

Visualization of Gene Ontology and Cluster Analysis Results

Master's Thesis Report (30 credit points)

Supervisor: Prof. Dr. Andreas Kerren

Co-Supervisor: Ilir Jusufi

Student: Vladyslav Aleksakhin

Linnaeus University
School of Computer Science, Physics and Mathematics

Contents

Glossary	5
1 Introduction	6
1.1 Motivation	6
1.2 Thesis Collaboration	6
1.3 Report structure	6
2 Background	7
2.1 Information Visualization	7
2.2 Bioinformatics	9
2.3 Gene Ontology	9
2.4 Clustering	10
3 Data Extraction Algorithm	10
3.1 Dataset Description	11
3.2 GML Graph File Format	14
3.3 Other Graph File Formats	16
3.3.1 GraphML	16
3.3.2 DOT Graph File Format	18
3.3.3 DGML	19
3.3.4 GXL	19
3.3.5 SVG	19
4 Visualization Approach	20
4.1 Probe Clustering Visualization	20
4.2 Cluster Analysis Results Visualization	23
4.3 Gene Ontology Visualization	24
5 Implementation	28
5.1 Java Graph Libraries Overview	30
5.2 OpenGL Visualization Standard	34
5.3 Program architecture	34
6 Conclusion and Future Work	37
References	39
Appendix A: Listing of a Complex GML File	44
Appendix B: Subgraph Extraction Algorithm	46
Appendix C: Complete UML Class Diagram	47
Appendix D: Program Execution Screenshots	48

Abstract

The task of the thesis is a new visualization method. Gene Ontologies and hierarchical clustering are both important tools in biology and medicine to study high-throughput data such as transcriptomics and metabolomics data. Enrichment of ontology terms in the data is used to identify statistically overrepresented ontology terms, giving insight into relevant biological processes or functional modules. Hierarchical clustering is a standard method to analyze and visualize data to find relatively homogeneous clusters of experimental data points. Both methods support the analysis of the same data set, but are usually considered independently. However, often a combined view is desired: visualizing a large data set in the context of an ontology under consideration of a clustering of the data.

Acknowledgments

I would like to express my gratitude to my supervisors Dr. Andreas Kerren and to co-supervisor Ilir Jusufi. They encouraged and stimulated me in each step of this work.

Moreover, my greatest gratitude goes to my family that made my study possible, for their constant moral support even though thousands of kilometers separated us.

Finally, I would like to thank to my best friend Olga for support and patience.

Glossary

- **GO**
Gene Ontology;
- **XML**
eXtensible Markup Language;
- **GML**
Graph Modeling Language;
- **SVG**
Scalable Vector Graphics;
- **IDE**
Integrated Development Environment;
- **JUNG**
Java Universal Network / Graph Framework;
- **JOGL**
Java OpenGL;
- **JNI**
Java Native API;
- **LWJGL**
Lightweight Java Game Library;

1 Introduction

1.1 Motivation

Computer information analysis is the only way to work with huge amount of data produced today. In any field of human work there are data and there are tasks of analyzing it. The topic of the thesis combined two areas of data computing: information visualization and biological data processing.

Data for processing was provided by the Plant Bioinformatics Group of Leibniz Institute of Plant Genetics and Crop Plant Research (IPK), Germany. Data consists of Gene Ontologies and hierarchical clustering, which are both important tools in biology and medicine to study high-throughput data.

The aim of this work is to create a new visualization approach and implement it to provide a useful tool for biologists in their everyday research work.

1.2 Thesis Collaboration

There are many datasets for analysis in biology. This work is intended for making a visualization tool for genes and gene relations, in order to help biologists with their work with genes. This project is result of collaboration between ISOVIS research group (Head: Prof. Dr. Andreas Kerren [13]) of Linnaeus University at Växjö, Sweden, and Plant Bioinformatics Group (Head: Prof. Dr. Falk Schreiber [14]) of Leibniz Institute of Plant Genetics and Crop Plant Research (IPK), Germany.

ISOVIS research group mainly focuses on the exploration analysis and visualization of typically large information data in Software Engineering, Geography, or Biology. There are several different techniques for information visualization, one of them is Human-Centered Visualization widely used in the research. It combines traditional visualization techniques with the ability of the human visual-brain system and/or the haptic-motoric system to explore and analyze complex data sets comprehensively. This kind of visualization merges several aspects of different research areas, such as Information Visualization, Scientific Visualization, Human-Computer Interaction, Data Mining, Information Design, Graph Drawing, and Computer Graphics.

As said on official page of Plant Bioinformatic Group: „The research group focuses on modeling, analysis, simulation and visualization of biological networks in the context of plant biological problems. Our aim is the development of methods and software tools for the analysis of complex biological networks. Therefore we integrate, process and analyze data from different areas of genome, proteome and metabolome research and present the results in a user-friendly way. The emphasis is on the linkage of experimental data about expression profiles and metabolite patterns with metabolic and regulatory networks. The data and complex connections are modeled using graphs. We are developing graph (network) analysis and interactive visualization methods to discover network properties and to make the data easily accessible to the user. A subsequent step is to use the data for the simulation of metabolic and regulatory networks.” [16]

1.3 Report structure

Section 1 presents the results of the literature review, where been exploring relevant research efforts in the fields of Bioinformatic, Information Visualization and Bio Visualization, in connection to the work. In Section 3 explains the problem and purpose of this work. Section is split in several parts which covers topics: thesis purpose and collaboration, visualization complexity and goals, description of the input data, data format overview, overview of different graph file formats. In this section, Problem Statement and Goals, have determined the requirements, use cases, and proposed the architecture for thesis application. First part of the Section 4 describes visualization solution for Cluster

Ananlisys tree, second part explains Gene Ontology visualization technique. On of the biggest section in the report is Section 5 which covers implementation details, architecture of the system, used libraries, project management tool, etc. Technical details of the visualization algorithms are explained in the Sections 4.2 and 4.3. Last part of the report is Section 6 describes problems that been faced during work and future improvements.

The result of this work is a base for a research paper – Ilir Jusufi, Andreas Kerren, Vladyslav Aleksakhin, and Falk Schreiber. *Visualization of Mappings between the Gene Ontology and Cluster Trees*. Conference on Visualization and Data Analysis 2012 (EI-107) Part of IST/SPIE Electronic Imaging 2012 (Jan 22-26, 2012) VDA 2012 Dates: January 23-25, 2012.

2 Background

2.1 Information Visualization

The field of computer-based information visualization draws on ideas from several intellectual traditions: computer science, psychology, semiotics, graphic design, cartography, and art. The two main threads of computer science relevant for visualization are computer graphics and human-computer interaction. The areas of cognitive and perceptual psychology offer important scientific guidance on how humans perceive visual information. A related conceptual framework from the humanities is semiotics, the study of symbols and how they convey meaning. Design, as the name suggests, is about the process of creating artifacts well-suited for their intended purpose. Cartographers have a long history of creating visual representations that are carefully chosen abstractions of the real world. Finally, artists have refined methods for conveying visual meaning in sub-disciplines ranging from painting to cinematography.

Information visualization has gradually emerged over the past fifteen years as a distinct field with its own research agenda. The distillation of results from areas with other goals into specific prescriptive advice that can help us design and evaluate visualization systems is nontrivial. Although these traditions have much to offer, effective synthesis of such knowledge into a useful methodology of our own requires arduous gleaning.

The standard argument for visualization is that exploiting visual processing can help people explore or explain data. We have an active field of study because the design challenges are significant and not fully understood. Questions about visual encoding are even more central to information visualization than to scientific visualization. The subfield names grew out of an accident of history, and have some slightly unfortunate connotations when juxtaposed: information visualization is not unscientific, and scientific visualization is not uninformative. The distinction between the two still not agreed on by all, but the definition used here is that information visualization hinges on finding a spatial mapping of data that is not inherently spatial, whereas scientific visualization uses a spatial layout that is implicit in the data.

„Graph drawing or Graph layout, as a branch of graph theory, applies topology and geometry to derive two-dimensional representations of graphs. A drawing of a graph is basically a pictorial representation of an embedding of the graph in the plane (generally, with edge intersections allowed), usually aimed at a convenient visualization of certain properties of the graph in question or of the object modeled by the graph. Graph drawing is motivated by applications such as Very-large-scale (VLSI) [1] integration circuit design, social network analysis, cartography, and bioinformatics, many of which make use of information visualization.” [2]

There are lots of methods and approaches to graph visualization. They are all based on various perception qualities of human. The Radial Dendograms [3] algorithm is used for results of cluster analysis visualization.

Dendrogram [4] plot is one of the visualization algorithm for hierarchical structures. It illustrating the outcome of decision tree-type clustering in statistics. Most commonly,

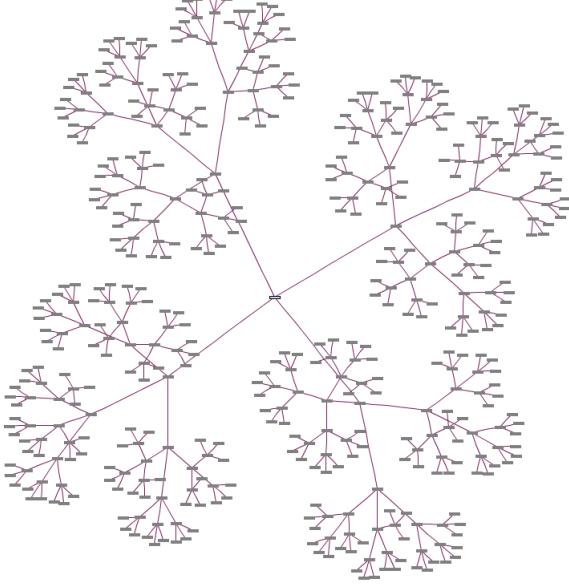


Figure 1: Sample visualisation of the tree graph

dendograms are drawn in a Cartesian layout, as an upright tree. However, this layout does not make good use of space, it is sparse towards the root and crowded towards the leaf nodes. The spacing between nodes at different levels in the hierarchy is not uniform, which is due to the shrinking number of nodes from bottom to top. For this reason, long, wide-spanning connecting lines are needed to merge nodes at higher levels. A better layout in this respect is the polar or radial layout, where leaf nodes are located on the outer ring and the root is located in the center, as a focal point. A more uniform node spacing results, leading to a better utilization of space and resulting in a better illustration of the class relationships. Recently, Barlow and Neville [5] presented an empirical user study for tree layouts (with less than 200 leaves) in which they compare some of the major schemes: organizational chart (a standard drawing of a tree), tree ring (basically a pie chart of circular segments), icicle plot (the cartesian version of the tree ring), and tree map. According to the measured performance within a group of 15 users, the three former methods yielded similar results, with the icicle plot having a slight advantage. However, given the much larger number of leaves in our case (1000 and more) and the fact that the tree ring is the most compact of the three winning configurations, a radial layout seemed to be the most favorable one for our purpose. Radial graph layouts that illustrate hierarchical relationships are very popular, and for the special application of dendograms, we know only of one other application using a radial layout, the recent one by Kreussler and Schumann [6] that mentioned before, users often would like to focus on certain portions of the display, while compressing others, without losing context. Fisheye lenses and hyperbolic zooming have been proposed to provide these capabilities. In the context of tree rings Yang, Ward and Rundensteiner [7] have proposed a system in which users may either perform a polar zoom (i.e. expand the width of one or more adjacent rings while reducing others) or a radial zoom (i.e. expand the arc angle of some adjacent segments while reducing others). Users can perform these operations by pinning down one ring or arc segment and dragging another. A limiting factor here is that users cannot perform both operations simultaneously, which can be awkward in certain instances. To address this shortcoming, our application generalizes these concepts by allowing arbitrary warps of the dendrogram domain, i.e. we allow radial and polar zooms simultaneously.

2.2 Bioinformatics

"In the last few decades, advances in molecular biology and the equipment available for research in this field have allowed the increasingly rapid sequencing of large portions of the genomes of several species. In fact, to date, several bacterial genomes, as well as those of some simple eukaryotes (e.g., *Saccharomyces cerevisiae*, or baker's yeast) have been sequenced in full. The Human Genome Project, designed to sequence all 24 of the human chromosomes, is also progressing. Popular sequence databases, such as GenBank and EMBL, have been growing at exponential rates. This deluge of information has necessitated the careful storage, organization and indexing of sequence information. Information science has been applied to biology to produce the field called Bioinformatics.

The simplest tasks used in bioinformatics concern the creation and maintenance of databases of biological information. Nucleic acid sequences (and the protein sequences derived from them) comprise the majority of such databases. While the storage and organization of millions of nucleotides is far from trivial, designing a database and developing an interface whereby researchers can both access existing information and submit new entries is only the beginning. The most pressing tasks in bioinformatics involve the analysis of sequence information" [8]

Here we can find short introduction and history of Bioinformatics: „Bioinformatics is the application of information technology to the field of molecular biology. The term bioinformatics was coined by Paulien Hogeweg in 1978 for the study of informatic processes in biotic systems. Bioinformatics now entails the creation and advancement of databases, algorithms, computational and statistical techniques, and theory to solve formal and practical problems arising from the management and analysis of biological data. Over the past few decades rapid developments in genomic and other molecular research technologies and developments in information technologies have combined to produce a tremendous amount of information related to molecular biology. It is the name given to these mathematical and computing approaches used to glean understanding of biological processes. Common activities in bioinformatics include mapping and analyzing DNA and protein sequences, aligning different DNA and protein sequences to compare them and creating and viewing 3-D models of protein structures." [9] More resources about Bioinformatic are available here [?]. The primary goal of bioinformatics is to increase our understanding of biological processes. What sets it apart from other approaches, however, is its focus on developing and applying computationally intensive techniques (e.g., data mining, machine learning algorithms, and visualization) to achieve this goal. Major research efforts in the field include sequence alignment, gene finding, genome assembly, protein structure alignment, protein structure prediction, prediction of gene expression and protein-protein interactions, genome-wide association studies and the modeling of evolution.

2.3 Gene Ontology

As stated above, there are different knowledge databases for biologic information storage. Gene Ontology [10] project is one of the first-rate international projects. The Gene Ontology, or GO, is a major bioinformatics initiative to unify the representation of gene and gene product attributes across all species. The aims of the Gene Ontology project are threefold; firstly, to maintain and further develop its controlled vocabulary of gene and gene product attributes; secondly, to annotate genes and gene products, and assimilate and disseminate annotation data; and thirdly, to provide tools to facilitate access to all aspects of the data provided by the Gene Ontology project. The GO is part of a larger classification effort, the Open Biomedical Ontologies (OBO) [11]. The Gene Ontology project provides an ontology of defined terms representing gene product properties. The ontology covers three domains; cellular component, the parts of a cell or its extracellular environment; molecular function, the elemental activities of a gene product at the molecu-

lar level, such as binding or catalysis; and biological process, operations or sets of molecular events with a defined beginning and end, pertinent to the functioning of integrated living units: cells, tissues, organs, and organisms. Each GO term within the ontology has a term name, which may be a word or string of words; a unique alphanumeric identifier; a definition with cited sources; and a name space indicating the domain to which it belongs. Terms may also have synonyms, which are classed as being exactly equivalent to the term name, broader, narrower, or related; references to equivalent concepts in other databases; and comments on term meaning or usage. The GO ontology is structured as a directed acyclic graph, and each term has defined relationships to one or more other terms in the same domain, and sometimes to other domains. The GO vocabulary is designed to be species-neutral, and includes terms applicable to prokaryotes and eukaryotes, single and multicellular organisms. The GO ontology is not static, and additions, corrections and alterations are suggested by, and solicited from, members of the research and annotation communities, as well as by those directly involved in the GO project. For example, an annotator may request a specific term to represent a metabolic pathway, or a section of the ontology may be revised with the help of community experts. Suggested edits are reviewed by the ontology editors, and implemented where appropriate.

2.4 Clustering

„Data clustering (or just clustering), also called cluster analysis, segmentation analysis, taxonomy analysis, or unsupervised classification, is a method of creating groups of objects, or clusters, in such a way that objects in one cluster are very similar and objects in different clusters are quite distinct. Data clustering is often confused with classification, in which objects are assigned to predefined classes. In data clustering, the classes are also to be defined.” [12]

Clustering algorithms can be applied in many fields, for instance:

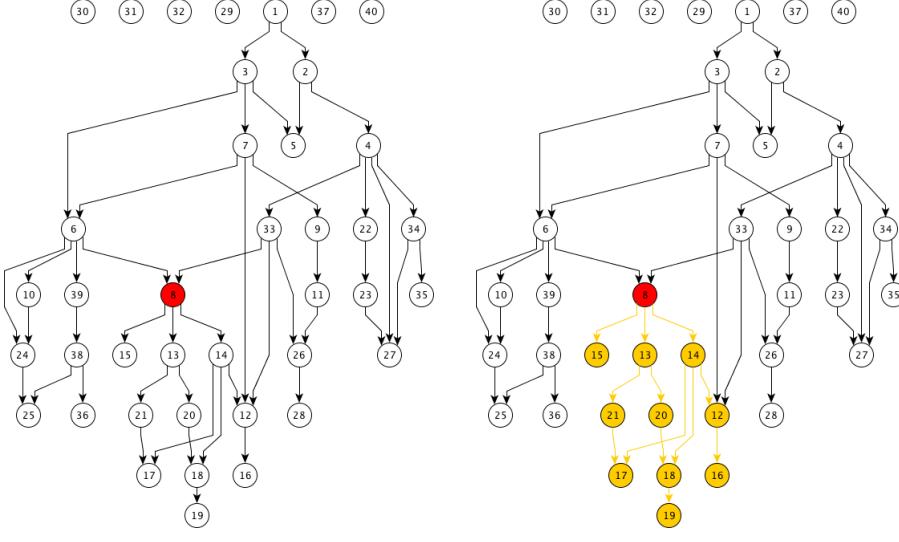
- Marketing: finding groups of customers with similar behavior given a large database of customer data containing their properties and past buying records;
- Biology: classification of plants and animals given their features;
- Libraries: book ordering;
- Insurance: identifying groups of motor insurance policy holders with a high average claim cost; identifying frauds;
- City-planning: identifying groups of houses according to their house type, value and geographical location;
- Earthquake studies: clustering observed earthquake epicenters to identify dangerous zones;
- WWW: document classification; clustering web log data to discover groups of similar access patterns.

