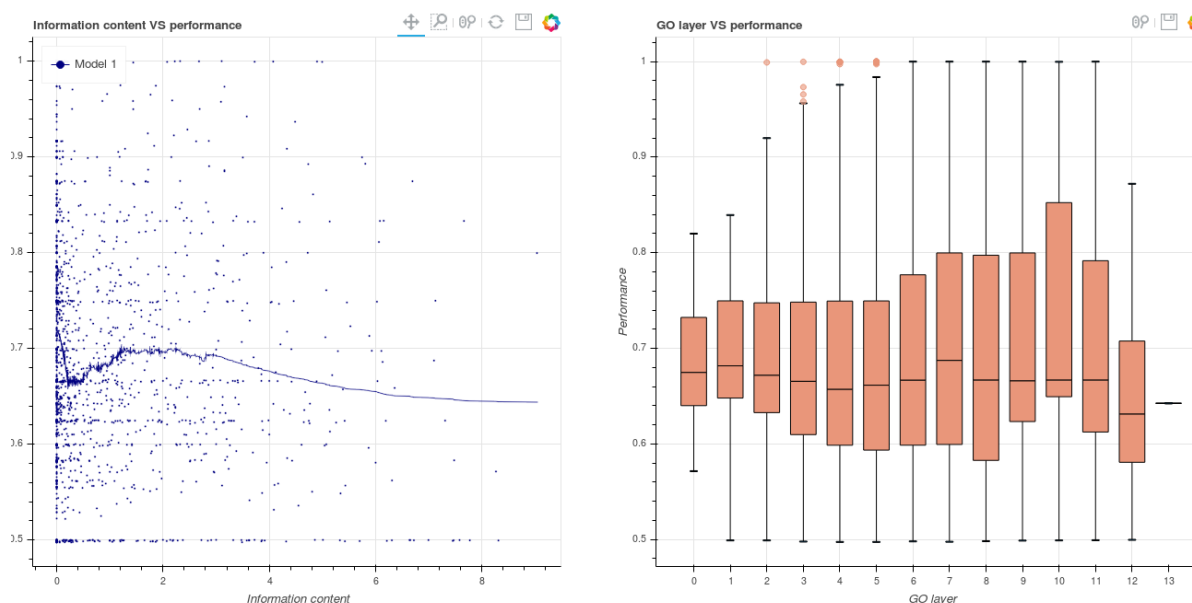


Figure 7: Statistics plots of a visualization with one model. Left: Information content VS accuracy score. Right: Boxplot of the layer distance from the root VS the accuracy score.

Visualizing two models

The visualization of two models can be seen in figure 8. Terms are visualized as rectangles. The prediction accuracy is high when the bar is black and low when the bar is white. The upper part of the rectangle represents the score of the first model. The lower part of the rectangle represents the score of

the second model. The lines between the terms are the edges. The terms are shown in more detail in figure 9.

