MSc thesis

microRNA Regulatory Networks and the Gene Ontology: Analysis and Visualization

Alkiviadis Symeonidis

Computer Science Department
University of Crete

Heraklion, March 2006

Abstract

In the last years biologists' attention has been drawn in a new area which explores the behavior of genes. Key elements in this area are microRNAs, short (22 nucleotide) RNA genes. The important property of microRNAs is their ability to regulate other genes. Apart from this, microRNAs are also included as parts of other genes. These two types of relations between microRNAS and other genes form a network which is called "microRNA regulatory network". Unfortunately, the experimental verification of such regulations is a hard, time-consuming and expensive task. Therefore, attention has been shifted to algorithms for the prediction of such regulations. Since efforts are focused on the development of predictions, not enough attention has been paid to the discovery of the structure and other properties of these networks. We attempt to fill this gap with an application for the interactive visualization of microRNA regulatory networks.

Furthermore, knowledge about genes' functional annotation is stored and organized in the Gene Ontology (G.O.). We propose an alternative way of looking at microRNA regulatory networks, through the G.O. To support this, we also propose an algorithm for the visualization of the G.O. In our application a browser for the G.O. is also included and we offer the ability to analyze these two objects concurrently.

This work provides interactive algorithms for the visualization of microRNA regulatory networks and proposes a different point of view for them, through the Gene Ontology.

Περίληψη

Τα τελευταία χρόνια το ενδιαφέρον των βιολόγων έχει εστιαστεί σε μία νέα περιοχή που ερευνά τη συμπεριφορά των γονιδίων. Στοιχεία κλειδιά σε αυτήν την περιοχή είναι τα microRNAs, μικρά (22nt) γονίδια RNA .Η βασική ιδιότητα των microRNA είναι η ικανότητα τους να ρυθμίζουν άλλα γονιδίων. Εκτός αυτού, microRNAs περιέχονται σε τμήματα άλλων γονιδίων. Αυτοί οι δύο τύποι σχέσεων μεταξύ microRNAs και άλλων γονιδίων δημιουργούν ένα δίκτυο, το «microRNA δίκτυο ρυθμίσεων». Δυστυχώς, η πειραματική πιστοποίηση τέτοιων ρυθμίσεων είναι μία δύσκολη, χρονοβόρα και ακριβή εργασία. Για αυτό, η προσοχή έχει μεταφερθεί σε αλγορίθμους για την πρόβλεψη τέτοιων ρυθμίσεων. Επειδή οι προσπάθειες έχουν εστιαστεί στην παραγωγή τέτοιων προβλέψεων, ανεπαρκές ενδιαφέρον έχει δοθεί στην εξερεύνηση της δομής και άλλων ιδιότητων αυτών των δικτύων. Επιχειρούμε να καλύψουμε αυτό το κενό με μία εφαρμογή για τη διαδραστική (interactive) οπτικοποίηση των microRNA δικτύων ρυθμίσεων.

Επιπλέον, γνώση για τη λειτουργία-δράση διαφόρων γονιδίων βρίσκεται αποθηκευμένη και οργανωμένη στη Γονιδιακή Οντολογία (Γ.Ο.). Προτείνουμε μια εναλλακτική οπτική γωνία για τα δίκτυα ρυθμίσεων microRNA , μέσω της Γ.Ο. Για να το υποστηρίξουμε, προτείνουμε επίσης έναν αλγόριθμο για την οπτικοποίηση της Γ.Ο. Στην εφαρμογή έχουμε συμπεριλάβει ένα εργαλείο για την διάσχιση- εξερεύνηση της Γ.Ο. και προσφέρουμε τη δυνατότητα να αναλυθούν τα δύο αντικείμενα παράλληλα.

Αυτή η εργασία παρέχει διαδραστικούς αλγορίθμους για την οπτικοποίηση microRNA δικτύων ρυθμίσεων και προτείνει μία διαφορετική οπτική γωνία για αυτά, μέσω της Γονιδιακής Οντολογίας.

Acknowledgments

First of all I would like to thank my supervisor Professor Ioannis G. Tollis for his interest, support and guidance. Our useful discussions and his suggestions made me think and helped me overcome difficulties that appeared during this work. Next, principal researcher Martin Reczko is another person I need to thank. He participated in our meetings with useful suggestions, showed great interest for the work and provided us with the biological background that was necessary.

I would like to thank Professor Constantinos Stefanidis and principal researcher Giorgos Potamias for their partitipation in the evaluation committee and for their comments on the work.

Many members of the algorithmic analysis group in the Computer Science Department of the UoC and the IMBB lab of FORTH have also helped me with suggestions that improved the quality of this work.

I also thank the FORTH and the UoC for the financial support and for the infrastructures I used.

Of course during the years I spent in Heraklion a number of friends played the most important role. They always encouraged me in my lows and were glad with my ups. I would not like to risk forgetting a name, so I just want to say a great thanks. After all, they know who they are. Thank you.

Finally, four people back in my birthplace, had me always in their minds and probably where more anxious than me about my progress. So above all, I would like to thank my father Giorgos, my mother Maria, my sister Christina and my brother Aggelos. This work is dedicated to them.