3 Data Extraction Algorithm

First step in the program work is to load data and to compute connected components. Data is stored in GML file format. More detail information about this format and other common graph file formats is explained in the Section 3.1.

Here is program algorithm explanation using sample graphs:

1. The program visualizes Gene Ontology and cluster analysis result tree. Visualization technique is discussed in the Section 4.



(a) Selected node in the Gene Ontology (b) Extract sub-graph for selected node

Figure 2: Sub-graph extraction from the Gene Ontology

2. Interactively select node in the Gene Ontology graph (Figure 2a).
3. When node is selected in the Gene Ontology program computes all successors (Figure 2b).
4. Extract leafs from successors (Figure 3).
5. Founded sub tree cached. This sub tree is highlighted in cluster analysis tree.
6. Then the program searches corresponded leaves in cluster analysis result tree by label as seen in Figure 4a.
7. For this leaves the program founds root connected to all leaves and extract corresponding sub trees (Figure 4b).

The aim of the work is to provide flexible tool specially made for biology scientists to deal with huge data sets. Trace relations, as was discussed earlier, in the two separated datasets (Gene Ontology graph and cluster analysis result tree): both graphs are stored separately in the different files. Also during program design we should consider that cluster analysis results graph was produced from Gene Ontology graph using separate tool and clustering algorithm specific to the purpose, that means there are various cluster graphs for the same Gene Ontology.

Provide effective space filling visualization method and allow interactive relations highlighting. The tool also provides ability to track throw graphs discovering: focused or selected vertices labels for both graphs separately, Gene Ontology levels.

Considering end-user requirements there is use case specific for biology scientist – interest in the concrete gene in the Gene Ontology; provide search mechanism to find specific gene by name and view it.

3.1 Dataset Description

Gene Ontology data and cluster analysis results presented as directed graphs. They are stored in separate files in special format – GML files. GML file format covered in the next section.

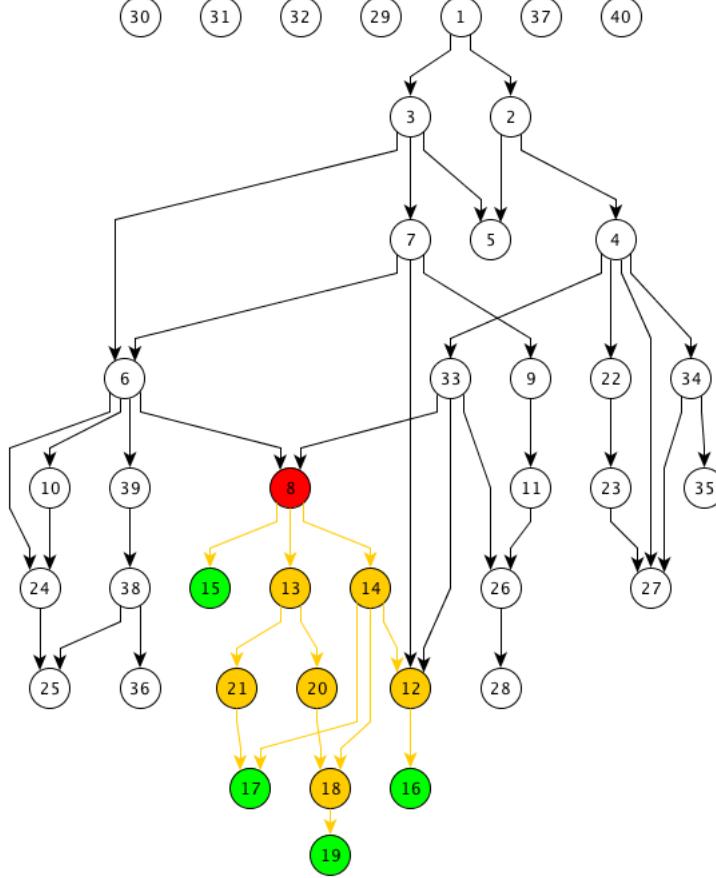


Figure 3: Extract leafs from the Gene Ontology sub-graph

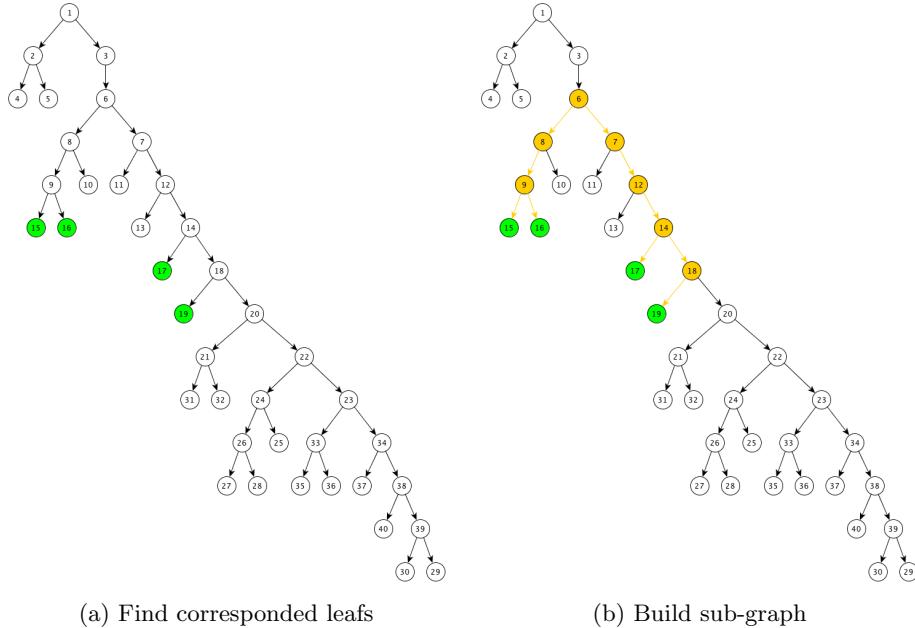


Figure 4: Analise Cluster graph

GO graph is directed acyclic graph has 10,042 vertices and 24,155 edges. It has 1 root, 2,729 nodes and 7,312 leafs (terminal nodes). Figure 5 shows visualization of the Gene Ontology graph using Hierarchical layout by yEd [20] graph editing tool. It obviously

shows imperfectness of classic visualization of the graphs.

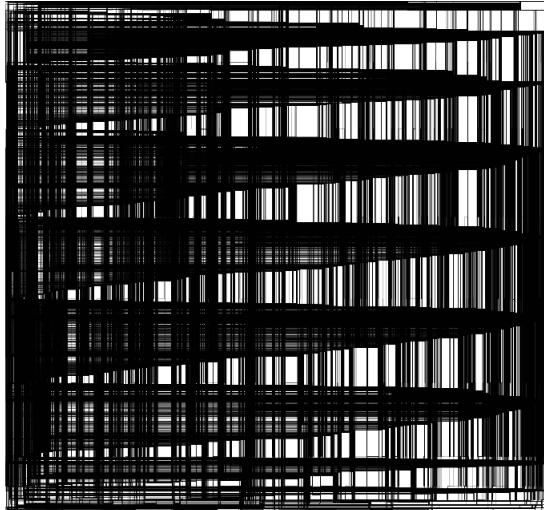


Figure 5: Gene Ontology yEd visualization

Cluster graph is directed binary tree has 14,623 nodes and 14,622 edges. Cluster graph as a tree has single root, 7,310 nodes and 7,312 leafs. To get an impression of the graph Figure 6 shows a cluster tree using Cytoscape [22] visualization tool.

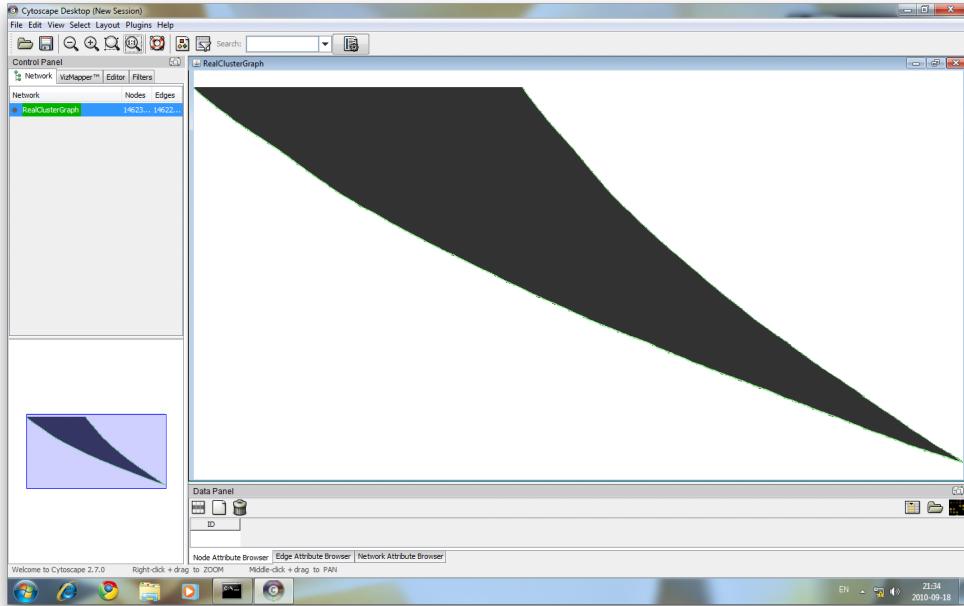


Figure 6: Cluster analysis result tree Cytoscape visualization tool

Both graphs are independent from each other from developer point of view: they have different node ids and edge ids. But they are corresponded by graph node labels – both of this graphs have same label for terminal (leaf) graph nodes. This property is used in the sub graph extracting algorithm.

The application should work with large quantity of data over tens of thousands that is why performance is one of the main requirements. It is important to give a consideration on optimization.

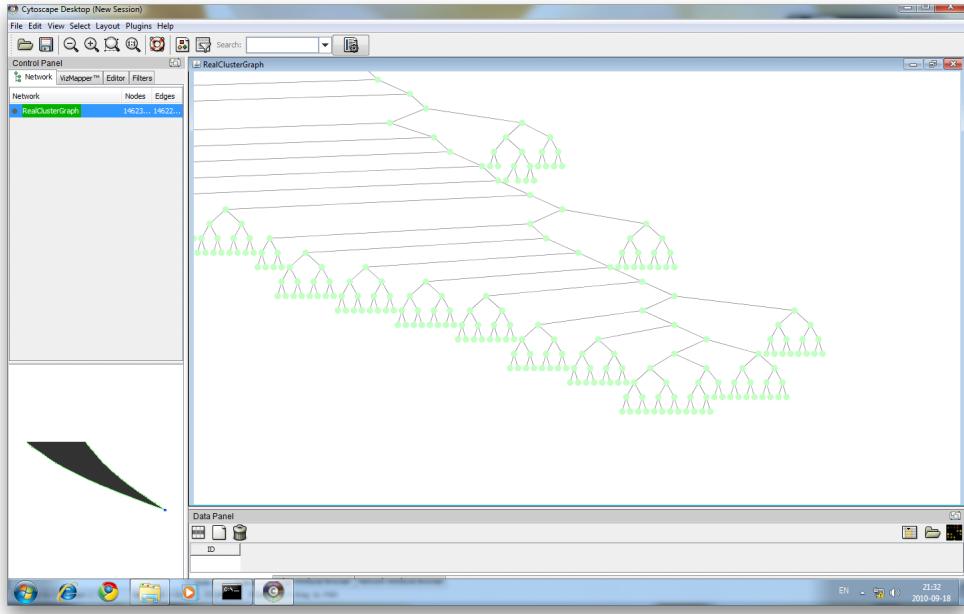


Figure 7: Zoomed cluster analysis result tree

3.2 GML Graph File Format

,,GML, the Graph Modeling Language, is our proposal for a portable file format for graphs. GML's key features are portability, simple syntax, extensibility and flexibility. A GML file consists of a hierarchical key-value lists. Graphs can be annotated with arbitrary data structures. The idea for a common file format was born at the GD'95; this proposal is the outcome of many discussions. GML is the standard file format in the Graphlet citeGraphlet graph editor system. It has been overtaken and adapted by several other systems for drawing graphs.” [18]

GML format is platform independent, and easy to implement. Furthermore, it has the capability to represent arbitrary data structures, since advanced programs have the need to attach their specific data to nodes and edges. GML is flexible enough that a specific order of declarations is not needed, and that any non-essential data may be omitted. Simple graph is showed in the Listing 1

Listing 1: GML description of sample graph

```
graph [
  directed 1
  node [
    id 1
  ]
  node [
    id 2
  ]
  node [
    id 3
  ]
  edge [
    source 1
    target 2
  ]
```

```

edge [
    source 2
    target 3
]
edge [
    source 3
    target 1
]
]

```

Figure 8 shows manual visualization of the sample graph using yEd [20] graph visualization tool.

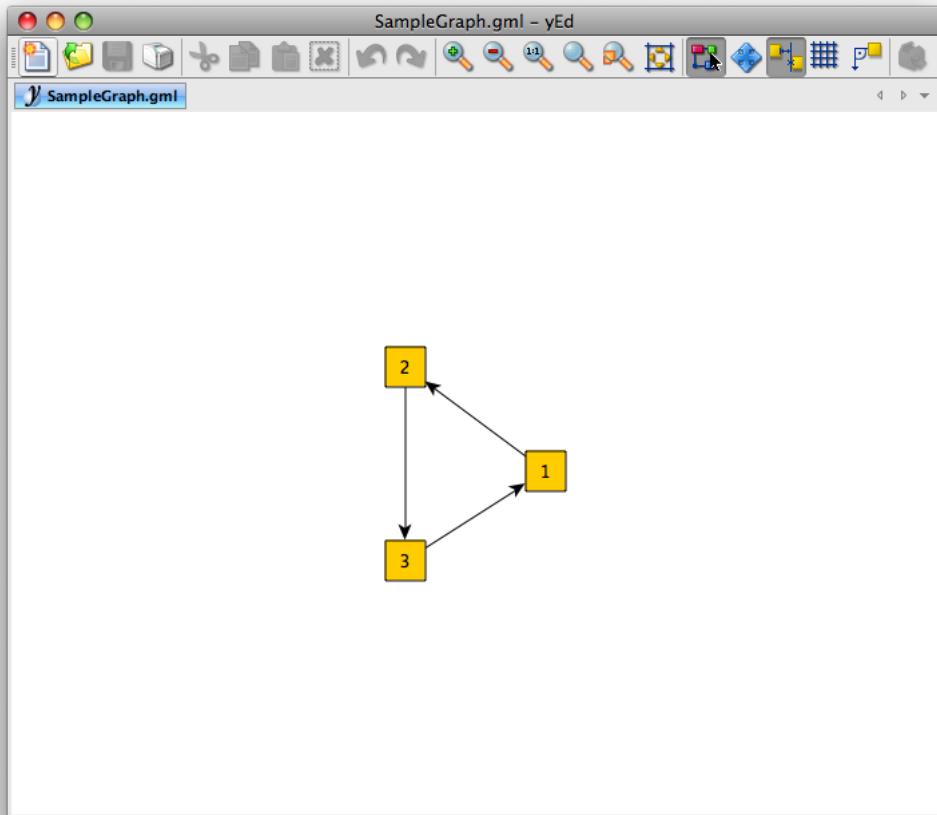


Figure 8: Manual visualization of the sample graph

Listing of the more complex graph with additional properties and is in the Appendix A and the visualization of this graph showed in Figure 9

Applications supporting GML [19]

- Clairlib [21], a suite of open-source Perl modules intended to simplify a number of generic tasks in natural language processing (NLP), information retrieval (IR), and network analysis (NA).
- Cytoscape [22], an open source bioinformatics software platform for visualizing molecular interaction networks, loads and save previously-constructed interaction networks in GML.
- NetworkX [23], an open source Python library for studying complex graphs.

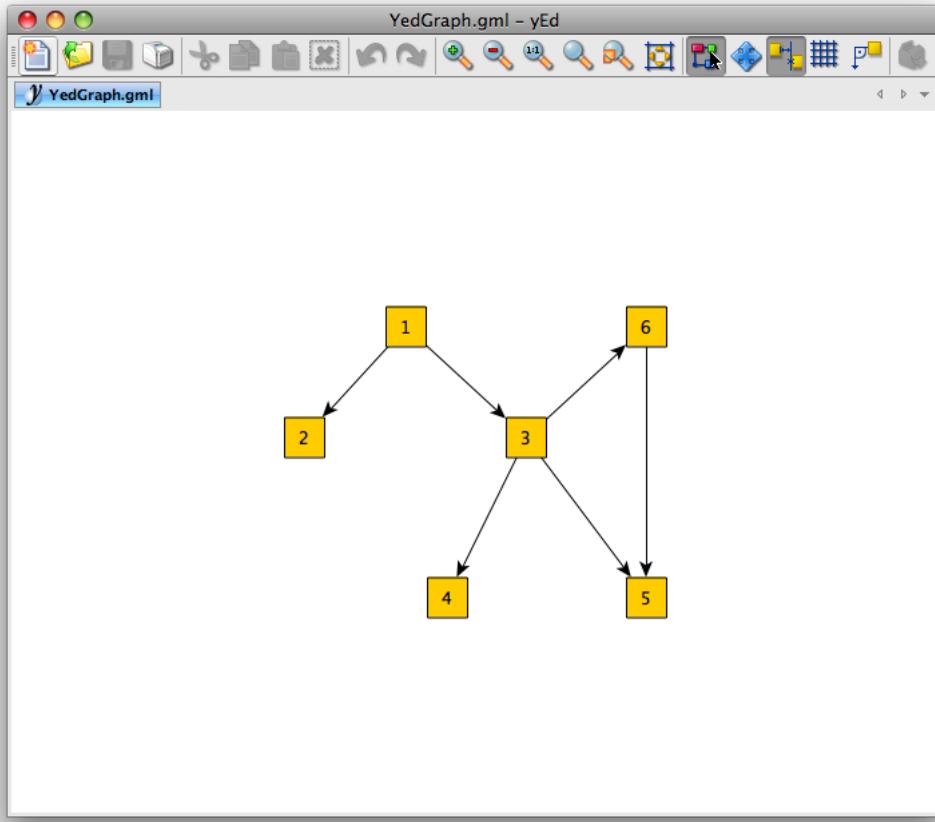


Figure 9: Manual visualization of the sample graph

- ocamlgraph[24], a graph library for OCaml.
- OGDF[25], the Open Graph Drawing Framework, an open source C++ library containing implementations of various graph drawing algorithms. The library is self contained; optionally, additional packages like LP-solvers are required for some implementations.
- Tulip [26] (software) is a free software in the domain of information visualization capable of manipulating huge graphs (with more than 1.000.000 elements).
- yEd [20], a free Java-based graph editor, supports import from and export to GML.