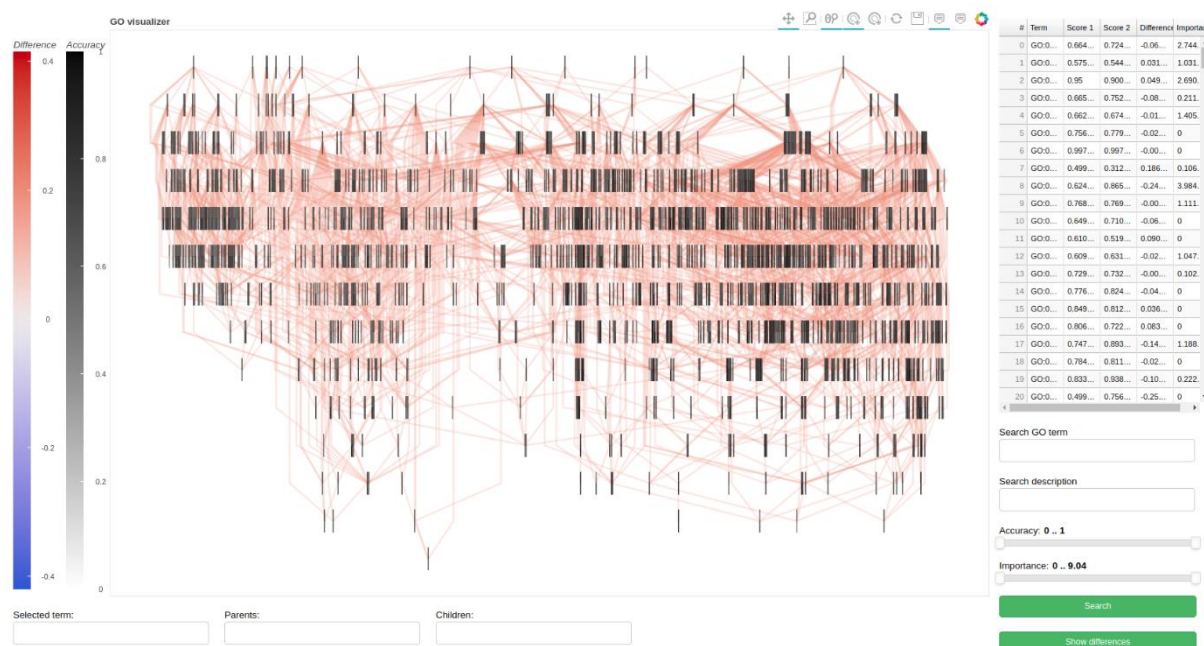


Figure 8: Visualization of two models.

When the user clicks on the "show differences" button a new colour scale will be shown. The first model's prediction accuracy is higher when the term is red. The second model's prediction accuracy is higher when the term is blue. The terms are roughly equal when the term is white. A demonstration of this feature can be seen in figure 9. On the left side are the terms with the two prediction accuracy scores, on the right are the differences between the prediction accuracy scores.

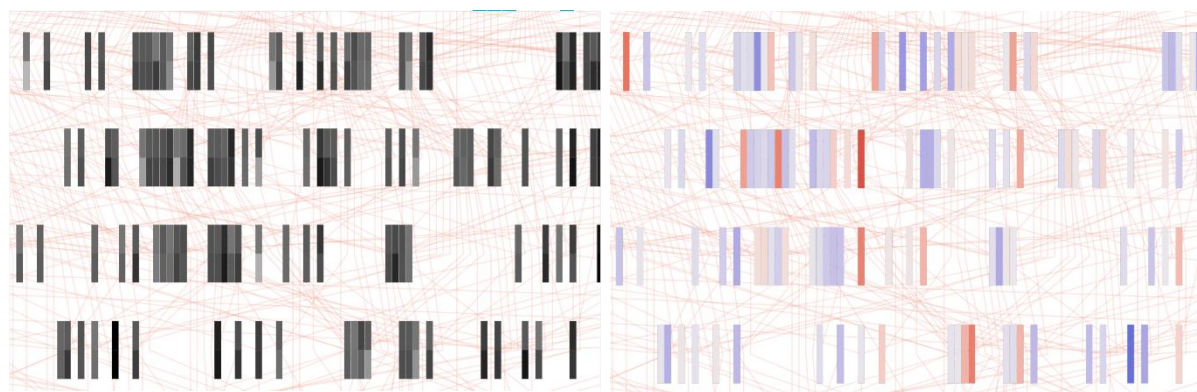


Figure 9: Left: Two terms prediction score where the upper part is model one and the lower part is model two. Right: The difference between model 1 and model 2 where red means score1 > score2 and blue means score1 < score2.

Under the GO tree visualization two statistical plots are shown. These can be seen in figure 10. The plot on the left contains the information content plotted against the prediction performance with a regression line. The red and blue glyphs represent two models, the green glyphs represent the difference between these models. This plot will not be shown if no importance file has been provided. The legend is interactive, data points for one group will be hidden when the group is unselected in the legend. On the right is a boxplot with the GO layer plotted against the difference in prediction performance. The orange circles are the outliers.