Contents

List of Figures

Chapter 1

Introduction

In the last years a new area of biology has been discovered, exploring the non-coding transcriptome. Key elements are the microRNAs, short (∼22 nucleotide) RNA genes that regulate the fate of other transcribed RNAs. As knowledge about microRNAs rises and more data become available, computer aid for understanding this information has become necessary. While databases can store and manipulate this information, a visual representation is also required to have a global view of the data and the structural information. MicroRNAs are related to other RNA genes in two ways: a) they regulate genes and/or b) they are included in genes as part of their sequences . In addition, knowledge about genes' functional annotation is stored and organized in the Gene Ontology (G.O.). In this thesis we present algorithms for the visualization of microRNA regulatory networks and the G.O. hierarchy. In addition, we propose an alternative way of looking at microRNA regulatory networks, through the G.O. and offer the ability to analyze these two objects concurrently.

1.1 microRNAs

When microRNAs were first discovered in *caenorhabditis elegans* (1993, [1]) their role was unknown. Today, we know that these short RNA sequences play a very important role. They can silence well defined genes, in other words inhibit protein expression. Furthermore, they seem to be able to regulate different genes in various ways , while other short interfering RNA (siRNA) sequences can only silence the gene from which they originate. The importance of microRNAs and the attention paid to them increased vastly when researchers realized that such sequences exist in most (almost all eukaryotic) organisms, human included [2]. The challenge is to identify the regulating relations between microRNAs and genes. Zamore's words reveal what an enormous impact this could have:
"Changing a single base in a microRNA gene could potentially change its target specificity, so regulating many of an organism's developmentally important genes with microRNAs would provide a large and rapidly accessible pool of options for evolutionary change"[3].

1.2 microRNA Target Predictors

Unfortunately, up to now not many such regulating pairs have been (experimentally) verified. But as we keep learning how microRNAs behave, new methods for the prediction

of microRNA targets are being developed. Based on different algorithms many programs have been created:

- **TargetScanS** Lewis et al. MIT,
  { http://genes.mit.edu/targetscan,[4],[5]}
- **miRanda** Enright et al. Memorial Sloan-Kettering Cancer Center,
  { http://www.microrna.org/miranda.html, [6], [2], [7]}
- **DIANA-MicroT** Hatzigeorgiou et al. Univ. of Pennsylvania,
  { http://diana.pcbi.upenn.edu/DIANA-microT , [8]}
- **rna22** Rigoutsos et al. IBM
- **PicTar** Rajevsky et al. NYU,
  { http://pictar.bio.nyu.edu/, [9], [10] }
- **RNAhybrid** Rehmsmeier et al. Univ. of Bielefeld,
  { http://bibiserv.techfak.uni-bielefeld.de/rnahybrid/welcome.html, [11]}

Since all these programs predict microRNA targets, a matter of comparison rises. Which is the most effective, and more important: do they predict the same target genes? On the 17th of February 2005 in New York, the NYAS competition aimed to answer these questions. The task was the same for all programs: Given two microRNAs, their targets had to be predicted. The results varied widely and the overlap was very small. In this work, we have used results obtained with the miRanda program. In our data (but this holds in general as well) there are two ways one microRNA can be related to a gene. The first is that the microRNA sequence is included in the intron of a gene. The second and most important is a gene being regulated by a microRNA. The aforementioned programs try to predict any such regulation. In miRanda, for every prediction made, binding energy is also reported. The higher the value is, the "stronger" the regulation is. Based on this, a score is assigned to each regulation. A threshold value of 110 is proposed to be the value under which regulation should not be considered.

1.3 Gene Ontology

The multitude of standards in gene naming has created problems in the communication between databases. Different research groups use different gene identifiers, so for interfacing, a list of synonyms has to be stored for every gene. In addition, genes are involved in biological functions, whose detailed description increases the difficulty in interconnecting databases.

The best organized effort to create and use a common set of biological annotation terms is the Gene Ontology Consortium (www.geneontology.org). Many of the synonyms for the names of the genes are supported, while a unique G.O. identifier is assigned to every term. In G.O. three main ontologies for describing a gene's functionality exist. The first and larger in size is the "*biological process*" branch. Here, information about the broad biological goal of every gene, such as *development* or *reproduction* is stored. In the second category, "*molecular function*" , the tasks performed by individual gene products are described. Two important categories in this branch are *binding* and *catalytic activity*. The third and smaller branch is the "*cellular component*", where the subcellular location

each gene product operates in is described. *Organelle* and *extracellular region* are two important terms in this branch. All the terms in the gene ontology are organized in the three branches in hierarchical order. The hierarchy has two meanings. Either going from general to specific, with an *isA* relationship or going from the whole to a part with a *partOf* relationship. So every term that appears in the "*biological process*" branch *is a* biological process. Similarly, *cell surface* is *part of* a *cell*. The terms in the gene ontology are regularly updated and their number is about 20.000 with more than half of them belonging to the "biological process" branch. This update does not consist only of insertions of new terms. It also contains updating or removing some terms. For terms that are no longer in use, three additional branches exist: Obsolete Biological Process, Obsolete Molecular Function and Obsolete Cellular Component, where terms that have been removed from the respective branch are placed.

Another phenomenon in the G.O. is the existence of synonyms. In some occasions multiple names-definitions are assigned to the same entity. This mainly serves compatibility with previous versions after the insertion of new information.