3.3 Other Graph File Formats

3.3.1 GraphML

GraphML is a comprehensive and easy-to-use file format for graphs. It consists of a language core to describe the structural properties of a graph and a flexible extension mechanism to add application-specific data. [27] Its main features include support of:

- directed, undirected, and mixed graphs;
- hyper graphs;
- hierarchical graphs;
- graphical representations;

- references to external data;
- application-specific attribute data;
- light-weight parsers;

The GraphML document consists of a graphml element and a variety of sub elements: graph, node, edge. Figure 10 below is a simple graph. It contains 11 nodes and 12 undirected edges.

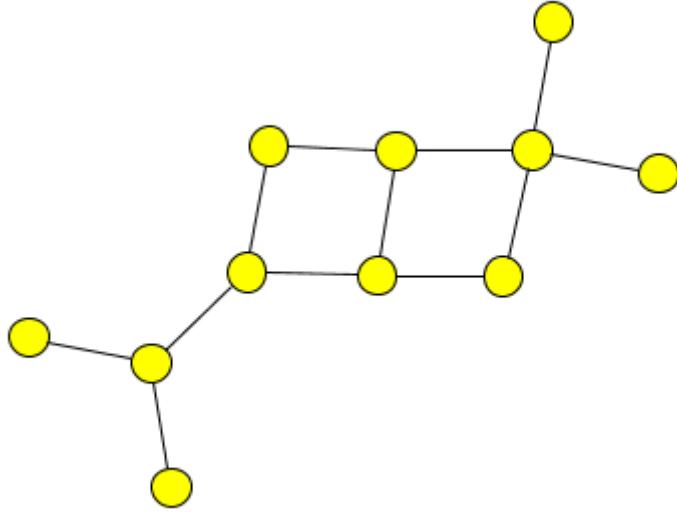


Figure 10: A simple graph

And a corresponded graphml file is showed in Listing 2

Listing 2: Simple graphml file

```
<graphml>
  <graph id="G" edgedefault="undirected">
    <node id="n0"/>
    <node id="n1"/>
    <node id="n2"/>
    <node id="n3"/>
    <node id="n4"/>
    <node id="n5"/>
    <node id="n6"/>
    <node id="n7"/>
    <node id="n8"/>
    <node id="n9"/>
    <node id="n10"/>
    <edge source="n0" target="n2"/>
    <edge source="n1" target="n2"/>
    <edge source="n2" target="n3"/>
    <edge source="n3" target="n5"/>
    <edge source="n3" target="n4"/>
    <edge source="n4" target="n6"/>
    <edge source="n6" target="n5"/>
    <edge source="n5" target="n7"/>
```

```

<edge source="n6" target="n8" />
<edge source="n8" target="n7" />
<edge source="n8" target="n9" />
<edge source="n8" target="n10" />
</graph>
</graphml>

```

3.3.2 DOT Graph File Format

DOT is a plain text graph description language. It is a simple way of describing graphs in human readable form.

„DOT graphs are typically files that end with the .gv (or .dot) extension. At its simplest, DOT can be used to describe an undirected graph. An undirected graph shows simple relations between objects, such as friendship between people. The graph keyword is used to begin a new graph, and nodes are described within curly braces. A double-hyphen (- -) is used to show relations between the nodes.” [28]

Listing 3: DOT file format: undirected graph

```

graph graphname {
    a -- b -- c;
    b -- d;
}

```

„Similar to undirected graphs, DOT can describe directed graphs, such as flowcharts and dependency trees. The syntax is the same as for undirected graphs, except the digraph keyword is used to begin the graph, and an arrow (- >) is used to show relationships between nodes.” [28]

Listing 4: DOT file format: directed graph

```

graph graphname {
    a -> b -> c;
    b -> d;
}

```

Visual representation of both graphs showed Figure 11 below.

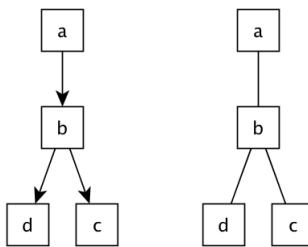


Figure 11: Directed and undirected graphs

3.3.3 DGML

DGML is an XML-based file format for directed graphs. Here is what a simple directed graph with three nodes and two links between them looks like

Listing 5: DGML file format

```
<?xml version='1.0' encoding='utf-8'?>
<DirectedGraph xmlns="http://schemas.microsoft.com/vs/2009/dgml">
  <Nodes>
    <Node Id="a" Label="a" Size="10" />
    <Node Id="b" Background="#FF008080" Label="b" />
    <Node Id="c" Label="c" Start="2010-06-10" />
  </Nodes>
  <Links>
    <Link Source="a" Target="b" />
    <Link Source="a" Target="c" />
  </Links>
  <Properties>
    <Property Id="Background" Label="Background" DataType="Brush" />
    <Property Id="Label" Label="Label" DataType="String" />
    <Property Id="Size" DataType="String" />
    <Property Id="Start" DataType="DateTime" />
  </Properties>
</DirectedGraph>
```

The complete XSD schema for DGML is available at <http://schemas.microsoft.com/vs/2009/dgml/>. DGML not only allows describing nodes and links in a graph, but also annotating those nodes and links with any user defined property and/or category.

3.3.4 GXL

GXL (Graph eXchange Language) is an XML based exchange format for several kinds of graph information. GXL provides a standardized notation for exchanging instance data (graph) including their structure (graph schema).

„GXL was created to fulfill the need to exchange data between re-engineering tools. Previously, interoperability between tools relied on converters between local formats. This approach requires case-by-case negotiation of exchange semantics. As the research area matured, it became apparent that a standard exchange format was needed and that this format should provide a mechanism to help articulate semantics.” [29].

3.3.5 SVG

SVG is open graphical data storage format. Has been in development since 1999 by a group of companies within the W3C. SVG drew on experience from the designs of two older formats: Precision Graphics Markup Language (PGML) developed from Adobe’s PostScript and Vector Markup Language (VML) developed from Microsoft’s RTF. Which were submitted to W3C in 1998.

SVG allows three types of graphic objects:

- Vector graphics
- Raster graphics
- Text

„Graphical objects, including PNG and JPEG raster images, can be grouped, styled, transformed, and composited into previously rendered objects. SVG does not directly support z-indices that separate drawing order from document order for overlapping objects, unlike some other vector mark up languages like VML. Text can be in any XML name space suitable to the application, which enhances search ability and accessibility of the SVG graphics. The feature set includes nested transformations, clipping paths, alpha masks, filter effects, template objects and extensibility..” [30]

4 Visualization Approach

4.1 Probe Clustering Visualization

As was discussed before in the Section 3.1 cluster analysis result graph is binary tree.

„A binary tree is a connected acyclic graph such that the degree of each vertex is no more than 3. A rooted binary tree is such a graph that has one of its vertices of degree no more than 2 singled out as the root. With the root thus chosen, each vertex will have a uniquely defined parent, and up to two children; however, so far there is insufficient information to distinguish a left or right child. If we drop the connectedness requirement, allowing multiple connected component in the graph, we call such a structure a forest” [35] Simple binary tree showed in Figure 12

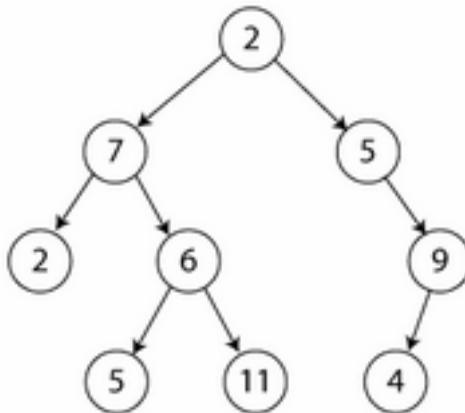


Figure 12: A simple binary tree graph

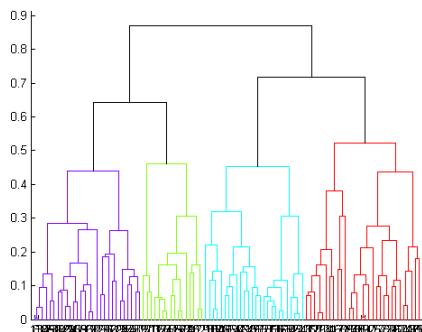
There are several visualization methods for binary trees and more specific methods for cluster results. The main method for visualizing clusters is a dendrogram. Here is sample dendrogram visualization in Figure 13a produced by MATLAB 7.2.

Figure 13b shows a polar dendrogram visualization algorithm of the same cluster tree produced by MATLAB.

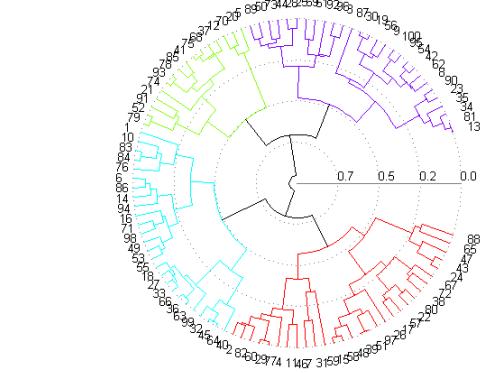
One of the main ideas was to use polar dendrogram algorithm for cluster visualization. Figure 14 shows visualization of the Cluster using native JUNG radial layout algorithm. Red are nodes and white is edges, black is background. As picture shows algorithm doesn't count nature of the cluster: very deep binary tree not wide as it is common for cluster analysis results, that's why it has many edge overlapping.

Also to visualization issue it has performance issue – without any measurement was seen for an eye that program does not allow smooth interaction. Low performance issue was in nature of the visualization in the JUNG: it uses very complex hierarchical structure with many utility classes per visualized object and the cost is big memory usage. Also JUNG uses Java 2D [36] which by itself is ‘heavyweight’ because it's part of the Java AWT – Abstract Windows Toolkit.

„The Abstract Window Toolkit (AWT) is Java's original platform-independent windowing, graphics, and user-interface widget toolkit. The AWT is now part of the Java



(a) Dendrogram



(b) Polar Dendrogram

Figure 13: Cluster visualisations

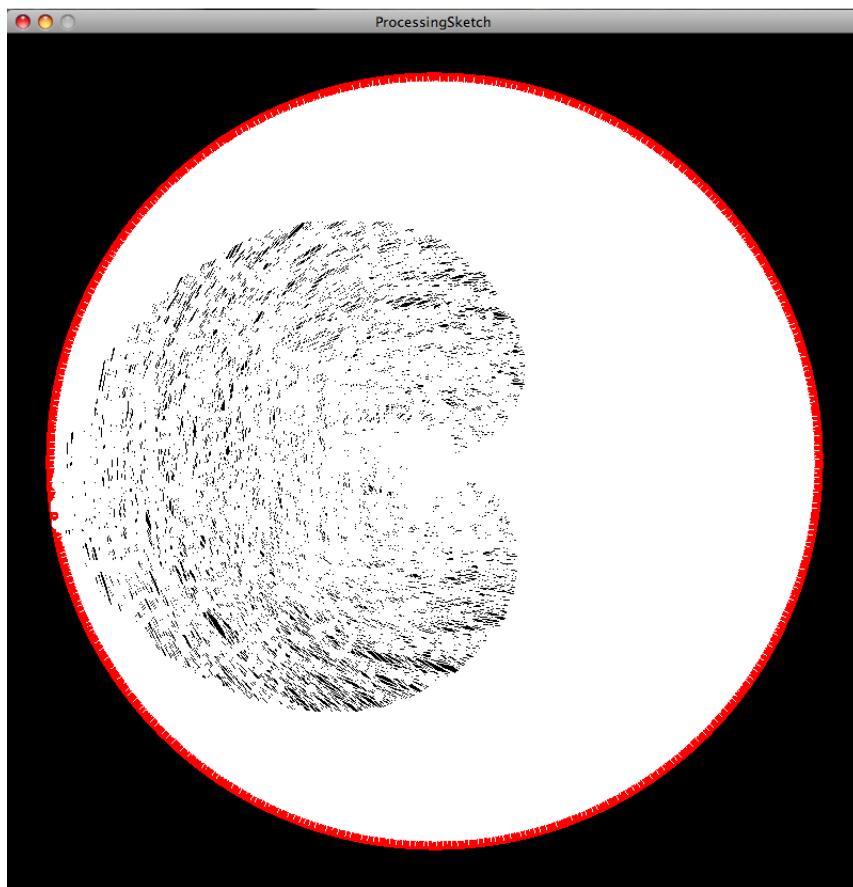


Figure 14: Cluster visualization using JUNG radial layout

Foundation Classes (JFC) the standard API for providing a graphical user interface (GUI) for a Java program. When Sun Microsystems first released Java in 1995, AWT widgets provided a thin level of abstraction over the underlying native user interface. For example, creating an AWT check box would cause AWT directly to call the underlying native subroutine that created a check box.” [37] This technology is outdated and replaced by Swing.

„Swing is the primary Java GUI widget toolkit. It is part of Sun Microsystems’ Java Foundation Classes (JFC) an API for providing a graphical user interface (GUI) for Java programs. Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit. Swing provides a native look and feel that

emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform.” [38]

„Since early versions of Java, a portion of the Abstract Window Toolkit (AWT) has provided platform-independent APIs for user interface components. In AWT, each component is rendered and controlled by a native peer component specific to the underlying windowing system. By contrast, Swing components are often described as lightweight because they do not require allocation of native resources in the operating system’s windowing toolkit. The AWT components are referred to as heavyweight components.” [38] More detail comparison can be found here. [39]

Figure 15 shows improved JUNG radial algorithm and own visualization implementation using JOGL.

Root has fixed position in the center of the ring. The layout places all leafs evenly on the outer ring based on the distance from root. Each node is placed on the concentric ring corresponding to its distance to the root node – level. Each node is radially centered over children. The number of concentric rings is exactly equal to the number of levels of the graph.

JOGL will be discussed in the Section 5.2.

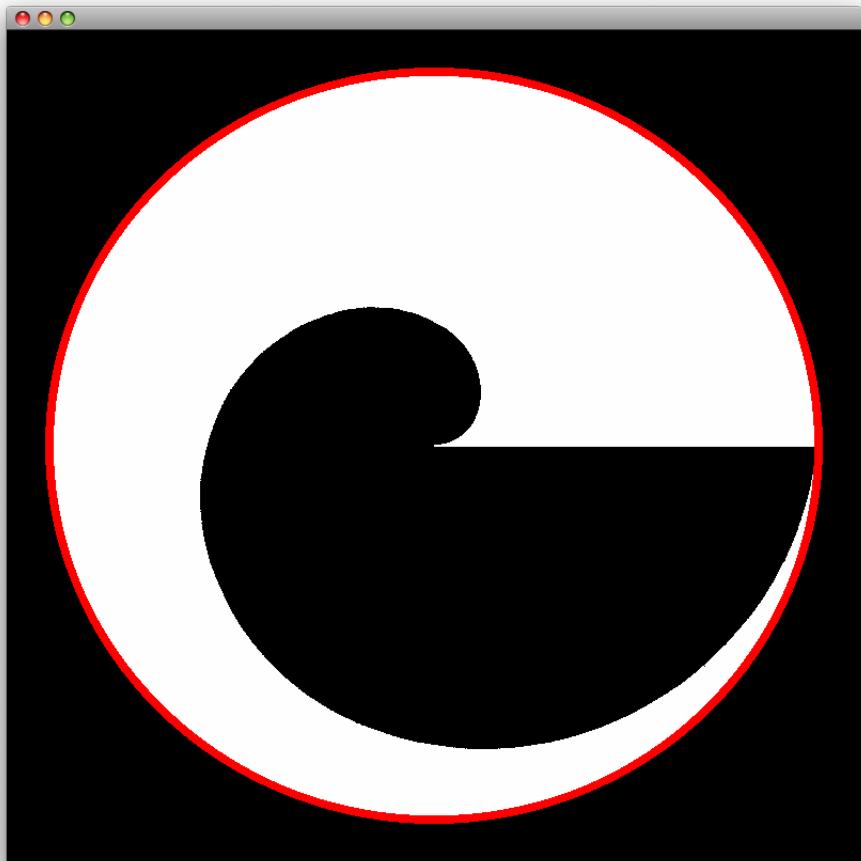
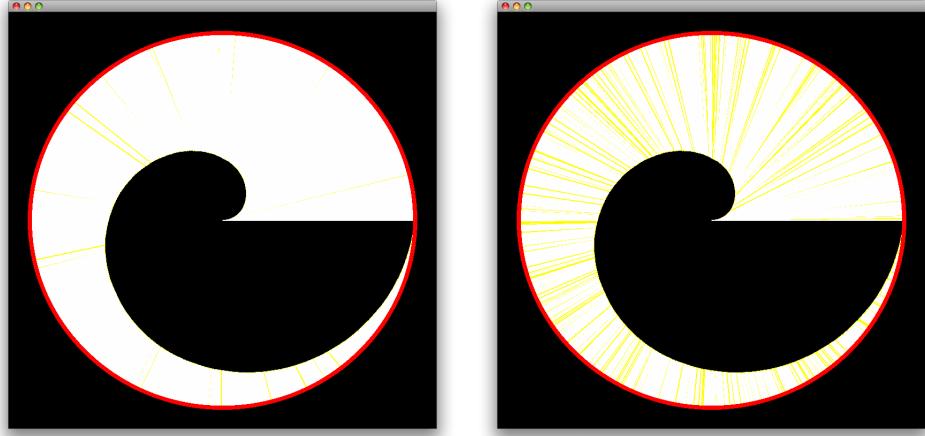


Figure 15: Cluster visualization using JOGL and improved JUNG radial layout

Figure 16a and Figure 16b shows cluster visualization and highlighted subgraph (subgraph extraction algorithm was discussed in the Section 3). This pictures show the nature of the dataset. Improved version has good performance and better visualization but still has issues: there are too many elements in the scene, it is impossible to identify separate gene or trace highlighted graph genes.