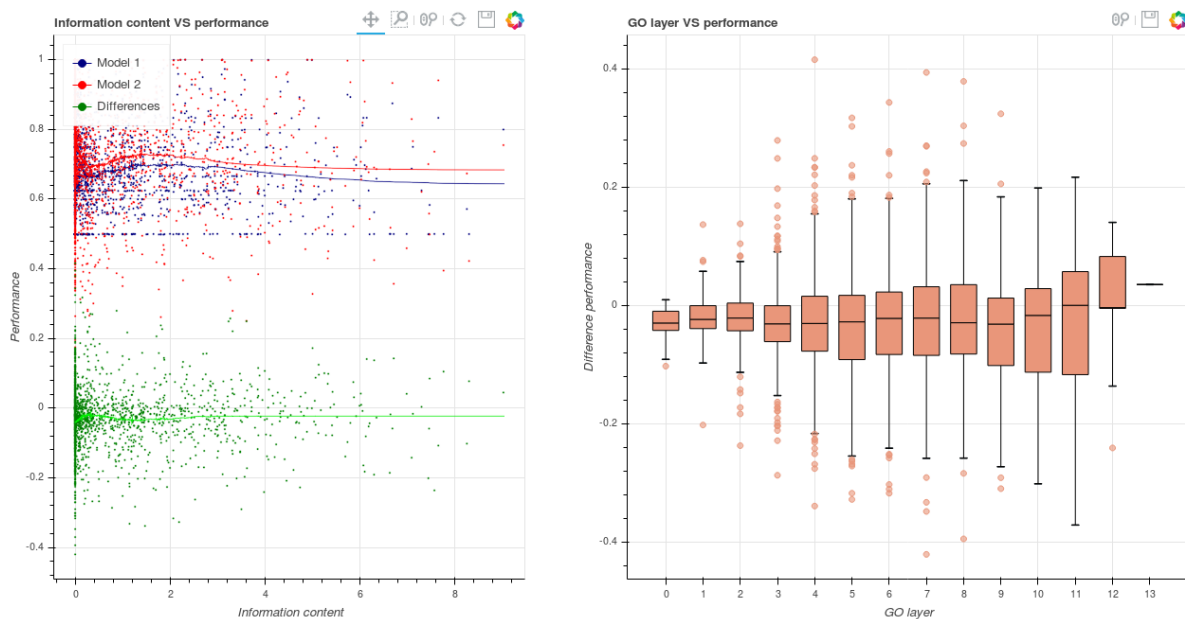


Figure 10: Statistics plots of a visualization with two models. Left: Information content VS accuracy scores and difference between the scores. Right: Boxplot of the layer distance from the root VS the difference between the score from model 1 and 2.

Interactive features

On the right-top side of the GO tree visualization is a toolbar with tools which make the visualization interactive. These tools can be seen in figure 11. These features will be explained from left to right.



Figure 11: Tools used in the visualization.

The first tool is the drag tool. The drag tool allows the user to drag the contents of the plot around.

The second tool is the box zoom. The box zoom tool allows the user to define a rectangular selection by left dragging the mouse. The zoom will adjust to the rectangular selection.

The third tool is the X-wheel zoom. This tool allows the user to zoom on the X-axis. This zoom improves the amount of visible terms. A normal zoom would also zoom on the Y-axis where it is not really necessary.

The fourth tool is the first tap tool. This tap tool allows the user to click on a term. Another web browser tab opens with the Amigo2 page for the clicked term.

The fifth tool is the second tap tool. This tap tool allows the user to click on a term. The descendants and ancestors and their edges will be highlighted. The selected term will be highlighted with a red transparent rectangle. The names of the selected term, ancestor terms and descendant terms are also shown in their respective text fields. This can be seen in figure 12.