In addition to the text description, a unique G.O. identifier is assigned to each term, allowing for easier manipulation of the hierarchy. In a separate database, every gene is associated to terms in all three categories based on knowledge about its functionality.

Part of our work aims to identify microRNAs that regulate genes with the same functionality, in other words, genes related to the same G.O. terms. The opposite, genes with similar functionality that are regulated by one or more common microRNA(s) was also among our goals.

1.4 From Biology to Graph Theory

The described biological topics, microRNA networks and the G.O. can be mathematically modelled as graphs. A graph is a structure which is used to represent objects and relations between them. A graph $G(V, E)$ consists of a set V of vertices, which represent the objects and a set of edges $E \subseteq \{V * V\}$, which represents relations. Both the microRNA networks and the G.O. have many characteristics which place them in special families of graphs.

1.4.1 microRNA regulatory network: A directed bipartite graph

The microRNA network consists of a set of microRNAs and a set of related genes. There exist two types of relationships between genes and microRNAs:

1. a microRNA regulates a gene
2. a microRNA is included in the intron of a gene

We construct a graph that represents this information in the following way: First, the set of vertices V is determined to be the set of microRNAs and the set of genes. In addition, for every relationship between a microRNA and a gene, we add an edge that joins the two respective vertices. In order to discriminate between the two different types of relationship, we direct the edges. So we define that an edge directed from a microRNA-vertex to a gene-vertex, implies that this microRNA regulates the gene. On the contrary, an edge from a gene to a microRNA means that the gene includes the microRNA sequence. With this convention, an edge does not simply show a pair of vertices that are related. It also stores

information about the type of relationship between them according to its direction. Graphs whose edges have a direction are called *directed*. Furthermore, using the definition above no relationship between two microRNAs or two genes exists. Thus, we can divide the set of vertices V into two subsets $V1, V2$ such that no edge has both endpoints in V1 or in V2. The two subsets are:

1. $V1$: the set of microRNAs
2. $V2$: the set of genes

Graphs with this property are called *bipartite*. In Figure 1.1 we show how the network is represented.

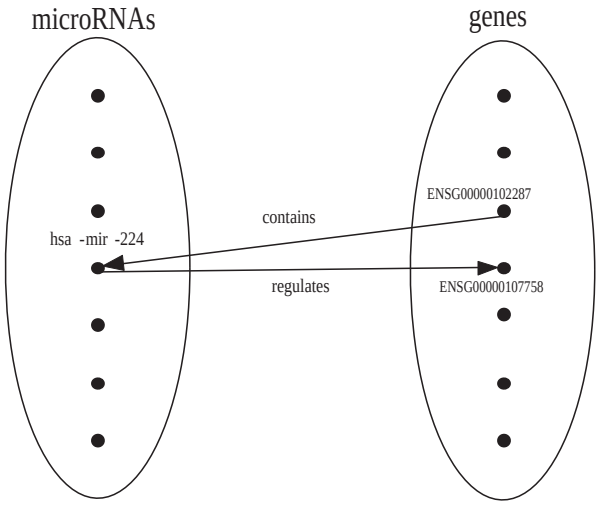


Figure 1.1: An example of a microRNA network.

1.4.2 Gene Ontology: A directed acyclic graph (DAG)

The G.O. stores thousands of terms that describe information related to the biological role of genes; from the activity itself to the exact subcellular location each gene product operates in. The G.O. is organized in a hierarchial manner, which means that the terms are placed in levels that go from general to specific. This information can also be represented by a graph. The set of vertices V is the set of terms in the G.O. This mapping though is not one to one, due to the existence of synonyms. For the set of synonym terms that describe the same entity only one vertex is created. An edge is used to declare the *isA* or *partOf* relationship that exists between two terms. Again, the assignment of direction to edges is required in order to declare the general and the specific term. In this case, our convention is that the edges are directed from general to specific terms. The G.O. hierarchical construction guarantees the absence of cycles (sequence of adjacent edges which starts and ends in the same vertex) in the graph. A cycle would mean that some term is more specific (and general) than itself. Directed graphs with this property are called *Directed Acyclic Graphs (DAGs)* and are a very interesting category in Graph Theory with many practical applications. Figure 1.2 shows a small portion of the first layers in the G.O. hierarchy.