(a) Cluster graph and highlighted subgraph (b) Cluster graph and highlighted subgraph

Figure 16: Cluster graph visualization using improved JUNG polar dendrogram layout

4.2 Cluster Analysis Results Visualization

Figure 7 from Section 3.1 shows cluster graph specific structure: it is very high, unbalanced and has not so deep sub-parts. It is possible to use this disadvantage as advantage and abstract sub-parts to reduce drawing area. Extract those nodes and edges that form the longest path of the cluster graph - „backbone”. Figure 19 shows algorithm work step by step. Backbone vertices are filled with yellow and showed in Figure 17a. Next step is to abstract branches into groups, group size is scaled according to amount of elements inside.

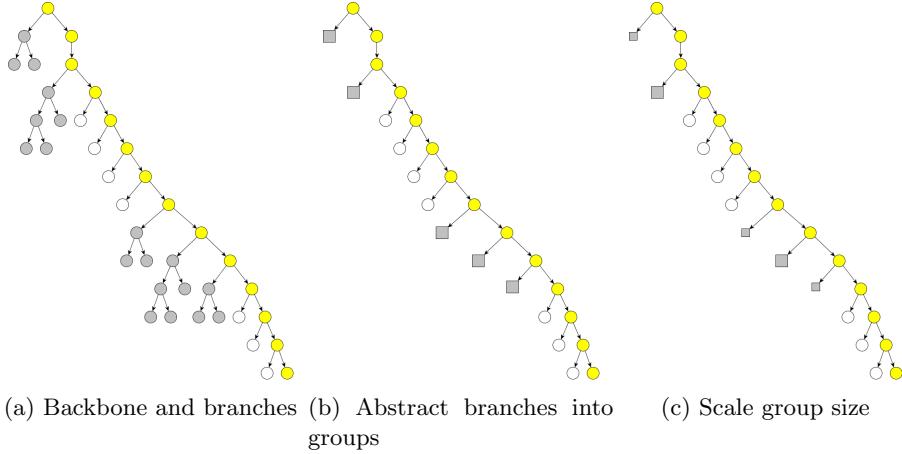


Figure 17: Cluster Visualization algorithm

The last step is to represent backbone as a spiral, thus preserving space and giving us a possibility to show the complete tree in one view. Figure 18 shows how this approach works.

Then backbone formed as rectangular spiral with a root in the center and moving in clockwise direction. Figure ?? shows complete visualization result for real cluster tree. This will have to reuse space as much as possible and still gives overview of location of the highlighted vertices in cluster hierarchy – how far from a root.

It is possible to explore sub-part (rectangles) of the Cluster graph using lens. User can interactively choose any sub-part and lens with inner content will appear. There are two different lens layouts: polar 20a and HV-tree 20b. Polar lens layout is based on algorithm

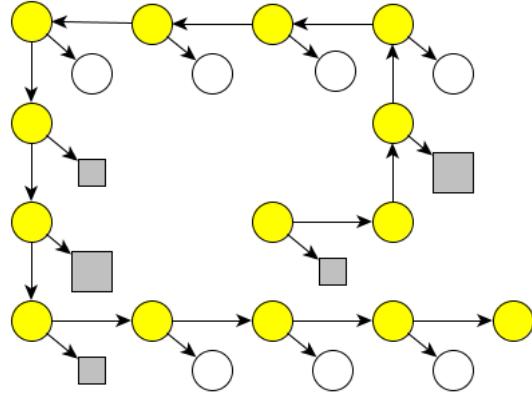


Figure 18: „Rectangular Spiral Layout”

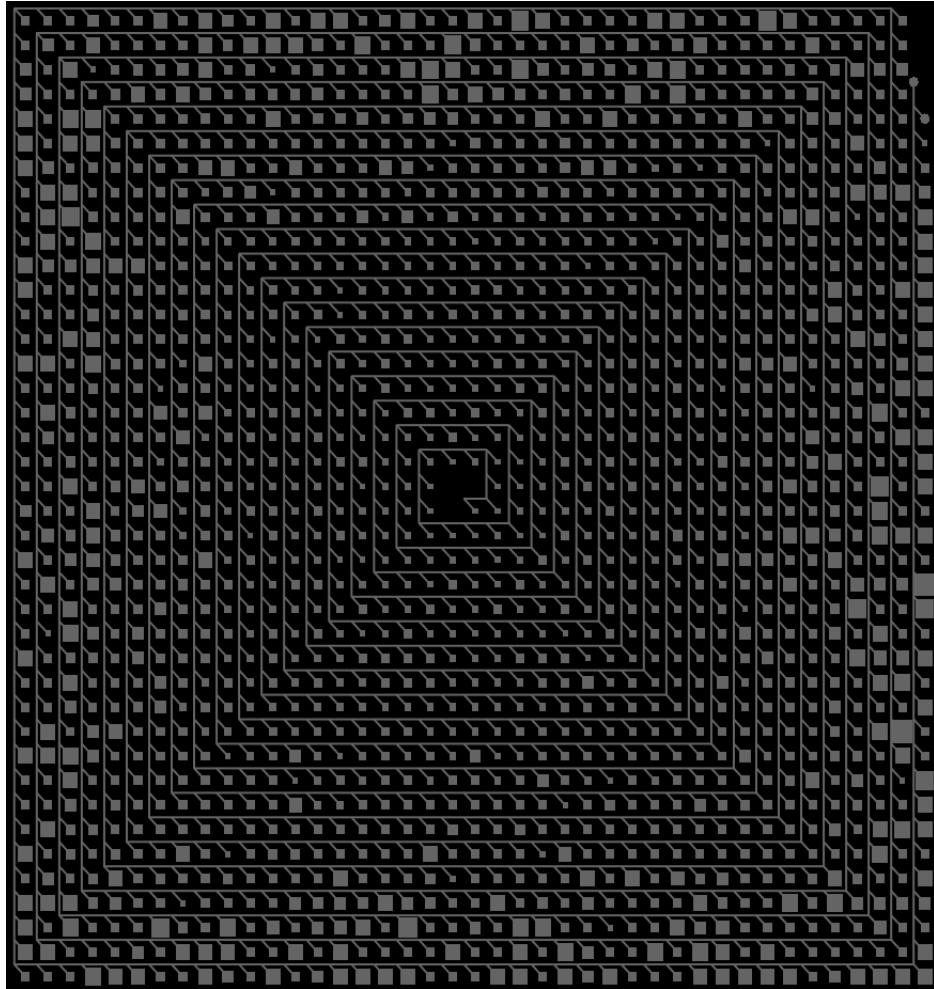
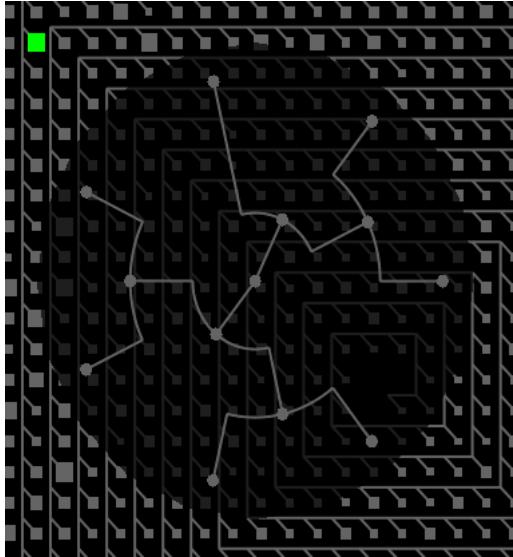


Figure 19: Rectangular spiral Cluster graph visualization

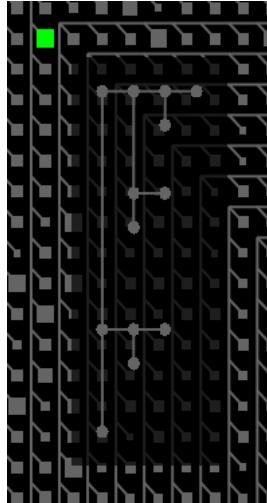
used for initial visualization of the Cluster graph explained earlier in Section 4.1. Both implementations are own made and are not based on any third party source code.

4.3 Gene Ontology Visualization

Graph Ontology graph is directed acyclic and highly connected: 24,153 edges and 10,041 vertices, where 3,918 are unconnected components. Figure 21 shows how extremely connected graph is, picture was produced by yEd graph editing tool.



(a) Polar lens layout



(b) HV-tree lens layout

Figure 20: Different lens layouts

High connectivity between elements in the source graph makes it very complicated to explore. Provided visualization approach has several goals. The first goal is to reduce amount of connections between vertices by showing edges only for „current” sub-graph. It allows to see where sub-graph is aligned in the whole graph and in the same time helps to track it’s inner structure. Second goal is ability to switch from working with genes of the GO graph to discovering relations between GO and Cluster graphs. For this purpose there are two view modes for GO graph:

- levels overview – show graph levels from top to bottom with corresponded content as „preview”;
- zoomed view – visualize only on three levels at the same time to focus on genes inside;

Last but not least goal is to explicitly show nodes and leaves. Each level is divided into two sections where leaves are red in the left and node genes are on the right and have white color. Color schema can be changes through settings menu but not leaves-nodes location. This feature gives further insight into the topology of a specific layer by gaining information about the distribution of leaves and nodes on a particular layer.

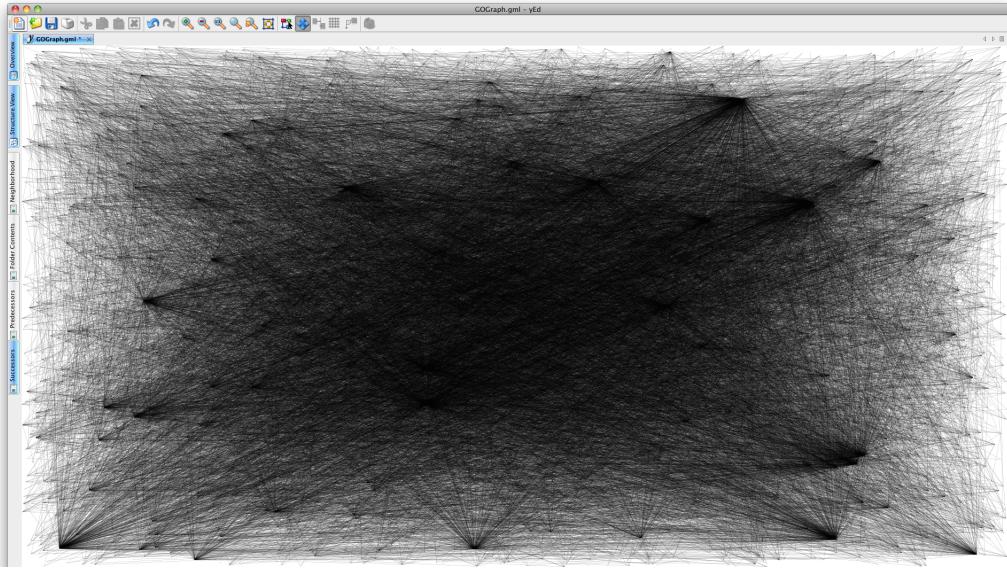


Figure 21: Gene Ontology graph visualization using yEd

There are two layouts implemented based on the goals discussed before.

First GO layout is „Levels Layout” is showed in Figure 22. Genes are ordered by layers depending on their graph-theoretic distance from the root.

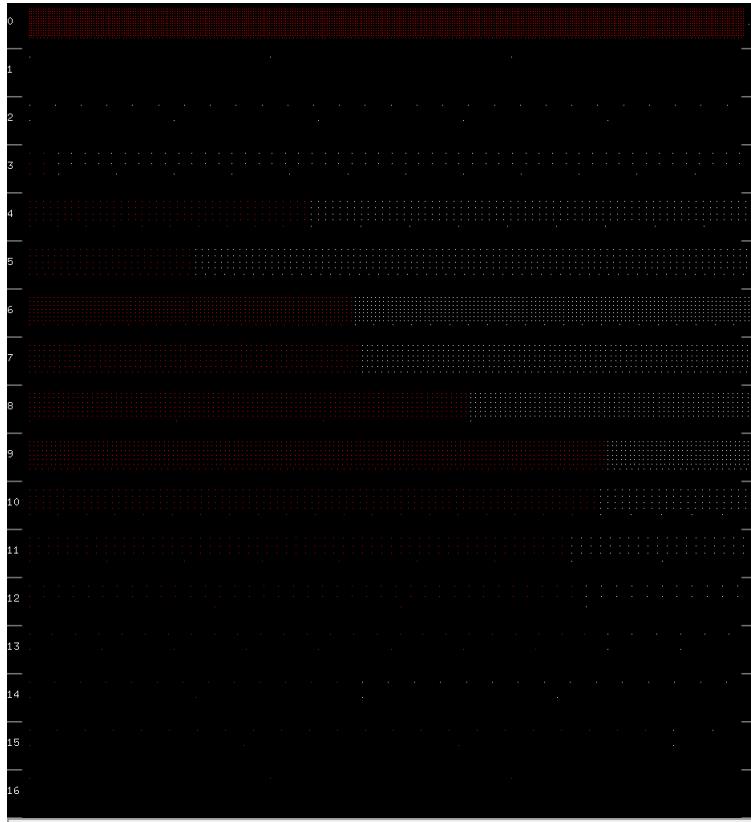


Figure 22: Gene Ontology Levels Layout visualisation

Figure 23 displays the situation if the user zooms in the view. Although the resulting visualization looks to mimic bar charts, the number of leaves cannot be precisely compared between different layers, as the area the red node pixels (leaves) cover is not proportional to the total number of leaves in each layer. But, it is proportional to the sum of nodes in that

particular layer. In other words, the covered area depends on the specific layer density. There are unconnected components in the Gene Ontology graph. Unconnected nodes are placed in the top layer number – zero. There is an option to show-hide unconnected components from the main menu. The spatial arrangement of the node pixels within a layer, except the placing of leaves and nodes in specific regions, is random.

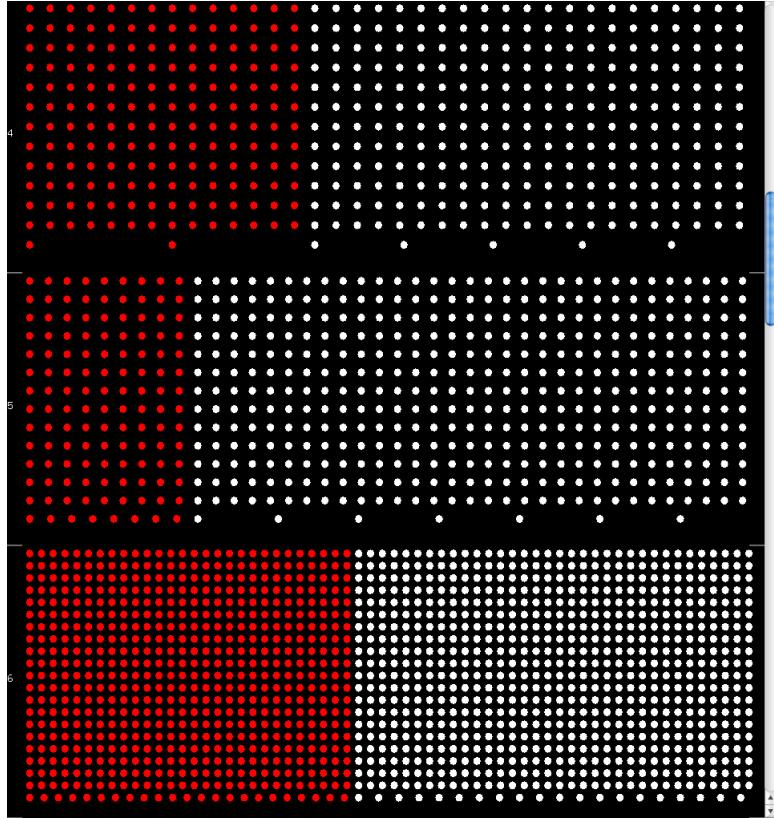


Figure 23: Zoomed view

Second layering approach „Leaves Bottom Layout” (Figure 24) is similar to the first one in terms of placing the nodes into corresponding layers based on the distance from the source node and random distribution of the node pixels within each layer. However, all leaves are placed into one single layer together with unconnected nodes at the bottom of the GO view, i. e., in the layer with the highest number. Unconnected nodes can be filtered out if necessary. This approach gives insight into the distribution of nodes among different layers without the distraction of the leaves, thus enriching the perception of the graph topology. More screen shots can be found in the Appendix D.

All provided features help to improve visualization and decreases amount element in the scene. But even with hided edges visualized sub-tree could be complicated. The result of sub-graph highlighting is showed in Figure 25a.