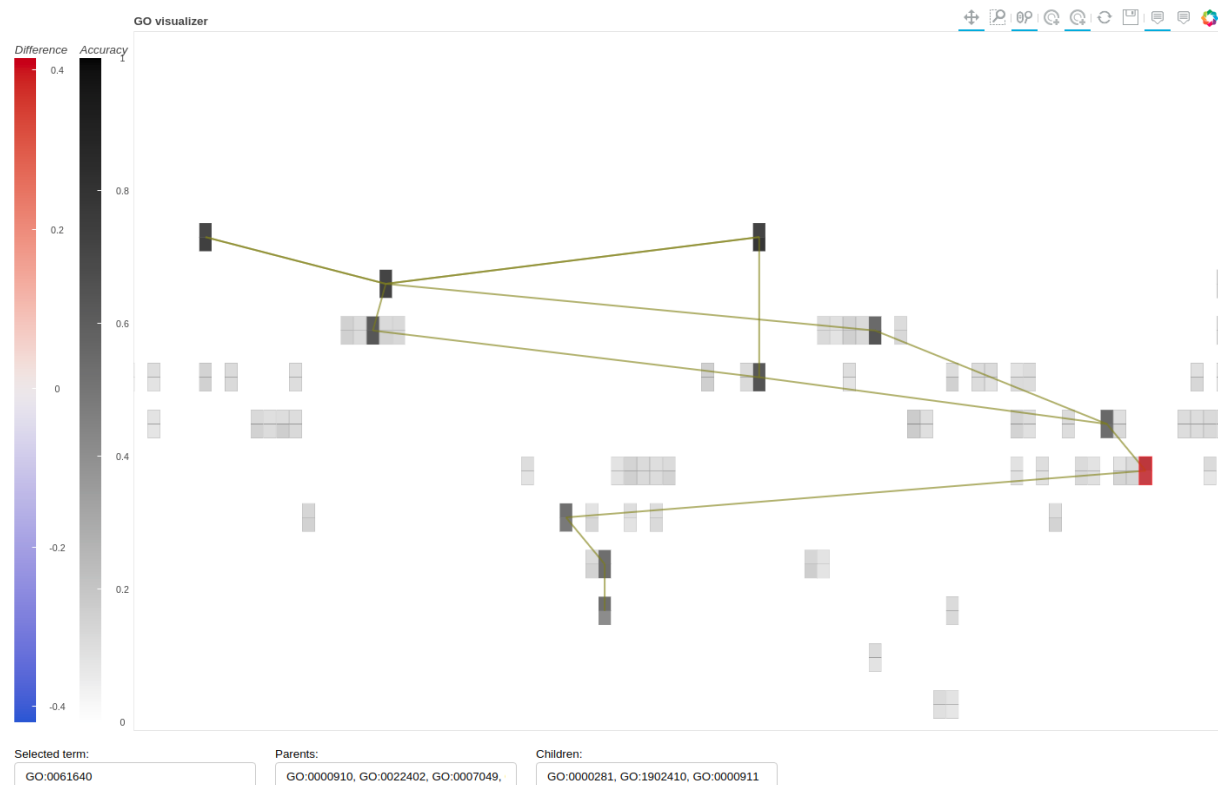


Figure 12: Demonstration of the second tap tool. This tool selects the selected term and its parents and children. All the terms are also shown in their respective text field. The selected term is highlighted in red.

The sixth tool is the reset tool. This resets the plot to the state it was in when the application is loaded.

The seventh tool is the save tool. This tool saves an image of the currently visible terms as a .png file.

The last two tools are hover tools. The first hover tool is for the terms, the second hover tool is for the edges. These tools display extra information about a term or edge when the cursor is above it. This can be seen in figure 13 with on the left side the result of a hover on edge, and on the right side a hover on term.

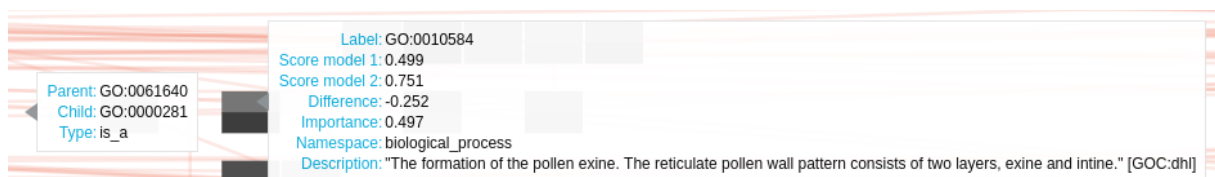


Figure 13: Demonstration of the hover tools. Left: Result of hovering above an edge. Right: Result of hovering above a term.

Search feature

A search feature has been added to make it easy to find a (set of) specific term(s). When a term matches the search query it will be selected. All selected terms are added to the selected term text field and highlighted in the table. The user is able to filter based on the name of the GO term, description, namespace, score and importance. Search terms can be split by putting a "," between the terms. Accuracy and importance can be filtered between an upper and a lower value. The search query is executed when the user clicks on the "Search" button. All the terms in the information content VS

performance plot that match the search query are also selected. A demonstration of the search feature can be seen in figure 14.