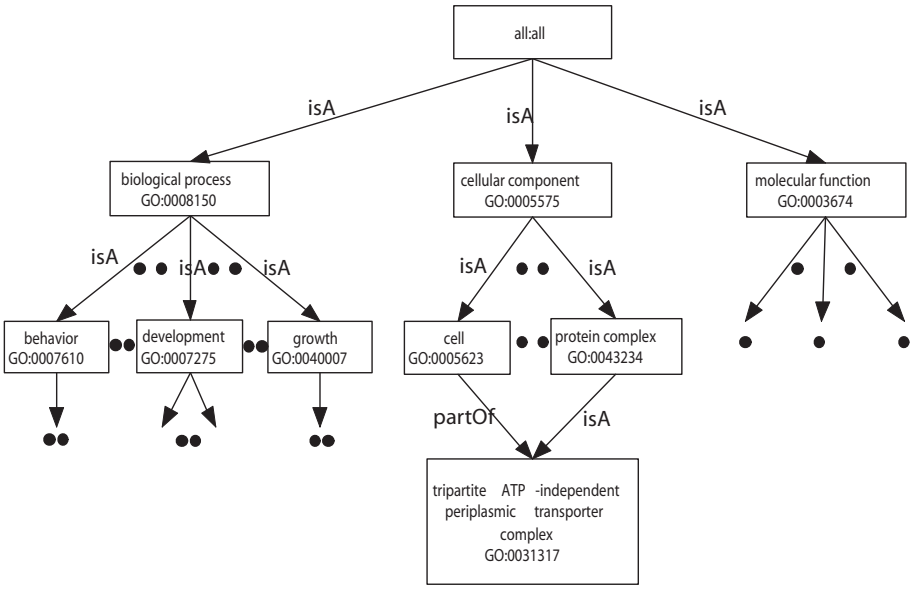


Figure 1.2: Part of the first layers of the Gene Ontology.

It is also important to notice that every term appears in only one of the three main branches (biological process, cellular component, molecular function). Thus, a more detailed description would be that the G.O. is a set of three *DAGs* merged under a common, artificial root. The three obsolete categories (Section 1.4.1) must also be taken into consideration. So the overall structure of the G.O. DAG is as illustrated in Figure 1.3

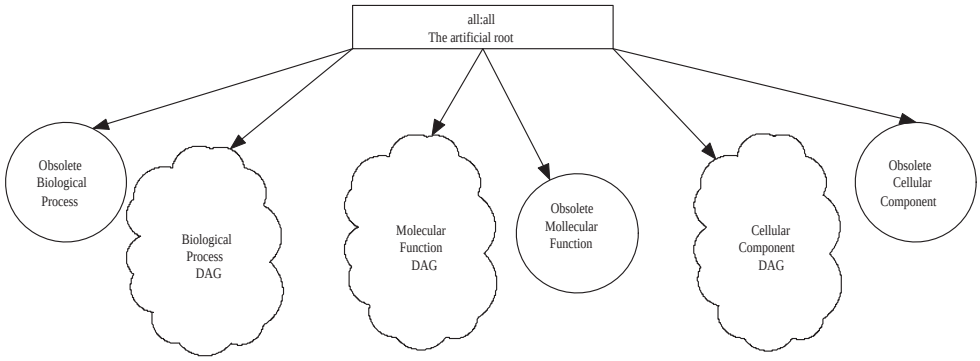


Figure 1.3: Structure of the Gene Ontology.

1.4.3 Organization of this Thesis

In Chapter 2 we review graph drawing methods and focus on their application for the visualization of biological information. We also review how the G.O. and microRNA regulatory networks are currently displayed. In Chapter 3 we describe our visualization approach, implementation, and user interface which is based on common interface tools. Chapter 4 describes how we visualize these networks combining different graph drawing techniques. Finally, in Chapter 5 we discuss possible extensions for our work.

Chapter 2

Previous work

2.1 Graph Drawing Techniques

Graph drawing has applications in many different fields such as computer and social networks, Entity-Relationship models or any other type of network. Based on the application field, the attributes to be emphasized and the structure of a network, many different drawing techniques have been proposed. In [12] a large number of such algorithms is described.

Directed graphs like the G.O. hierarchy are typically drawn using algorithms that try to display their hierarchical structure. The polyline drawing convention is the basis for such algorithms, [13, 14] and the main idea is to create hierarchical layers, [15], [16], [17]. Another approach to the visualization of hierarchical information is treemaps [18],[19],[20], where vertices that are low in the hierarchy are placed over those that are in the first layers in the hierarchy. This method was initially proposed for trees, but can be expended for directed acyclic graphs as well.

Bipartite graphs are commonly drawn based on the idea of the placement of each partition in a column. But general drawing techniques have also been used for their visualization. The best known technique for drawing graphs is probably the most general as well. Back in 1963 Tutte showed "How to draw a graph" in his so entitled work [21]. The main idea was that vertices connected with edges should be placed close enough to make the edge clear, but not to close. His algorithm was the first in a family of graph drawing techniques called force-directed [22, 23, 24, 25]. The name "force directed" is due to the fact that many techniques of this family simulate a natural system, usually using spring and magnetic forces, and allow the system to reach equilibrium.

Force directed algorithms are also used for the visualization of graphs with clusters. Cluster oriented methods [26], [27], [28], [29], [30], [31], [32], usually combine two techniques. First, a supergraph is created, where every cluster is represented by a vertex. Then this supergraph is drawn usually with a force-directed approach. Finally, every cluster is drawn individually using some appropriate graph drawing technique and replaces its supervertex. Appropriate algorithms include orthogonal [33],[34],[35] where every edge is drawn using successive horizontal and vertical line segments; and circular [36], [37], [38], [39], where all vertices are placed on the periphery of a cycle in such a way that the number of crossings remains low.

2.2 Visualization of Biological Information

Graph drawing in biology

During the past few years, graph drawing techniques have started being used for biological applications, and the convenience an efficient visualization provides has been revealed. Many biologically important networks exist: metabolic pathways, microarray (gene-expression-based) networks, microRNA regulatory networks, protein interaction networks, phylogenetic trees etc. An efficient visualization of such networks can be used to display the information and eventually reveal hidden or unknown properties. Depending on the nature and the structure of these networks, many graph drawing algorithms have been applied to biological data [40],[41],[42],[43],[44],[45]. These algorithms have been included in tools such as Cytoscape [46], Spotfire [47], Avadis[48], NetAffx from Affymetrics [49] and GenMapp [50].