Improved edge visualization for the same selected vertex „multicellular organismal process” is showed in Figure 25b. A newer solution is simple edge bundling. Edge bundling technique and usage example well described in the [32] and [33].

For all edges which go into same level compute „dummy node” located on the top of the target level and in the middle of outlined vertices, draw single line from source to „dummy node” as first part. Second part is to draw „Bezier line” from „dummy node” to target vertex. This solution helps to follow connection and use space for useful.

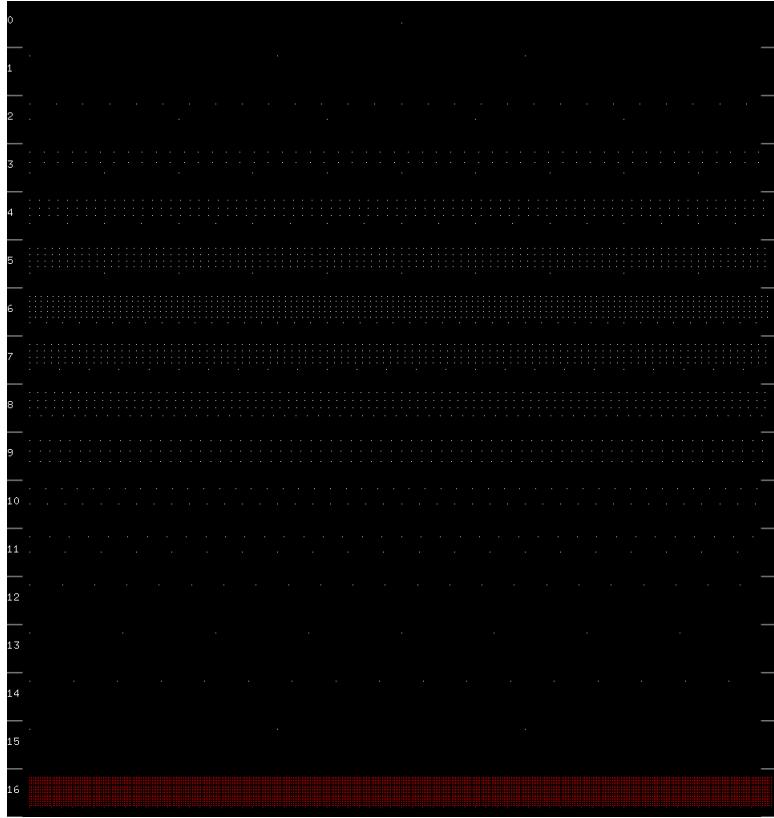


Figure 24: Leaves bottom layout

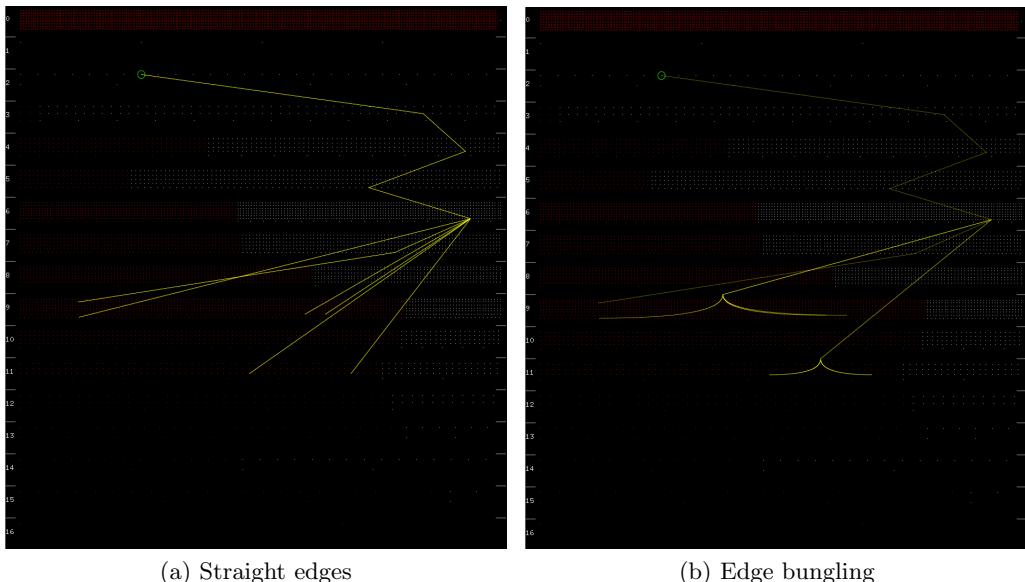


Figure 25: Gene Ontology sub-graph edge highlighting

5 Implementation

Main language and platform for developing thesis is Java. „Java is a programming language and computing platform first released by Sun Microsystems in 1995. It is the underlying technology that powers state-of-the-art programs including utilities, games, and business applications. Java runs on more than 850 million personal computers worldwide, and on billions of devices worldwide, including mobile and TV devices.” [34] Also,

there are a lot of different libraries in Java. Every ‘common’ part of the system could be replaceable. In the future sections would be detailed overview libraries for graphs, graph visualizations and graphic libraries.

Java is flexible platform which has big amount of different libraries. It helps not to write twice things already made but flexibility cause project structure complexity and library management. On the early stage there are no problems to control project throw sophisticated IDE (integrated development environment) but as project complexity grows more powerful building tool need appears. Maven is used during thesis work.

Maven [40] is free, open-source and de-facto project management standard on the Java platform, is part of Apache software project developed and supported by ASF (Apache Software Foundation) [41]. „Maven provides a comprehensive approach to managing software projects. From compilation, to distribution, to documentation, to team collaboration, Maven provides the necessary abstractions that encourage reuse and take much of the work out of project builds.” [42]

„Maven is a set of standards, a repository format, and a piece of software used to manage and describe projects. It defines a standard life cycle for building, testing, and deploying project artifacts. It provides a framework that enables easy reuse of common build logic for all projects following Maven’s standards. The Maven project at the Apache Software Foundation is an open source community which produces software tools that understand a common declarative Project Object Model (POM).” [43]

Maven’s primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal there are several areas of concern that Maven attempts to deal with:

- making the build process easy
- providing a uniform build system
- providing quality project information
- providing guidelines for best practices development
- allowing transparent migration to new features

In Figure 26 showed structure of thesis project folder. Project uses standard Maven project folder format.

Here is overview of key parts: java source code stored in the „src/main/java” folder, tests source code stored in the „src/test/java”. Necessary resources such as log4j configuration file stored in the „src/main/resources”, in the same folder thesis properties file is stored. During „package” phase all resources are copied by Maven.

„Native” directory stores JOGL native libraries to different platform such as Linux, Windows and Mac OS X. In the „lib.zip” stored jars for JUNG library which should be manually placed in the local Maven repository because they do not exist in central Maven repository. All other dependencies are exist in Maven repositories and handled by default dependency process.

Complete builds are stored in the „build” folder. Latest build stored in the „build/latest” folder, other builds stored in the folder that has such name pattern: GoClusterViz-%version%-%build date%-%build time%. Each build folder contains corresponded native build for each of three platforms Linux, Windows and Mac OS X.

Cluster graph and Gene Ontology graphs are stored in the „data” folder.

Second necessary part of the normal development process is revision control. „Revision control, also known as version control or source control (and an aspect of software configuration management or SCM), is the management of changes to documents, programs, and other information stored as computer files. It is most commonly used in software development, where a team of people may change the same files. Changes are usually

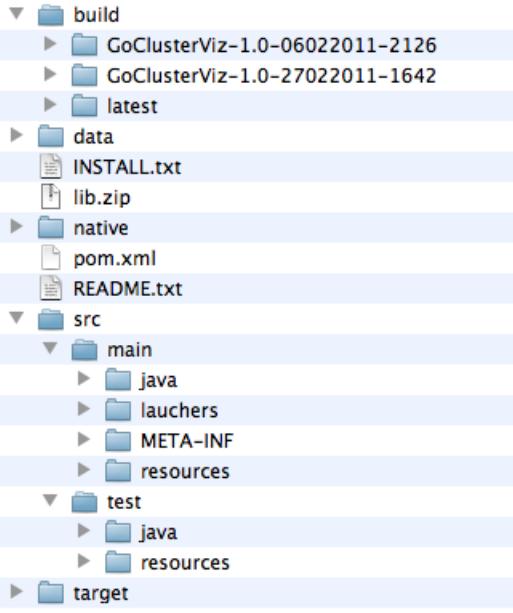


Figure 26: Thesis folder structure

identified by a number or letter code, termed the ‘revision number’, ‘revision level’, or simply ‘revision’. For example, an initial set of files is ‘revision 1’. When the first change is made, the resulting set is ‘revision 2’, and so on. Each revision is associated with a time stamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged.” [44]

During thesis development was used Git version control system. „Git is a distributed revision control system with an emphasis on speed. Git was initially designed and developed by Linus Torvalds for Linux kernel development. Every Git working directory is a full-fledged repository with complete history and full revision tracking capabilities, not dependent on network access or a central server. Git is free software distributed under the terms of the GNU General Public License version 2.” [45]

Git allows to store and work locally on the machine but for safety reasons all source code stored on GitHub. GitHub is a web-based hosting service for software development projects that use the Git revision control system. GitHub provides free hosting for open-source project. Thesis project home page on GitHub [46] and Git repository URL [47].

5.1 Java Graph Libraries Overview

There are overview of a few libraries for working with graphs in Java

1. Java Graph Editing Framework (GEF) [48]

The aim of project consists in generation of library for graph editing, which can be used for construction of high-end (high-quality) custom applications for working with graphs. GEF facilities (opportunities):

- simple and clear design, which allows a developer to expand library’s functionality
- Node-Port-Edge model of graph’s presentation, which permits to perform overwhelming majority of tasks occurring in working with graphs applications
- future XML-based format support (SVG)

2. ILOG JVViews [49]



Figure 27: Commit graph of the local repository made by gitk tool

ILOG JVViews gives (grants) components, aimed for using in custom applications, and also in common with Ajax and Eclipse platform.

3. JGraphT [50]

JGraphT is open source library, which provides with mathematical tool of graphs theory. JGraphT supports different kinds of graphs, including: oriented and unoriented graphs, graphs with weighted/non weighted/nominate (named) or anything else arc format, appointed by user, non upgradeable graphs - supported access to internal graphs in „Read Only” mode. Listenable graphs: allows outer listener to trace events appearance; sub graphs: graphs which are a view about other graphs. Being a powerful feature, JGraphT has been developed as easy and type-safe (with Java code generators use) feature for working with graphs. For example, any object can be node of a graph. You can build graphics on basis of: line, URL, XML documents and so forth, you can even build graphs of graphs.

4. Java Universal Network / Graph Framework (JUNG) [51]

JUNG (Java Universal Network/Graph Framework) – is a software library that pro-

vides a common utilities for the modeling, analysis, and visualization of data that can be represented as a graph or network. It is written in Java, which allows JUNG-based applications to make use of the extensive built-in capabilities of the Java API, as well as those of other existing third-party Java libraries. „The JUNG architecture is designed to support a variety of representations of entities and their relations, such as directed and undirected graphs, multi-modal graphs, graphs with parallel edges, and hyper-graphs. It provides a mechanism for annotating graphs, entities, and relations with meta data. The current distribution of JUNG includes implementations of a number of algorithms from graph theory, data mining, and social network analysis, such as routines for clustering, decomposition, optimization, random graph generation, statistical analysis, and calculation of network distances, flows, and importance measures (centrality, PageRank, HITS, etc.).” [52]

JUNG library is widely used in differ amount of projects. Here is a list of projects using JUNG:

- ExtC: an Eclipse plug-in that is useful for locating large, non-cohesive classes and for recommending how to split them into smaller, more cohesive classes. (Keith Cassell) [53]
- Djinn: a tool for visualizing java artifacts dependencies in a project: jars, directories, packages, classes. (Fabien Benoit) [54]
- Angur: An XML visualization/WYSIWYG Editor (Amir Mohammad shahi) [55]
- RDF Gravity: a tool for visualizing RDF/OWL graphs/ontologies. (Sunil Goyal, Rupert Westenthaler) [56]
- GUESS from HP Labs is a database-driven network analysis tool that provides flexible visualizations, scripting capabilities with Python/Jython, and interfaces with JUNG to let users take advantage of its algorithm library. (Eytan Adar, David Feinberg) [57]
- ADAPTNet is an applet that visualises the families of short oligo microarray probesets associated through common gene transcripts. (Michal Okoniewski, Tim Yates) [58]
- Augur [59] is a visualization tool designed to support the distributed software development process. (Jon Froehlich) [60]
- Ariadne is an Eclipse plug-in (under development) that links technical and social dependencies [61].
- Netsight is a proof-of-concept tool for the visual exploratory data analysis of large-scale network and relational data sets. (Yan-Biao Boey, Joshua O’Madadhain, Scott White, Padhraic Smyth) [62]
- InfoVis CyberInfrastructure provides an unified architecture in which diverse data analysis, modeling and visualization algorithms can be plugged in and run. [63]
- PWComp is a graph comparative metabolic pathway tool. (Joshua Adelman, Josh England, Alex Chen) [64]
- Google Cartography, featured in Google Hacks, uses the Google Search API to build a visual representation of the interconnectivity of streets in an area. (Richard Jones) [65]
- GINY is a project with similar aims to that of JUNG, which contains some code derived from JUNG. (Rowan Christmas) [66]
- GraphExplore is a JAVA application that renders networks of objects in a graphical form, which uses modified forms of the JUNG layout algorithm implementations. (Quanli Wang) [67]

- TOTEM (TOolbox for Traffic Engineering Methods) provides a framework where researchers can integrate their traffic engineering algorithms. These algorithms can therefore be applied on models of real networks. The TOTEM toolbox also gives network operators the opportunity to experiment the currently developed traffic engineering algorithms on their own network. Today, the TOTEM toolbox already federates a large set of traffic engineering algorithms published in the scientific literature. This project uses JUNG for the graphical representation of the network topology. (S. Balon, O. Delcourt, J. Lepropre and F. Skivee) [68]
- D2K ("Data to Knowledge") is a visual programming environment for building complicated data mining applications; T2K is a library of D2K modules that implements sophisticated algorithms for text analysis. Each of these uses JUNG for network visualization. [69]
- graphBuilder is an application that allows users to build network representations of relational databases and data files. It has been designed as a tool for exploring online scientific data repositories. (Ben Raymond) [70]
- Semiophore is an application for exploring large graphs where there are many variables on both nodes and links (including time-based/event variables). It works with a relational database. It provides several visualization approaches. One of them is based on JUNG. It provides several analysis routines, featuring SNA measures among them. User can interact with the network : dynamic multi-variables filtering, dynamic aggregation, network editing and production of quicktime videos from longitudinal analysis are possible. Semiophore can handle text/XML documents with NLP information extraction and text summarization routines [English and French support only] in order to automatically build network maps of actors/information. [71]
- Xholon uses JUNG to represent and visualize networks such as biochemical pathways and models (screenshots). (Ken Webb) [72]
- Flink is a website presenting the social networks and research activity of Semantic Web researchers based on a number of sources (web pages, publication databases, email archives, FOAF data). Flink uses JUNG for network representation and visualization as well as for computing network measures. Flink has won 1st prize at the Semantic Web Challenge [74] of 2004. (Peter Mika) [73]
- T-Prox(approve sites) is a proxy, designed to be used for usability analyzes of websites. It uses JUNG to visualize the users path through the site. (Sven Lilienthal) [75]
- Simple C-K Editor is a visualisation tool built on the C-K Design Theory. Its main purpose is to provide an easy tool to create, manipulate, edit and print C-K diagrams. [76]
- PCOPGene is web-based application to analyse microarray data with large sample-series. The user can identify several kinds of non-linear expression relationships inside the gene network, study the expression dependence fluctuations in detail, and crossing the results with external biomedical data-servers. [77]

There are a lot more graph visualization frameworks for Java: Piccolo [81], The Visualization Toolkit (VTK) [82], The InfoVis Toolkit [83], Improvise [84]. All of them can be used as for storing and visualizing graphs and networks.

In the scope of current thesis work was used JUNG graph library to store graph structures.

5.2 OpenGL Visualization Standard

OpenGL is a software interface to graphics hardware. This interface consists of about 120 distinct commands, which you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed to work efficiently even if the computer that displays the graphics you create isn't the computer that runs your graphics program.

„OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of geometric primitive - points, lines, and polygons.” [78]

There are many wrappers over OpenGL for developing on different programming languages. One of the favorite in the Java community are JOGL and LWJGL.

„The Lightweight Java Game Library (LWJGL) is a solution aimed directly at professional and amateur Java programmers alike to enable commercial quality games to be written in Java. LWJGL provides developers access to high performance cross platform libraries such as OpenGL and OpenAL (Open Audio Library) allowing for state of the art 3D games and 3D sound. Additionally LWJGL provides access to controllers such as Gamepads, Steering wheel and Joysticks. All in a simple and straight forward API.

LWJGL is available under a BSD license, which means it's open source and freely available at no charge.” [79]

In the thesis used JOGL wrapper library over OpenGL. Java OpenGL (JOGL) is a wrapper library that allows OpenGL to be used in the Java programming.

„The base OpenGL C API and associated operation system API, are accessed in JOGL via Java Native Interface (JNI) calls. As such, the underlying system must support OpenGL for JOGL to work. JOGL differs from some other Java OpenGL wrapper libraries in that it merely exposes the procedural OpenGL API via methods on a few classes, rather than trying to map OpenGL functionality onto the object-oriented programming paradigm. Indeed, most of the JOGL code is auto generated from the OpenGL C header files via a conversion tool named GlueGen, which was programmed specifically to facilitate the creation of JOGL.