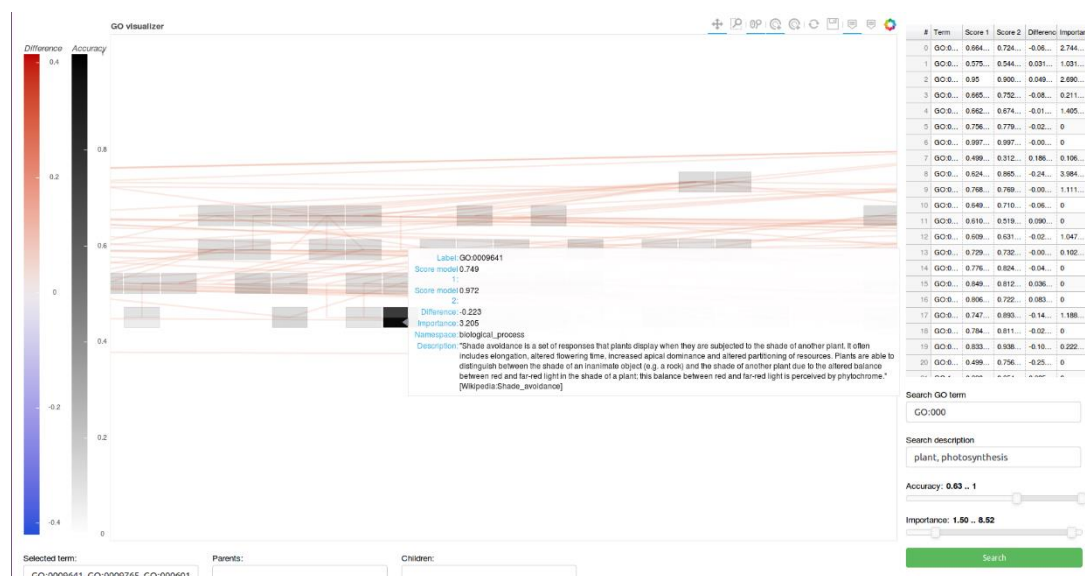


Figure 14: Demonstration of the search function.

Conclusion and discussion

A tool has been created that visualizes performances of one or two models. With this tool the user is able to compare two prediction models on a global and local level. Results can be easily analysed with the interactive features. The tool works well but there are always things that could be improved.

When the obo file is big and uncut, the connection to the Bokeh server is lost because the maximum message size of the web socket has been exceeded. This can be resolved by increasing the maximum web socket message size in Bokeh's networking library.

The application is also quite slow when a lot of terms are plotted and the edges cannot be visualized in an orderly fashion because there are too much edges. To improve this, Datashader could be used. Datashader is able to bundle edges and show more details after zooming in. An example of the results of using edge bundling on a graph can be seen in figure 15.^[12]

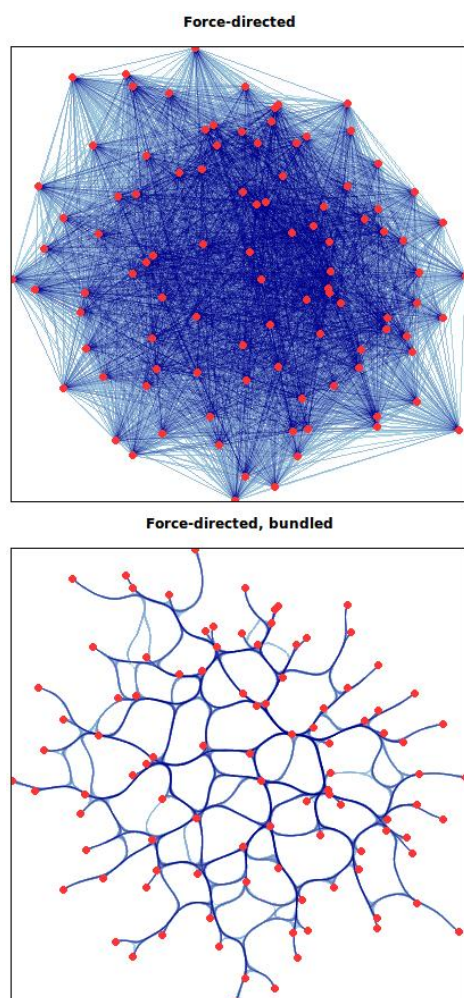


Figure 15: The result of using datashader on a force directed graph. Source: http://datashader.org/user_guide/7_Networks.html

The performance of the application has been measured with a small set of 1724 terms and the biological process subtree which contains 28.984 terms. The parser and visualization part of the pipeline are fast and the time did not increase by a lot when the dataset is bigger. This is not the case for the sorting of the terms per layer and computing the X positions. All the timing results can be found in table 2. When even more terms are visualized, computing the X positions will take even more time than the barycentric ordering.