Visualization of microRNA networks

The main question about microRNAs is which genes they regulate. Since there are many different programs that try to answer this question giving different results (as seen earlier), the web-sites for most of the programs provide a user interface where text-based search is feasible. However, little attention has been paid in the whole microRNA network. In [2] the authors report their findings on the distribution of the degrees of vertices, concluding to a biologically surprising result that the variance is great. This observation shows that more complex interesting questions can be easily answered if the information is organized in a graph. Another limitation of text-based searching is that it cannot provide a global image of the network and it is difficult to include information from other sources, such as the Gene Ontology. Since the answers provided by the predictors are not guaranteed to be accurate, research is mainly focused on predicting rather than visualizing regulations. So information is mainly available in databases as text. In our work, we make an attempt to fill this gap. We use a prediction as the answer to the basic question, and focus on the other problems.

Visualization of the Gene Ontology

The G.O. includes important information about genes' functionality. Microarray experiments, [51],[52], also provide information about genes. There have been many attempts to combine information from both sources [53],[54]. In most of the available tools, the G.O. is represented using expanding lists or trees(Fig. 2.1), which are very efficient to manipulate small hierarchies, but for larger hierarchies only manage to display a small portion of the graph. Such visualization techniques have been used in tools like GOfish [55], Amigo[56], CGAP[57] and dagEdit [58].

The structure of the G.O. indicates that some hierarchical graph drawing technique can be used in order to visualize it. The most important issue is the area required for drawing G.O. since the size of the graph is very big. Treemaps provide a space-efficient solution which additionally displays the hierarchy and allows for easy and fast navigation through the graph. So more recent approaches to the problem compute a portion of the G.O. hierarchy that is related to a given set of interesting genes and use treemaps to visualize it (Figure 2.2)[59].

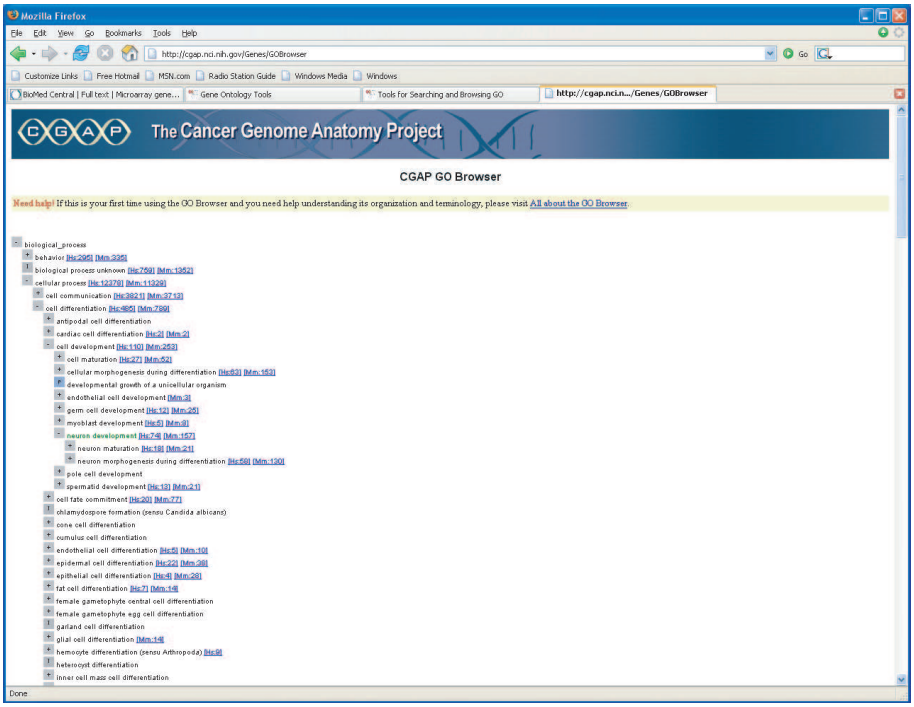


Figure 2.1: tree representation of the G.O.

In our work, we use both approaches in order to visualize the G.O. The first, list-based interface is widespread and commonly used and allows for local searching. On the other hand we have used treemaps to visualize the whole G.O. , and not only a part of it, providing an efficient way to navigate through it.