This design decision has both its advantages and disadvantages. The procedural and state machine nature of OpenGL is inconsistent with the typical method of programming under Java, which is bothersome to many programmers. However, the straightforward mapping of the OpenGL C API to Java methods makes conversion of existing C applications and example code much simpler. The thin layer of abstraction provided by JOGL makes runtime execution quite efficient, but accordingly is more difficult to code compared to higher-level abstraction libraries like Java3D. Because most of the code is auto generated.” [80]

5.3 Program architecture

An overview on the tool's architecture is given in Figure 28. The implementation is divided into several modules specialized for various tasks. The IO module implements data loading from gml files. The data is stored in extended gml file format, which contains additional properties for nodes, such as the node label. The Graph Core module extends the JUNG graph model [51] in order to fit it to our requirements. The implementation of Swing GUI and OpenGL user interactions is realized by the User Interaction module. The

Graph Visualization module and its submodules contain all code for the whole visualization process, including our own layout implementation, primitive drawing abstraction, and program state machine. One of the most important modules of is of course Subgraph Extraction that contains the implementation of the subtree calculation algorithm.

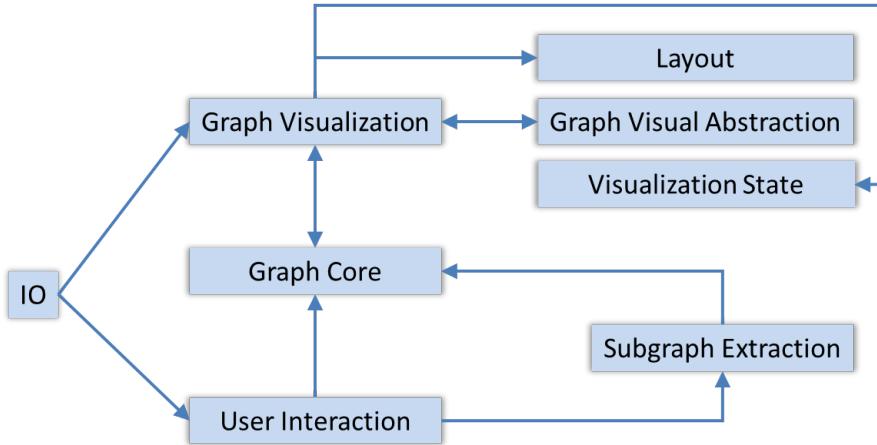


Figure 28: Module architecture of GoClusterViz tool

One of the main module in the program is Subgraph Extraction module. Extraction algorithm was explained in the Section 3 and in the Appendix A is pseudo-code, implementation is in the class `se.lnu.thesis.algorithm.Extractor`.

Complete UML class diagram of the GoClusterViz can be found in the Appendix C. Here is a brief overview of the package content.

Main class and program entry point is class – `GoClusterViz`.

Package `se.lnu.thesis.gui` contains all interface abstractions and Swing implementations for main window, menus and event handling.

`se.lnu.thesis.element` is hierarchy for graph visualization. It is implementation of the Composite pattern: „In software engineering, the composite pattern is a partitioning design pattern. The composite pattern describes that a group of objects are to be treated in the same way as a single instance of an object. The intent of a composite is to ‘compose’ objects into tree structures to represent part-whole hierarchies. Implementing the composite pattern lets clients treat individual objects and compositions uniformly.” [85]

Package `se.lnu.thesis.paint` contains all drawing implementations. Graph elements do not implement visualization shape, for that purpose there is `ElementVisualizer` abstraction. Diagram in Figure 30 shows complete UML diagram of the package `se.lnu.thesis.paint.visualizer`, there are several different shape visualizer. To reduce memory usage all visualizers are stored in the `ElementVisualizerFactory` – Flyweight design pattern.

„Flyweight is a software design pattern. A flyweight is an object that minimizes memory use by sharing as much data as possible with other similar objects; it is a way to use objects in large numbers when a simple repeated representation would use an unacceptable amount of memory. The term is named after the boxing weight class. Often some parts of the object state can be shared and it’s common to put them in external data structures and pass them to the flyweight objects temporarily when they are used.” [86]

Flyweight pattern is used to store different kind of visualizers for graph elements. It means for each element of the graph there is a reference to visualizer object containing its shape, metrics, and other formatting data, which reduce amount of data to hundreds or thousands of bytes for each character. For every element of the graph there is a reference to a flyweight visualizer object shared by every instance of the same graph element in the graph; only the position of each element and current state (selected, highlighted, focused) is stored internally.

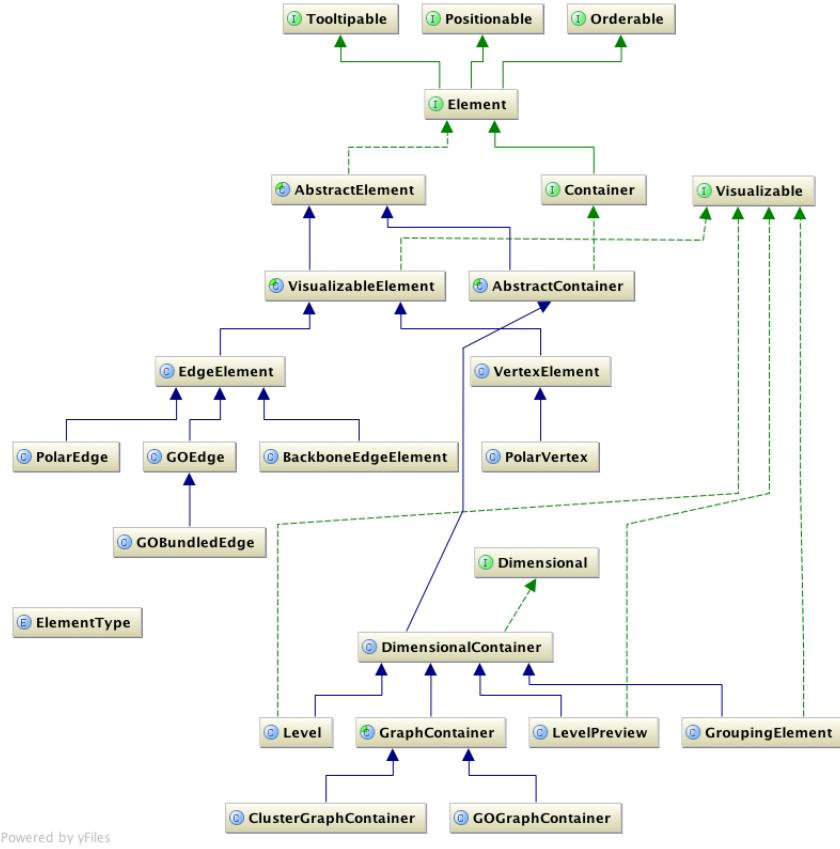


Figure 29: Graph elements visualisation hierarchy abstraction

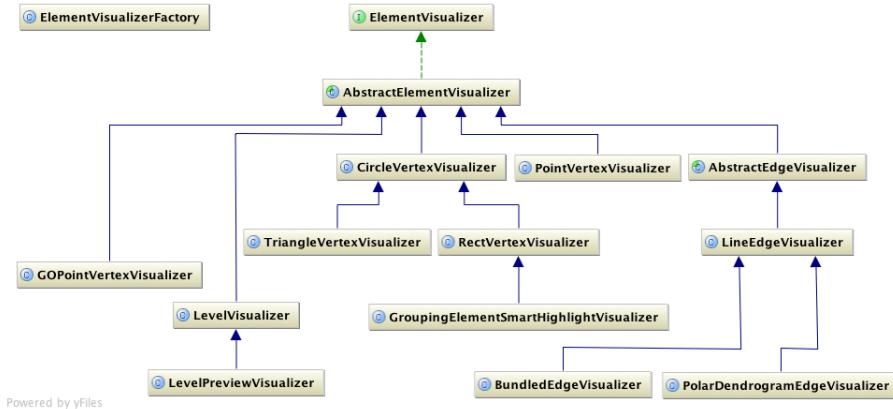


Figure 30: Flyweight element visualizers

User interaction state machine is implementation of the State software development pattern located in the package `se.lnu.thesis.paint.state`.

,,A monolithic objects behavior is a function of its state, and it must change its behavior at run-time depending on that state. Or, an application is characterized by large and numerous case statements that vector flow of control based on the state of the application. The State pattern does not specify where the state transitions will be defined. The choices are two: the context object, or each individual State derived class. The advantage of the latter option is ease of adding new State derived classes. The disadvantage is each State derived class has knowledge of (coupling to) its siblings, which introduces dependencies between subclasses. A table-driven approach to designing finite state machines does a

good job of specifying state transitions, but it is difficult to add actions to accompany the state transitions. The pattern-based approach uses code (instead of data structures) to specify state transitions, but it does a good job of accommodating state transition actions.” [87]

It means that graph has several states: normal view, zoomed view, lens is on the scene, etc. It releases the code from the long „switch” statements and determines logic into different classes.

Layout implementations are in the `se.lnu.thesis.layout` package 31.

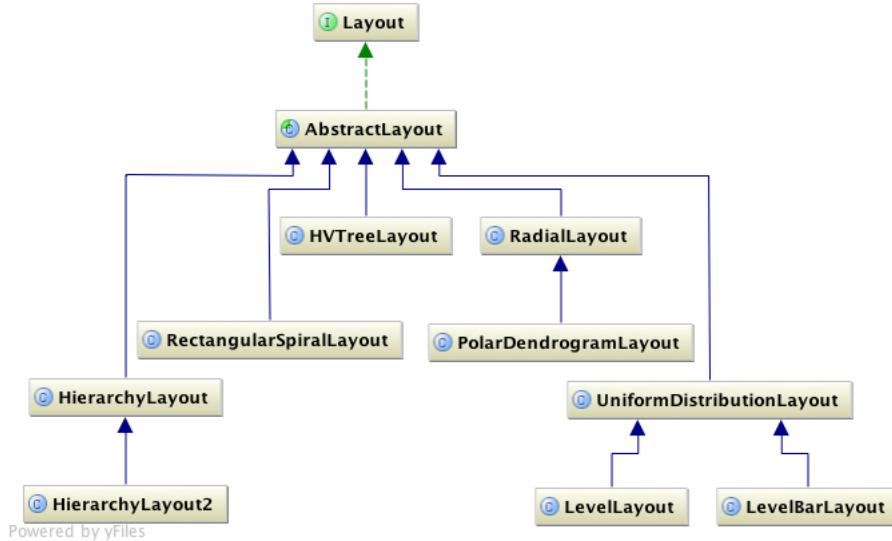


Figure 31: Layouts class diagram

6 Conclusion and Future Work

We presented a new method for the combined visualization of an ontology (represented as DAG) and an hierarchical clustering (represented as tree) of one data set. The proposed method interactively visualizes all the data without scrolling, thereby presenting an complete overview. It also allows for interactive selection and navigation to explore the data. We have showed that the program is able to tackle the problem in our research focus, i.e., the visualization and visual mapping between two huge and conceptually different data sets. However, there are some improvements that should be performed in the future.

As explained previously, the zoomed-in view for the GO shows three levels at the same time while displaying the subgraph by highlighting the nodes only. The edges are omitted due to clutter problems that can occur since edges from a higher level might go through the zoomed-in view to nodes in the lower layers. Since we are zoomed in, this does no make sense to show, because we have no insight from which layer those edges are coming from, nor to which layer they are going to. However, an improvement is possible by showing only edges between the three layers shown in the zoomed-in GO view. At the same time, the edge bundling algorithm could also be improved.

Thesis application is not general visualization tool. It has been developed specially for biology scientist and driven by their requirements which means that the application fulfills their needs. There are no application with same functionality. Visualization technique developed during the work is based on provided data, uses its specific structure as advance. On the other hand the visualization approach may apply to another fields.

As for future improved version of the rectangular spiral metaphor it is possible to provide nested spiral trees: the idea is to draw smaller similar spirals instead of aggregating

sub-trees into rectangles. For smaller sub-trees this approach will help to see the structure without lens.

For better user experience it is possible to provide animation for different events: lens appearing and hiding in the cluster graph, smooth level scrolling in the Gene Ontology visualization, step-by-step elements highlighting.

References

- [1] VLSI is available on the web <http://en.wikipedia.org/wiki/Very-large-scale-integration>. Last access in August 2011.
- [2] Graph drawing general information is available on the web http://en.wikipedia.org/wiki/Graph_drawing. Last access in August 2011.
- [3] P. Imrich, K. Mueller, D. Imre, A. Zelenyuk, And W. Zhu, Interactive Poster: A Hardware-Accelerated Rubbersheet Focus + Context Technique For Radial Dendograms , IEEE Information Visualization Symposium '03 Seattle, October 2003
- [4] Introduction into Dengrograms is available on the web en.wikipedia.org/wiki/Dendrogram. Last access in August 2011.
- [5] T. Barlow and P. Neville, A comparison of 2D visualization of hierarchies, Information Visualization pp 131-138, 2001.
- [6] M. Kreussler and H. Schumann, A flexible approach for visual data mining, IEEE Trans. VisualGraphics, vol. 8, no. 1, pp. 39-51, 2002.
- [7] J. Yang, M. Ward, and E. Rundensteiner, InterRing: An interactive tool for visually navigating and manipulating hierarchical structures, IEEE 2002 Symposium on Information Visualization, pp. 77-92, 2002.
- [8] Biotech is available on the web <http://biotech.icmb.utexas.edu/pages/bioinfo.html>. Last access in August 2011.
- [9] Bioinformatic definition information is is available on the web <http://www.absoluteastronomy.com/topics/Bioinformatics>. Last access in August 2011.
- [10] Gene Ontology project homepage is available on the web <http://www.geneontology.org/>. Last access in August 2011.
- [11] The Open Biomedical Ontologies is available on the web <http://www.obofoundry.org/>. Last access in August 2011.
- [12] G. Gan, C. Ma and J. Wu, Data Clustering. Theory, Algorithms, and Applications, Society for Industrial and Applied Mathematics pp 3, 2007.
- [13] Prof. Dr. Andreas Kerren home page is available on the web <http://homepage.lnu.se/staff/akemsi/>. Last access in August 2011.
- [14] Prof. Dr. Falk Schreiber home page is available on the web <http://bic-gh.ipk-gatersleben.de/~schreibe/>. Last access in August 2011.
- [15] ISOVIS research group home page is available on the web <http://cs.lnu.se/isovis/>. Last access in August 2011.
- [16] Plant Bioinformatic research Group home page is available on the web <http://nwg.bic-gh.de/>. Last access in August 2011.
- [17] Graphlet project homepage is available on the web <http://www.fmi.uni-passau.de/Graphlet/>. Last access in August 2011.
- [18] GML file format explanation is available on the web <http://www.infosun.fim.uni-passau.de/Graphlet/GML/gml-tr.html>. Last access in August 2011.
- [19] GML supported programs is available on the web http://en.wikipedia.org/wiki/Graph_Modelling_Language. Last access in August 2011.

- [20] yEd project homepage is available on the web http://www.yworks.com/en/products_yed_about.html. Last access in August 2011.
- [21] Clairlib is available on the web <http://www.clairlib.org/>. Last access in August 2011.
- [22] Cytoscape is available on the web <http://en.wikipedia.org/wiki/Cytoscape>. Last access in August 2011.
- [23] NetworkX is available on the web <http://en.wikipedia.org/wiki/NetworkX>. Last access in August 2011.
- [24] ocamlgraph is available on the web <http://ocamlgraph.lri.fr/>. Last access in August 2011.
- [25] OGDF is available on the web <http://www.ogdf.net/>. Last access in August 2011.
- [26] Tulip is available on the web <http://tulip.labri.fr/TulipDrupal/>. Last access in August 2011.
- [27] GraphML specification is available on the web <http://graphml.graphdrawing.org/>. Last access in August 2011.
- [28] DOT graph language is available on the web http://en.wikipedia.org/wiki/DOT_language. Last access in August 2011.
- [29] GXL introduction is available on the web <http://www.gupro.de/GXL/Introduction/section1.html>. Last access in August 2011.
- [30] SVG information is available on the web http://en.wikipedia.org/wiki/Scalable_Vector_Graphics. Last access in August 2011.
- [31] Graph Drawing Conference '95 is available on the web www.informatik.uni-trier.de/~ley/db/conf/gd/gd95.html. Last access in August 2011.
- [32] Danny Holten, Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data, IEEE Transactions On Visualization and Computer Graphics 2006, Volume 12, Number 5
- [33] Danny Holten, Jarke J. van Wijk, Force-Directed Edge Bundling for Graph Visualization, Eurographics/ IEEE-VGTC Symposium on Visualization 2009, Volume 28 (2009), Number 3
- [34] Java FAQ is available on the web http://www.java.com/en/download/faq/whatis_java.xml. Last access in August 2011.
- [35] Binary Tree definition is available on the web http://www.wordiq.com/definition/Binary_tree. Last access in August 2011.
- [36] Java 2D reference is available on the web <http://java.sun.com/products/java-media/2D/reference/index.html>. Last access in August 2011.
- [37] Java AWT overview is available on the web http://en.wikipedia.org/wiki/Abstract_Window_Toolkit. Last access in August 2011.
- [38] Java SWING overview is available on the web [http://en.wikipedia.org/wiki/Swing_\(Java\)](http://en.wikipedia.org/wiki/Swing_(Java)). Last access in August 2011.
- [39] AWT vs SWING comparison is available on the web <http://edn.embarcadero.com/article/26970>. Last access in August 2011.