Table 2: Performance of the pipeline with a small and a big dataset.

	1724 terms	28.984 terms
Parsing	2 seconds	2 seconds
Barycentric ordering	9,8 seconds	3014 seconds
Computing X positions	1,2 seconds	1057 seconds
Visualization	9,3 seconds	24 seconds
Total	22,3 seconds	4097 seconds -> 1 hour, 8 minutes

The application is currently unable to check if the input files are in the right format. This feature could be added in the future.

Finally the edge placement is not optimal. This is because the algorithm does not compute the perfect solution, but a solution that is close to perfect. It would take a lot more time to compute the perfect solution so the barycentric sorting method was chosen which is a lot faster.

References

1. Radivojac P. A (not so) Quick Introduction to Protein Function Prediction. 2013.
2. Expansion of the Gene Ontology knowledgebase and resources. *Nucleic Acids Res.* 2016;45(D1):D331-D338. doi:10.1093/nar/gkw1108.
3. AmiGO 2: Search. *Amigogeneontologyorg*. 2018. Available at: <http://amigo.geneontology.org/amigo/search/ontology>. Accessed June 12, 2018.
4. Araujo F, Barh D, Silva A, Guimarães L, Ramos R. GO FEAT: a rapid web-based functional annotation tool for genomic and transcriptomic data. *Sci Rep.* 2018;8(1). doi:10.1038/s41598-018-20211-9.
5. Wei Q, Khan I, Ding Z, Yerneni S, Kihara D. NaviGO: interactive tool for visualization and functional similarity and coherence analysis with gene ontology. *BMC Bioinformatics.* 2017;18(1). doi:10.1186/s12859-017-1600-5.
6. Binns D, Dimmer E, Huntley R, Barrell D, O'Donovan C, Apweiler R. QuickGO: a web-based tool for Gene Ontology searching. *Bioinformatics.* 2009;25(22):3045-3046. doi:10.1093/bioinformatics/btp536.
7. Clark W, Radivojac P. Information-theoretic evaluation of predicted ontological annotations. *Bioinformatics.* 2013;29(13):i53-i61. doi:10.1093/bioinformatics/btt228.
8. Healy, P., & Nikolov, N. S. (2013). Hierarchical Drawing Algorithms. *Handbook of Graph Drawing and Visualization*, 409–453.
9. Sugiyama K, Tagawa S, Toda M. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Trans Syst Man Cybern.* 1981;11(2):109-125. doi:10.1109/tsmc.1981.4308636.
10. Bokeh Development Team (2014). Bokeh: Python library for interactive visualization. <http://www.bokeh.pydata.org>.
11. Savitzky A, Golay M. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Anal Chem.* 1964;36(8):1627-1639. doi:10.1021/ac60214a047.
12. James A. Bednar, Jim Crist, Joseph Cottam, and Peter Wang (2016). "Datashader: Revealing the Structure of Genuinely Big Data", 15th Python in Science Conference (SciPy 2016).

Appendixes

Appendix 1: The longest-path algorithm pseudocode:

V = vertices (terms)

E = Edges (relationships)

U = all placed terms

Z = terms placed during the last layer

v = vertice (term)

$N_g^+(v)$ = Parent of vertice (term)

currentLayer = current layer

Requires: DAG $G = (V, E)$

$U \leftarrow \phi$

$Z \leftarrow \phi$

currentLayer $\leftarrow 1$

while $U \neq V$ do

 Select vertex $v \in V \setminus U$ with $N_g^+(v) \subseteq Z$

 if v has been selected then

 Assign v to the layer with a number currentLayer

$U \leftarrow U \cup \{v\}$

 end if

 if no vertex has been selected then

 currentLayer \leftarrow currentLayer + 1

$Z \leftarrow Z \cup U$

 end if

end while