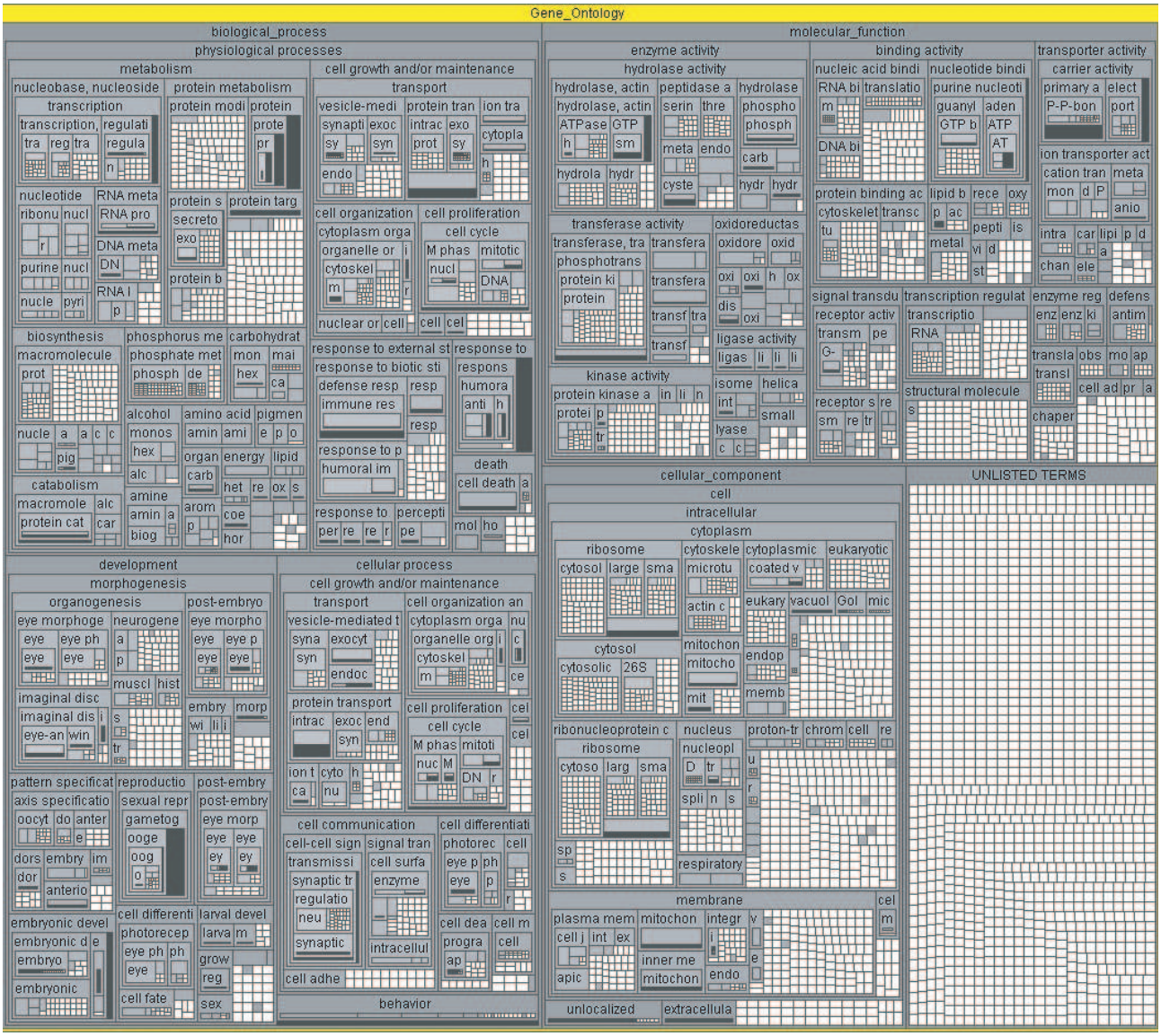


Figure 2.2: The Gene Ontology as a Treemap.

Chapter 3

A Graphics User Interface for the microRNA Regulatory Network and the Gene Ontology

In this Chapter we discuss what information we want to see from the predicted microRNA regulatory network and the Gene Ontology. We also present Graphics User Interfaces for them and show how these two interfaces can be used in parallel to analyze the microRNA network through the G.O.

3.1 microRNA regulatory network

As described in section 1.4.1 the microRNA network is a directed bipartite graph. The most important piece of information one wants to have access to, is the relations between genes and microRNAs. To be more precise for every microRNA one wants to see the genes it regulates and the genes that contain it their interior. Similarly, for every gene it is important to show the microRNAs that regulate it and also any microRNAs that are included in its interior.

The name of a gene does not tell much about its biological role by itself. There exist thousands of genes and it is impossible to remember the functionality of all of them. Thus, it is also important to show additional information about genes when we have it. And finally, since for many genes relations to various G.O. terms are available, we also want to have access to this information.

3.2 Graphics User Interface for predicted microRNA regulatory networks

Bipartite graphs are typically drawn with two columns of vertices (as in Figure 1.1), where each column contains one partition. In our context one partition is the set of genes and the other the set of microRNAs. The basis for this part of our work is the vertical placement of the two lists in a window in a clear manner (Figure 3.1). The first question that rises is how to display these two lists.

Two options are provided for ordering the genes in their list:

- alphabetic, for easy searching or

- in the same order they appear in our data, which is a set of spreadsheets provided as supporting material for [2]

We also support two options for ordering the microRNAs in the respective list.

- alphabetic, for easy searching or
- in decreasing number of regulating genes. This way, better regulators appear higher in the list

For the microRNAs, in both cases, the number of regulating genes is displayed in brackets next to the name.

The next problems to address were how to show all information we have about genes and microRNAs. As described, in the context of our work every gene has four attributes of interest:

- The set of microRNA sequences that are included in its intronic region, which is usually empty and rarely contains more than one microRNA
- The set of microRNA sequences that regulate it, whose cardinality varies from zero to 11. This number is also subject to the threshold used for the target predictor(see Section 1.2)
- Its annotation, which is a small description of the gene and is sometimes unknown in our data
- The set of G.O. terms that are related to it, extracted from the G.O.