- [40] Maven project home page is available on the web <http://maven.apache.org/>. Last access in August 2011.
- [41] Apache Foundation home page is available on the web <http://www.apache.org/>. Last access in August 2011.
- [42] Vincent Massol, Jason van Zyl, Better builds with Maven, Mergere Library Press, pp. 16, 2006.
- [43] Vincent Massol, Jason van Zyl, Better builds with Maven, Mergere Library Press, pp. 18, 2006.
- [44] Revision Control definition is available on the web http://en.wikipedia.org/wiki/Revision_control. Last access in August 2011.
- [45] Git overview is available on the web [http://en.wikipedia.org/wiki/Git_\(software\)](http://en.wikipedia.org/wiki/Git_(software)). Last access in August 2011.
- [46] Project home page on GitHub is available on the web <http://github.com/vadyalex/thesis>. Last access in August 2011.
- [47] Project Git repository is available on the web <git://github.com/vadyalex/thesis.git>. Last access in August 2011.
- [48] Java Graph Editing Framework is available on the web <http://gef.tigris.org/>. Last access in August 2011.
- [49] ILOG Jview is available on the web <http://www.ilog.com/products/jviews/>. Last access in August 2011.
- [50] JGraphT is available on the web <http://jgrapht.sourceforge.net/>. Last access in August 2011.
- [51] Java Universal Network / Graph Framework (JUNG) is available on the web <http://jung.sourceforge.net/>. Last access in August 2011.
- [52] JUNG graph library overview is available on the web <http://jung.sourceforge.net/index.html>. Last access in August 2011.
- [53] ExtC is available on the web <http://code.google.com/p/ext-c/>. Last access in August 2011.
- [54] Djinn is available on the web <http://www.jnovation.net/djinn>. Last access in August 2011.
- [55] Angur is available on the web <http://angur.sourceforge.net/>. Last access in August 2011.
- [56] RDF Gravity is available on the web <http://semweb.salzburgresearch.at/apps/rdf-gravity/index.html>. Last access in August 2011.
- [57] GUESS is available on the web <http://www.hpl.hp.com/research/idl/projects/graphs/index.html>. Last access in August 2011.
- [58] ADAPTNet is available on the web <http://bioinformatics.picr.man.ac.uk/adaptnet>. Last access in August 2011.
- [59] Cleidson de Souza, Jon Froehlich, Paul Dourish, Seeking the Source: Software Source Code as a Social and Technical Artifact, GROUP '05 Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work

- [60] Jon Froehlich web page is available on the web <http://www.cs.washington.edu/homes/jfroehli/>. Last access in August 2011.
- [61] Ariadne project homepage is available on the web <http://awareness.ics.uci.edu/~ariadne/>. Last access in August 2011.
- [62] Netsight is available on the web <http://jung.sourceforge.net/netsight>. Last access in August 2011.
- [63] InfoVis CuberInfrastructure is available on the web <http://iv.slis.indiana.edu/sw/>. Last access in August 2011.
- [64] PWComp project homepage is available on the web <http://www.ocf.berkeley.edu/~jadelman/pwcomp/index.html>. Last access in August 2011.
- [65] Google Cartography is available on the web <http://richard.jones.name/google-hacks/google-cartography/google-cartography.html>. Last access in August 2011.
- [66] GINY is available on the web <http://csbi.sourceforge.net/>. Last access in August 2011.
- [67] GraphExplorer is available on the web <http://graphexplore.cgt.duke.edu/>. Last access in August 2011.
- [68] TOTEM is available on the web <http://totem.run.montefiore.ulg.ac.be/>. Last access in August 2011.
- [69] D2K is available on the web <http://alg.ncsa.uiuc.edu/do/downloads>. Last access in August 2011.
- [70] graphBuilder project homepage is available on the web <http://data.aad.gov.au/analysis/gb.cfm>. Last access in August 2011.
- [71] Semiophore is available on the web <http://www.semiophore.net/>. Last access in August 2011.
- [72] Xholon is available on the web <http://sourceforge.net/projects/xholon/>. Last access in August 2011.
- [73] Flink is available on the web <http://flink.semanticweb.org/>. Last access in August 2011.
- [74] Semantic Web Challenge introduction is available on the web <http://challenge.semanticweb.org/>. Last access in August 2011.
- [75] T-Prox is available on the web <http://www.t-prox.net/>. Last access in August 2011.
- [76] Simple C-K Editor introduction is available on the web <http://code.google.com/p/ckeditor>. Last access in August 2011.
- [77] PCOPGene is available on the web <http://revolutionresearch.uab.es/>. Last access in August 2011.
- [78] Dave Shreiner, Mason Woo, Jackie Neider, Tom Davis, OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2.1 (6th Edition), Addison-Wesley Professional, 2007.
- [79] LWJGL is available on the web <http://www.lwjgl.org/>. Last access in August 2011.

- [80] JOGL is available on the web <http://jogamp.org/>. Last access in August 2011.
- [81] Piccolo is available on the web <http://www.cs.umd.edu/hcil/jazz/>. Last access in August 2011.
- [82] Visualisation Toolkit is available on the web <http://www.vtk.org/>. Last access in August 2011.
- [83] InfoVis Toolkit is available on the web <http://ivtk.sourceforge.net/>. Last access in August 2011.
- [84] Improvise project homepage is available on the web <http://www.cs.ou.edu/~weaver/improvise/index.html>. Last access in August 2011.
- [85] Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley., 1995, pp. 395.
- [86] Flyweight design pattern explanation is available on the web http://en.wikipedia.org/wiki/Flyweight_pattern. Last access in August 2011.
- [87] State design pattern explanation is available on the web http://sourcemaking.com/design_patterns/state. Last access in August 2011.

Appendix A: Listing of a Complex GML File

```
Creator "yFiles"
Version "2.8"
graph
[
    hierachic 1
    label ""
    directed 1
    node
    [
        id 0
        label "1"
        graphics
        [
            x 360.0
            y 67.0
            w 30.0
            h 30.0
            type "rectangle"
            fill "#FFCC00"
            outline "#000000"
        ]
        LabelGraphics
        [
            text "1"
            fontSize 13
            fontName "Dialog"
            anchor "c"
        ]
    ]
    node
    [
        id 1
        label "2"
        graphics
        [
            x 284.0
            y 150.0
            w 30.0
            h 30.0
            type "rectangle"
            fill "#FFCC00"
            outline "#000000"
        ]
        LabelGraphics
        [
            text "2"
            fontSize 13
            fontName "Dialog"
            anchor "c"
        ]
    ]
    node
    [
        id 2
        label "3"
        graphics
        [
            x 450.0
            y 150.0
            w 30.0
            h 30.0
            type "rectangle"
        ]
    ]
]
```

```

        fill "#FFCC00"
        outline "#000000"
    ]
    LabelGraphics
    [
        text "3"
        fontSize 13
        fontName "Dialog"
        anchor "c"
    ]
]
node
[
    id 3
    label "4"
    graphics
    [
        x 391.0
        y 270.0
        w 30.0
        h 30.0
        type "rectangle"
        fill "#FFCC00"
        outline "#000000"
    ]
    LabelGraphics
    [
        text "4"
        fontSize 13
        fontName "Dialog"
        anchor "c"
    ]
]
edge
[
    source 0
    target 1
    graphics
    [
        fill "#000000"
        targetArrow "standard"
    ]
]
edge
[
    source 0
    target 2
    graphics
    [
        fill "#000000"
        targetArrow "standard"
    ]
]
edge
[
    source 2
    target 3
    graphics
    [
        fill "#000000"
        targetArrow "standard"
    ]
]
]

```

Appendix B: Subgraph Extraction Algorithm

```

// selected vertex in the Gene Ontology by user
GO_vertex;

if GO_vertex is in cache {
    // already computed, load from cache
    load from cache GO_subgraph and Cluster_subgraph;
    return;
}

// extract subgraph using non-recursive DFS
// starting from vertex GO_vertex as root
GO_subgraph = extractSubgraph(GO_graph, GO_vertex);
save GO_subgraph to cache with key GO_vertex

// create empty subgraph
Cluster_subtree = new Graph;

for each vertex in GO_subgraph {
    if vertex is leaf {
        // leaf labels are the same for both graphs,
        // but vertex objects are different
        l_label = GO_graph.getLabel(vertex);
        leaf = Cluster_graph.getVertexByLabel(l_label);
        // get all connected vertices
        // from current leaf up to the Cluster root
        connectedVertices = invertDFS(Cluster_graph, leaf);

        // add connected vertices to cluster subtree
        // create edges
        for (int i=0; i<=connectedVertices.size()-1; i++) {
            node1, node2 = null;
            node1 = connectedNodes.get(i);
            Cluster_subtree.addVertex(node1);

            if (i+1<=connectedNodes.size()-1) {
                node2 = connectedNodes.get(i+1);

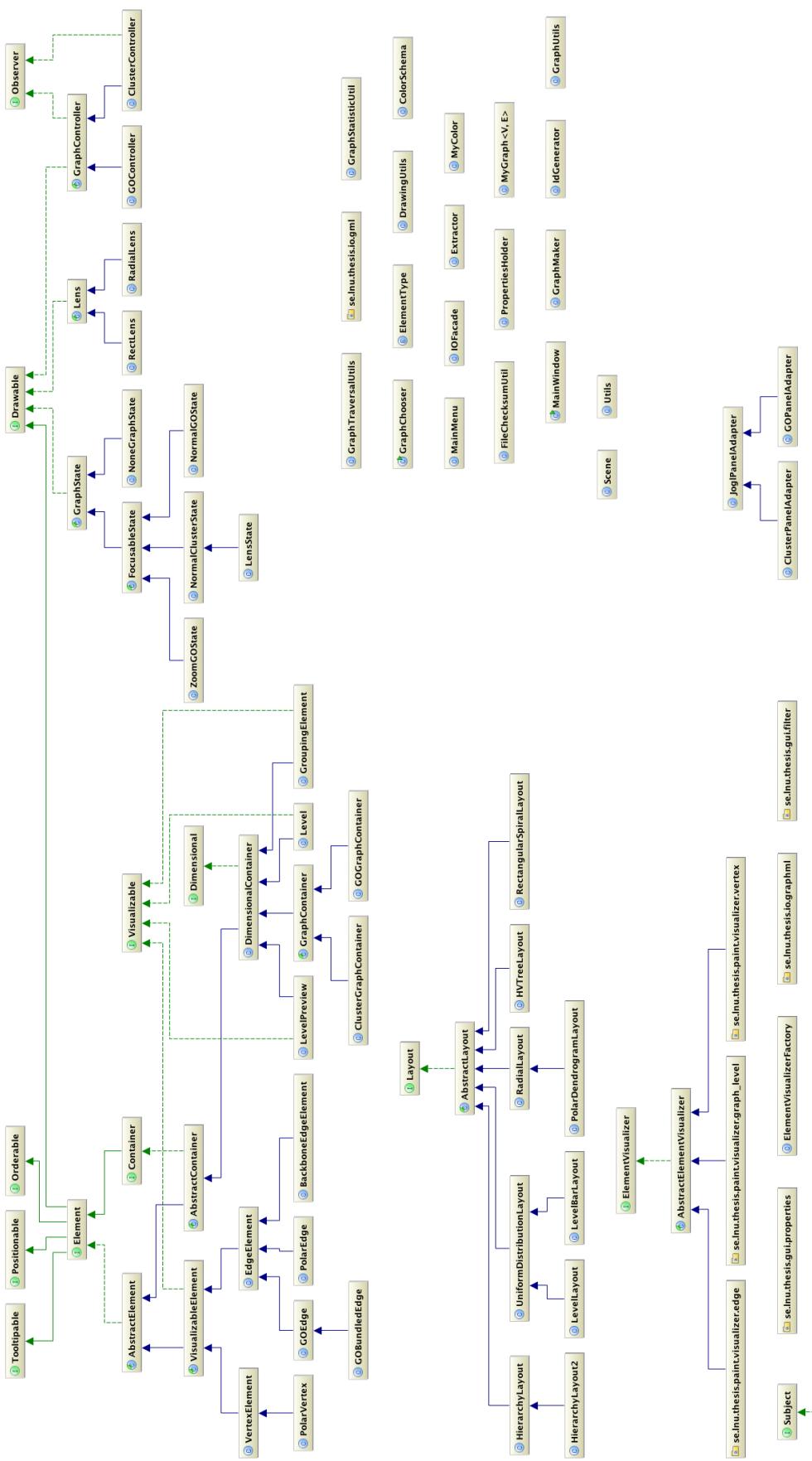
                Cluster_subtree.addVertex(node2);
                Cluster_subtree.addEdge(node2, node1);
            }
        }

        // find lowest common root for subgraph
        // and remove vertices from the root to common root
        removeRootChain(Cluster_graph, Cluster_subtree);
    }
}

save Cluster_subtree to cache with key GO_vertex

```

Appendix C: Complete UML Class Diagram



Appendix D: Program Execution Screenshots

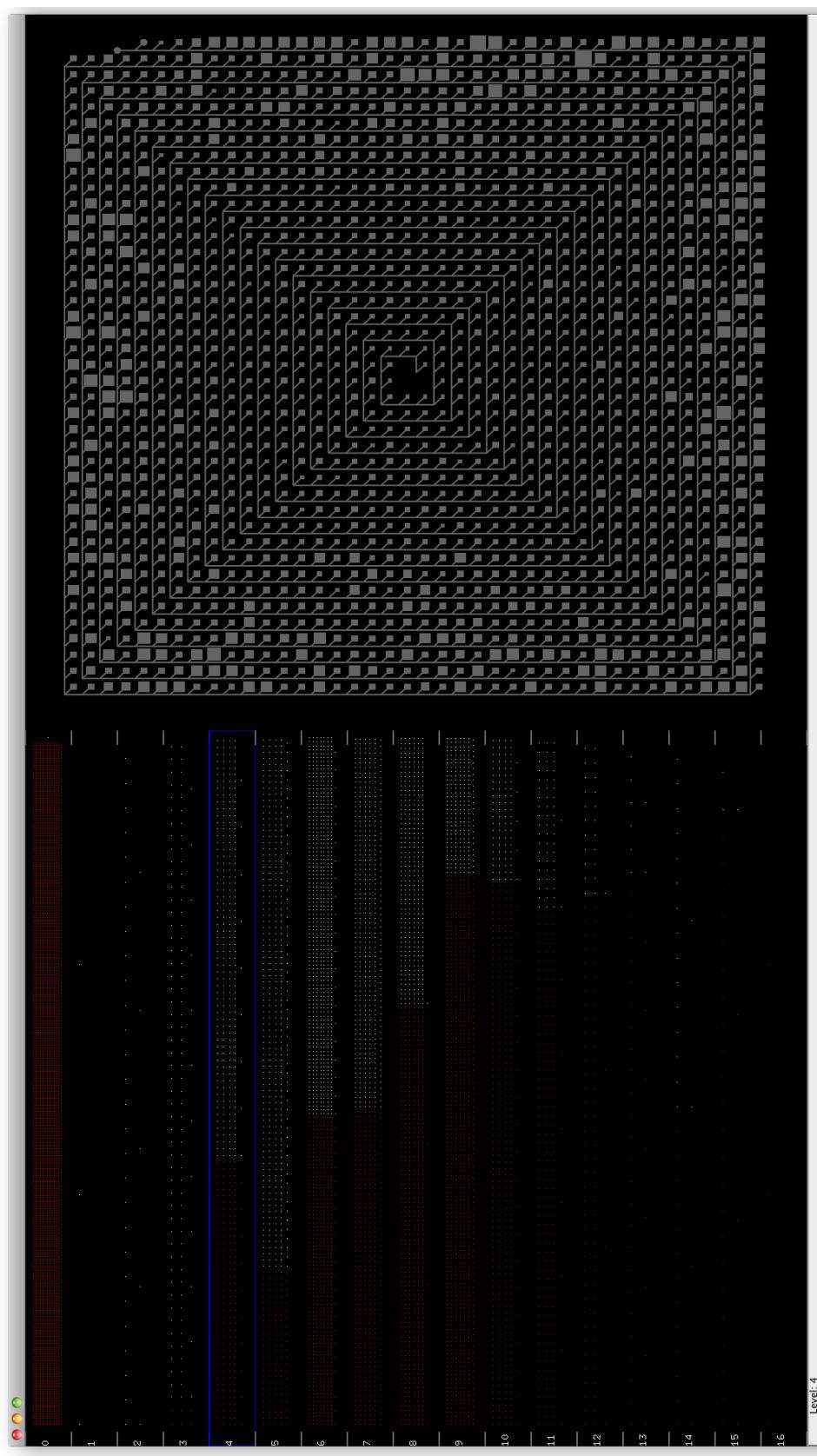


Figure D.2: Visualization of the Gene Ontology graph by levels and Cluster analysis result graph. Blue rectangle – highlighted level 4