In order to display these four attributes for some selected gene, four more components have to be added in the window of Figure 3.1. The three first attributes (included microRNAs, regulating microRNAs and annotation) appear in three boxes next to the genes' list. In Figure 3.1 they are labeled 1,2,3 respectively. The G.O. terms that are related to some selected gene are displayed in a box in the middle of the window (labeled 4 in the Figure). Whenever some gene is chosen by clicking on it, the four mentioned boxes are updated to contain information about the selected gene.

Similarly, for every microRNA we want to know the set of genes it regulates and if there exists some gene that contains it in its intron. This information appears for some selected microRNA next to the main list of microRNAs in two smaller list-boxes (5, 6 in the Figure). The four inner lists with genes and microRNAs are connected to the two main lists of the window. This allows the user to traverse a path in the microRNA network easily by repeatedly selecting terms from the four inner lists.

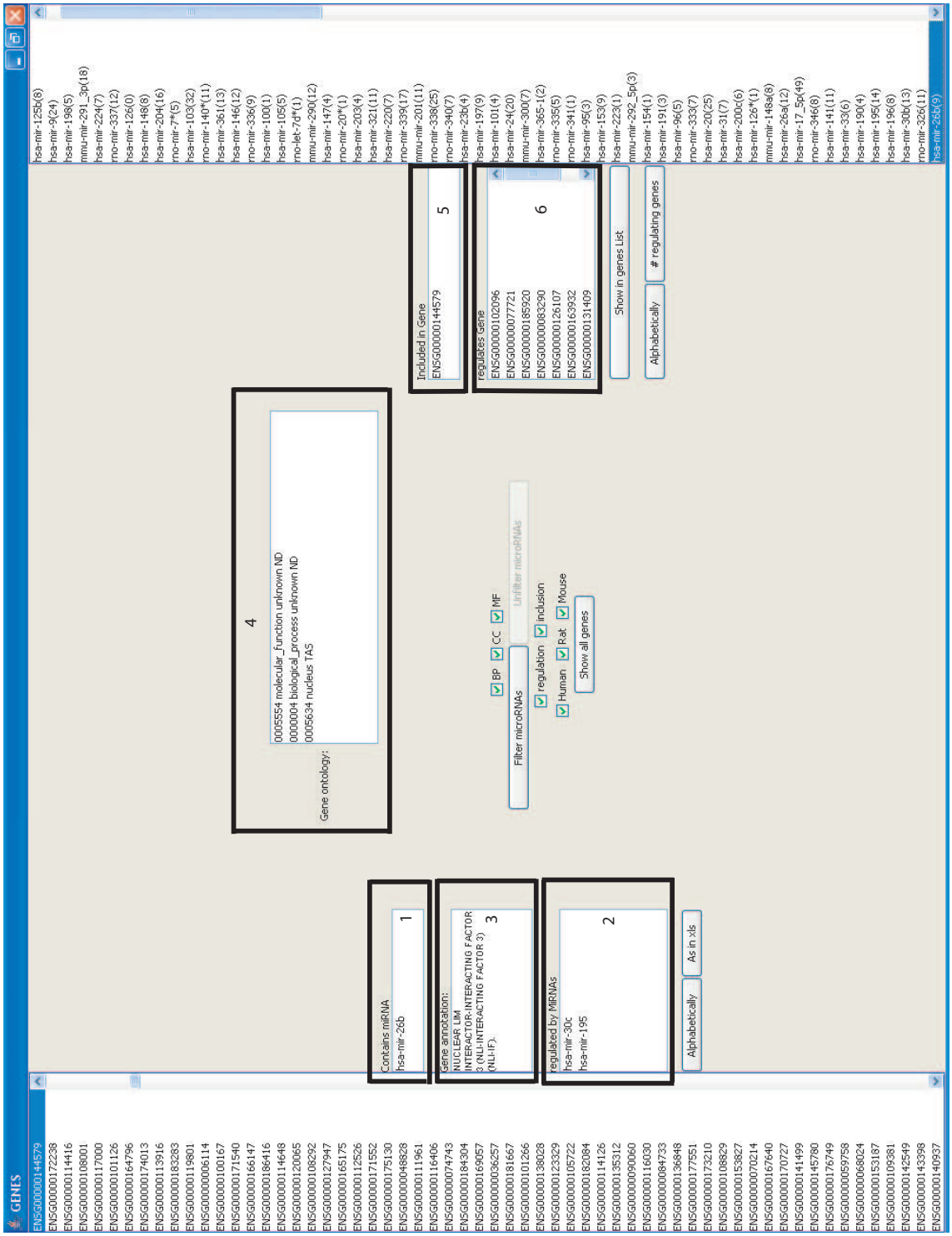


Figure 3.1: Box 1 shows all microRNA sequences contained in gene ENSG00000144579. Box 2 contains all microRNAs that regulate the gene. Box 3 contains the annotation of the gene. Box 4 contains the G.O. terms associated to the gene. Box 5 contains the genes that include microRNA hsa-mir-26b. Box 6 contains all genes that are regulated by hsa-mir-26b.