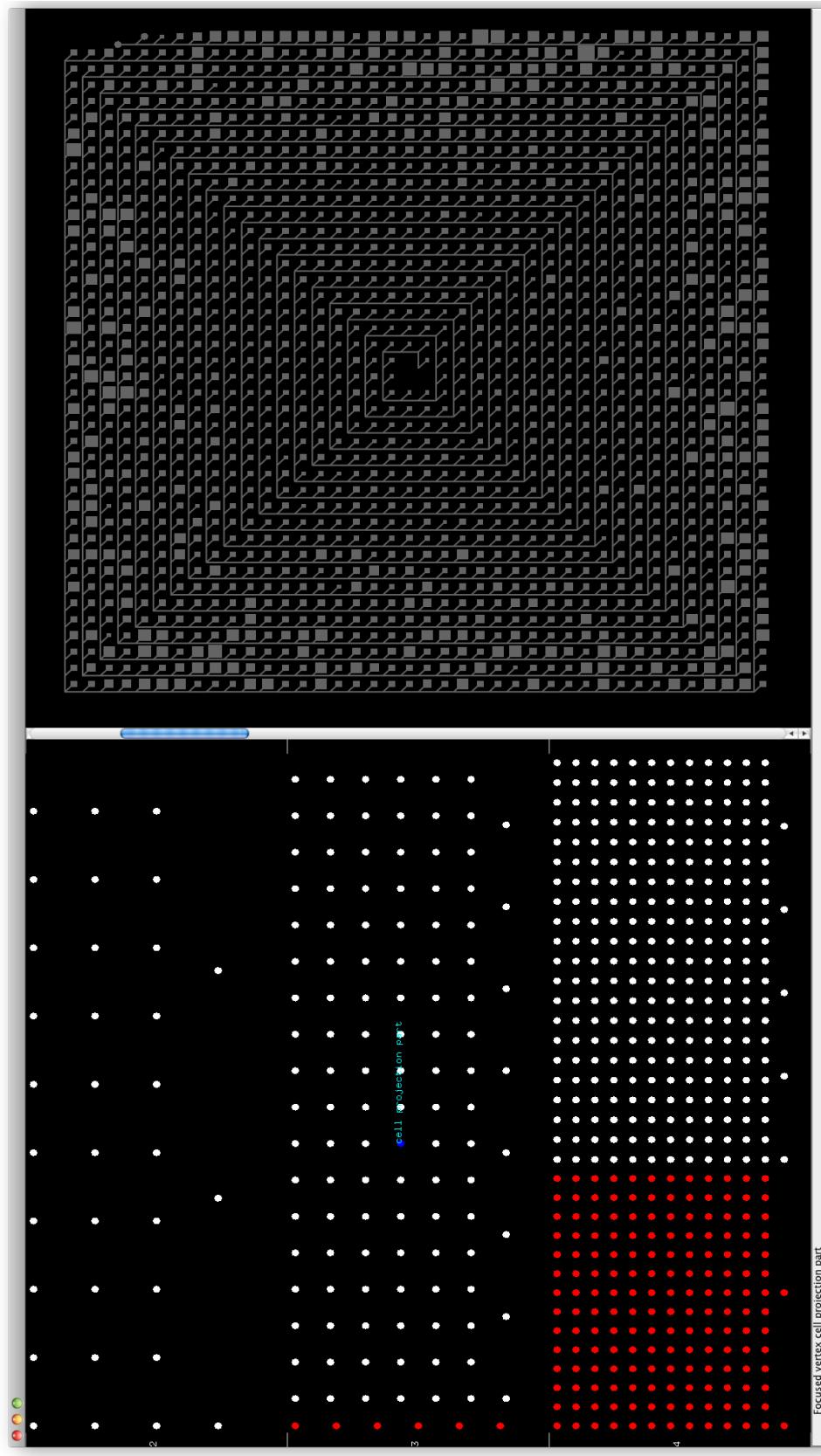


Figure D.3: Zoomed visualizatoin of the Gene Ontology levels (2, 3 and 4) and focused vertex “cell projection part”

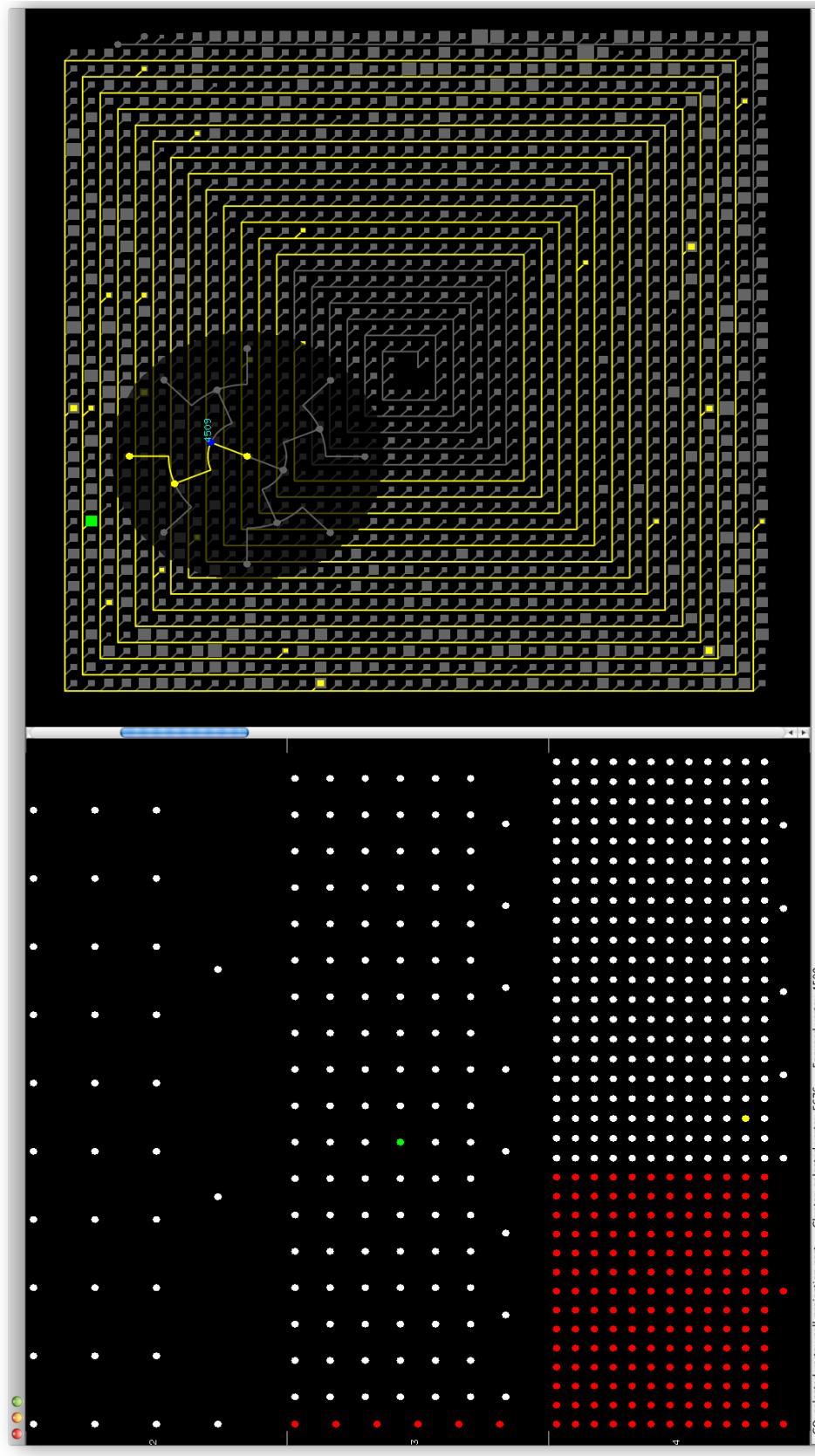


Figure D.4: Highlighted sub-graph visualization for GO gene “cell projection part” and Radial lens view of the Cluster grouped vertex “5676” (green rectangle)

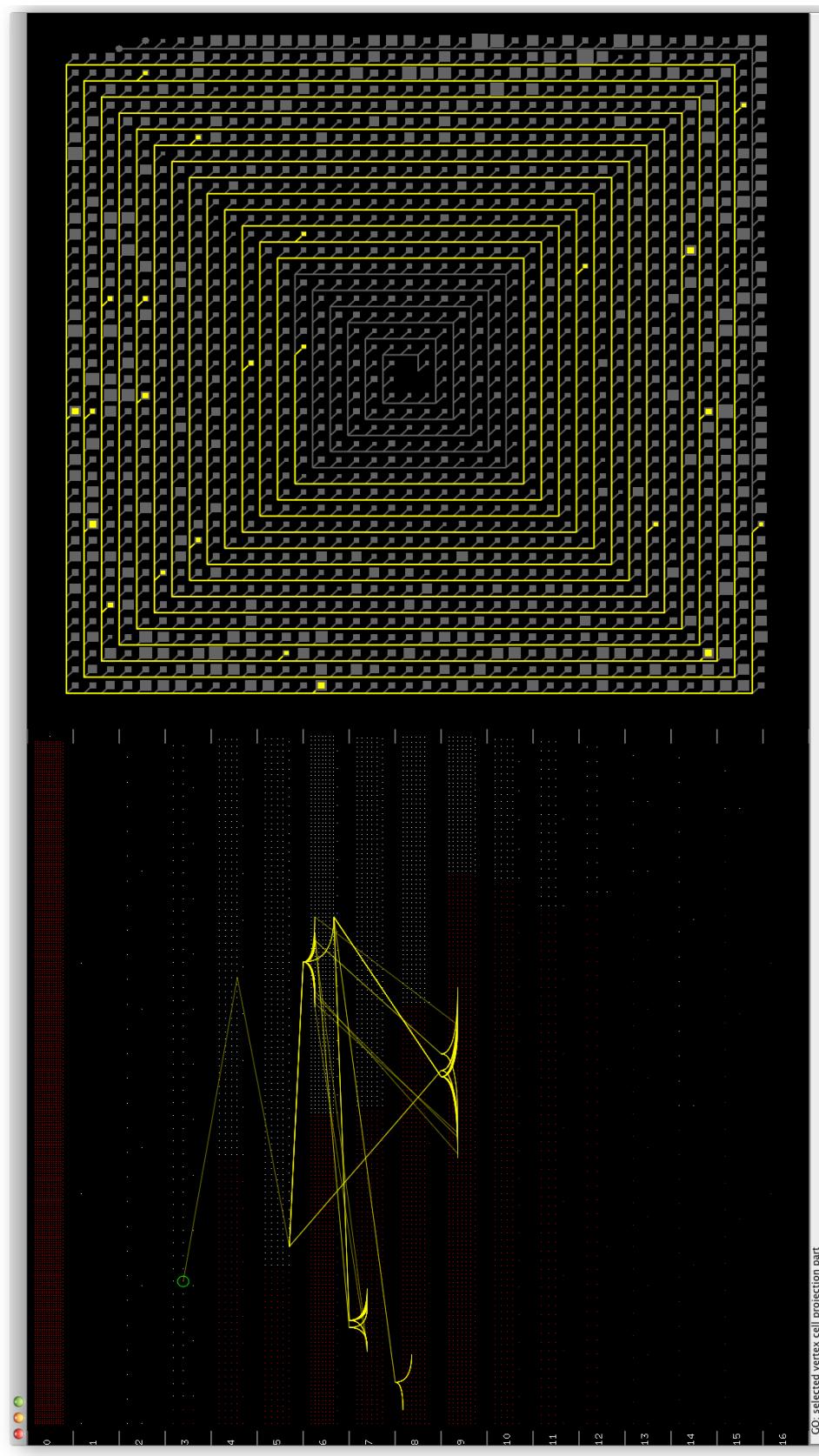


Figure D.5: Highlighted sub-graph visualization for selected vertex “cell projection part” (green circle)

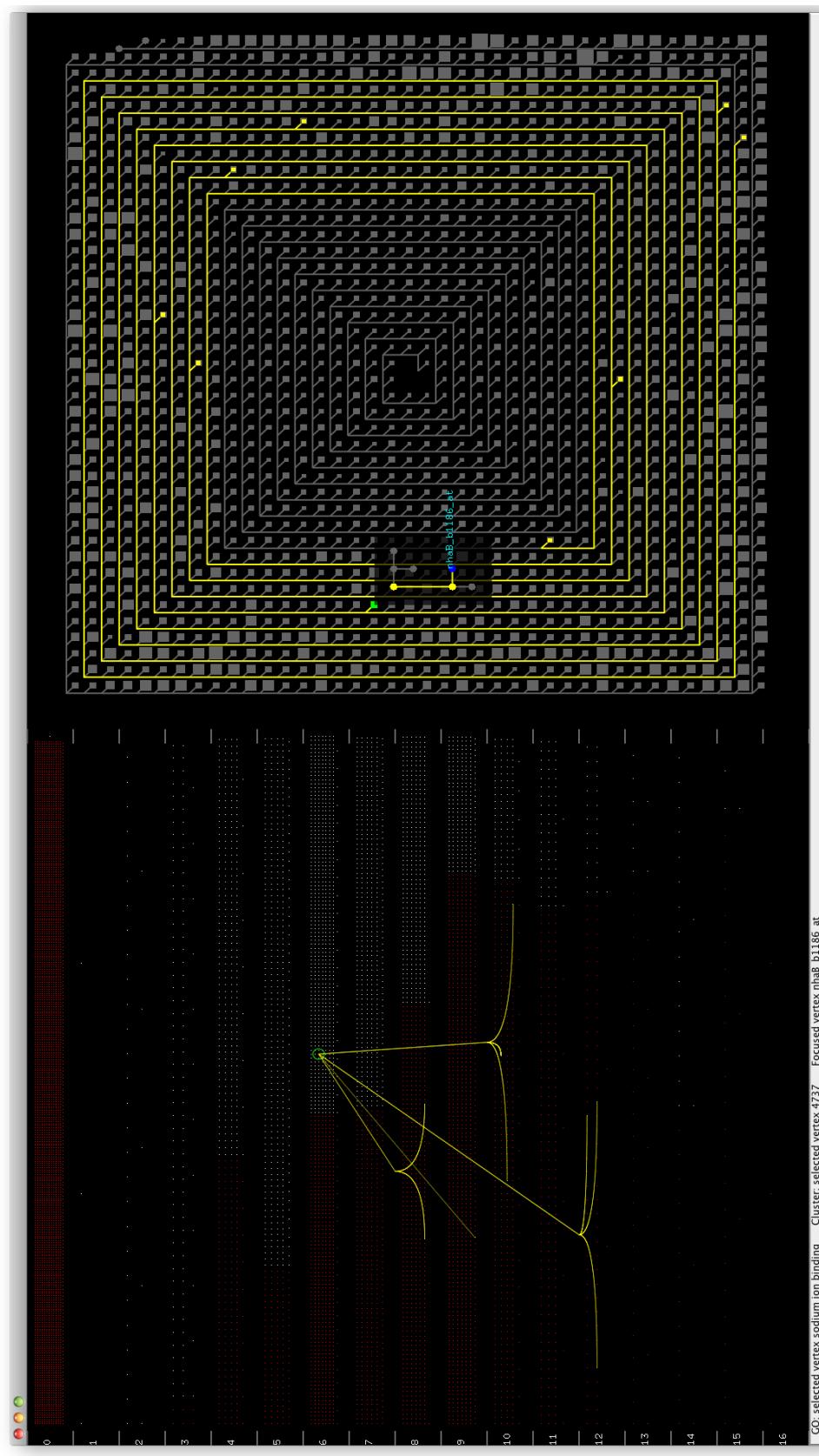


Figure D.6: Highlighted sub-graph visualization for GO vertex “sodium ion binding” and Rect lens view of the Cluster grouped vertex “4737”.

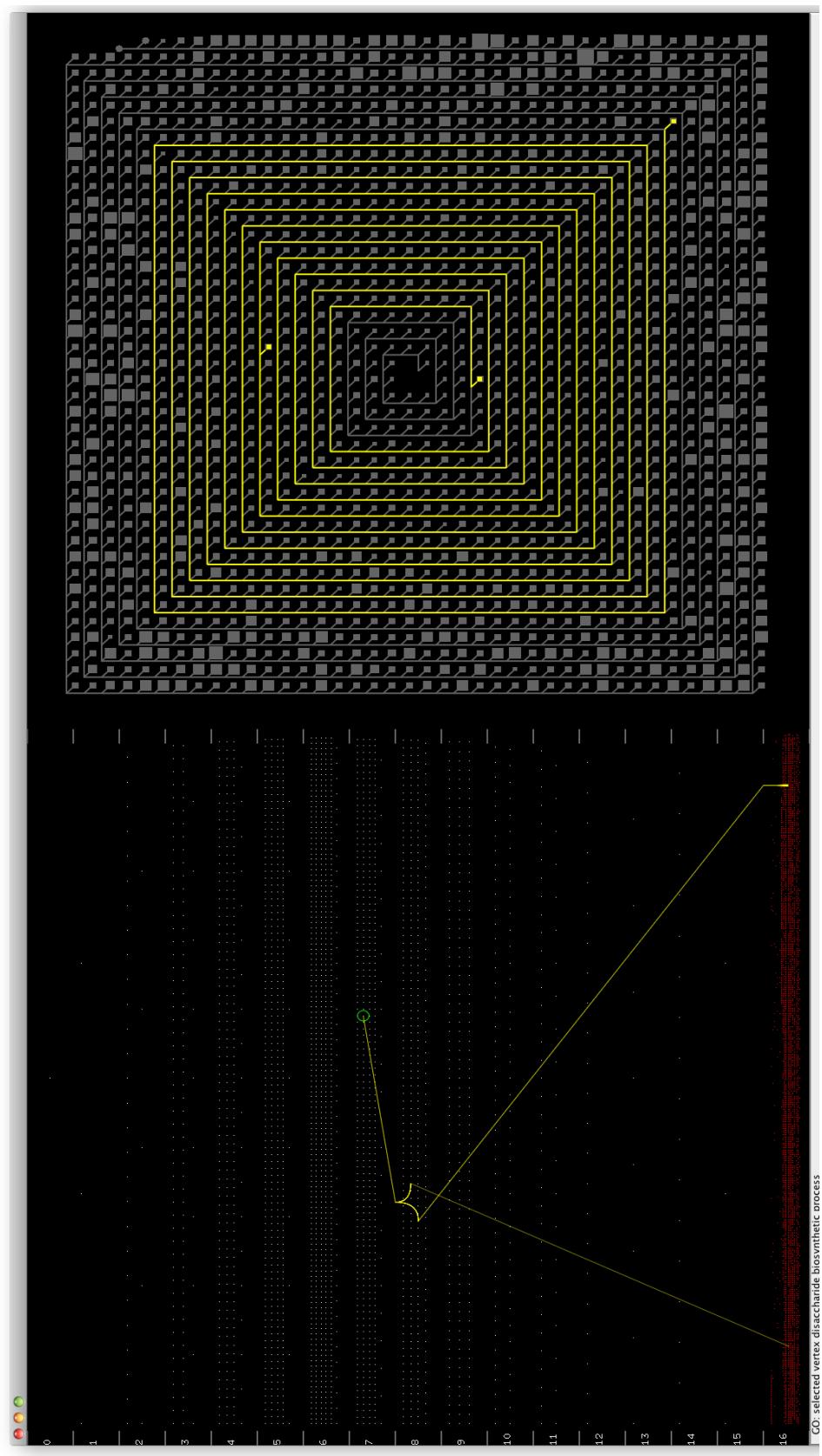


Figure D.7: Visualization of the Gene Ontology graph using “Leaf bottom layout” without option “Show unconnected components” and highlighted sub-graph for the GO vertex “disaccharide biosynthetic process”