Usually, there is a specific subset of interest, either some GO terms or a set of microRNAs and genes, so a number of options are available to the user.

- **Filter microRNAs/unfilter microRNAs** The set of genes that appears in the main list, can be modified based on user actions as will be described in the following section. In such a case, many microRNAs do not have any relationship with the genes of interest. Selecting "filter" will remove these microRNAs, while by "unfiltering" they can be restored. Furthermore, some microRNAs are not related to any gene for high values of the predictor's threshold. Even for the proposed value of 110, a small number of microRNAs is isolated. Filtering removes these microRNAs.
- **Regulation/Inclusion** As already mentioned there are two types of relationship between genes and microRNAs: inclusion and regulation. While they are both of interest the user can select to explore only one of them. This is achieved by shrinking the list of microRNAs to only those that have at least one edge of the required type. All microRNAs that appear in the inner lists (next to the genes) but are unavailable, are printed using light gray color. The same happens every time some gene or microRNA appears in any of the inner lists, but has been for some reason filtered out
- **Human/rat/mouse** MicroRNAs in our data come from various experiments and include sequences found in humans, mice and rats. The option to exclude one or two types of them is available with three checkboxes.
- **BP/CC/MF** The G.O. consists of three totally different branches (Section 1.3). With these three checkboxes the user can filter out any of them.
- **Select microRNA's regulating genes** When a microRNA is selected, the main genes' list can be updated to include only the genes that are regulated by it.
- **Show all genes** With the above option only a small subset of the genes are shown in the main gene's list. In the next Section we show that the list can be also modified through the G.O. The user can use this option to place all genes back in the list.

These options make searching for different properties in the network easier. For example, if the "Inclusion" option is enabled and the "Regulation" option is discarded, the user is left with a small number of microRNAs which appear in the introns of some gene(s). Selecting these microRNAs one-by-one one can easily check if the genes that contain a microRNA are regulated by this microRNA as well. While intuitively one would expect this to happen for many pairs, only one such pair exists: microRNA *hsa-mir-181a* is included in gene *ENSG00000148200* and is also predicted to regulate it. This observation becomes more interesting with the fact that even if no threshold is used and all predictions are considered (even the "weakest") no additional pairs appear. The actual existence of this relation is an open question but two of the other predictors we have checked (TargetScan and PicTar) also predict the same regulation.

The only functional part not discussed yet is the list of G.O. terms that are associated to a gene. This is strongly related to the G.O. browser which is discussed in the next section.

3.3 Gene Ontology

The G.O. is a hierarchy of terms that are used to describe various concepts of genes' functionality. It is important to be able to browse this hierarchy since it is updated often

and new terms are inserted in branches that can be of interest. Furthermore the use of a small name and an id do not provide enough information. So for most terms a small description is also available, and it must also be displayed.

Since the G.O. provides a widely accepted naming scheme, many browsers for the G.O. have been developed. Most of them are based on the commonly used expanding tree, like the ones that are used for browsing folders in computer systems. The G.O. team itself in their web-site (www.geneontology.org) provides such a mechanism, developed by Amigo ([56]). The main advantages of this technique are (a) the hierarchy is clear and (b) the user can select to hide branches that are not of interest. Figure 3.2 shows an instance during the graph exploration. Since this user interface is very simple and most people interested in the G.O. are used to it, most of the tools that work on G.O. use this method as well. For the same reason, we have adopted a similar technique. The technique has been properly enriched so that communication with the previously described microRNA network user interface is feasible. One important goal is to provide the ability to limit the working sets of genes and microRNAs only to those related to some interesting G.O. terms. A stand alone version of this tool has been built to serve as an application for navigating the G.O.

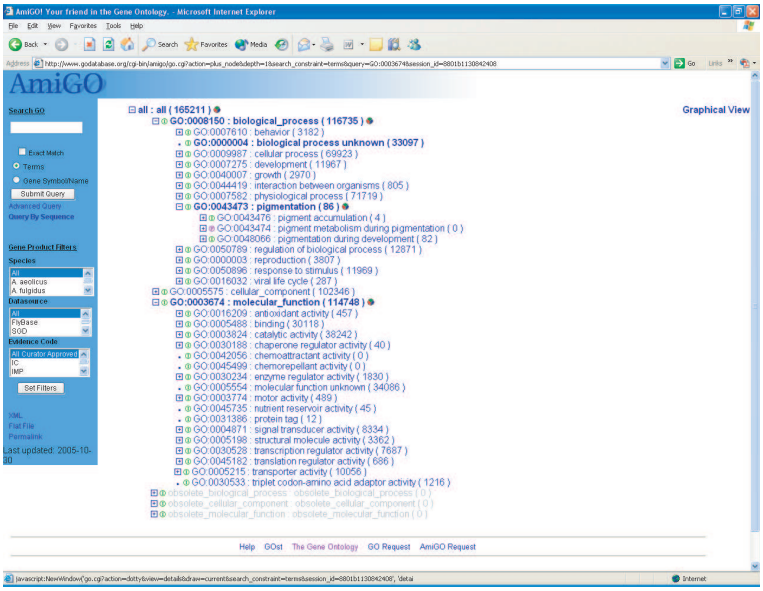
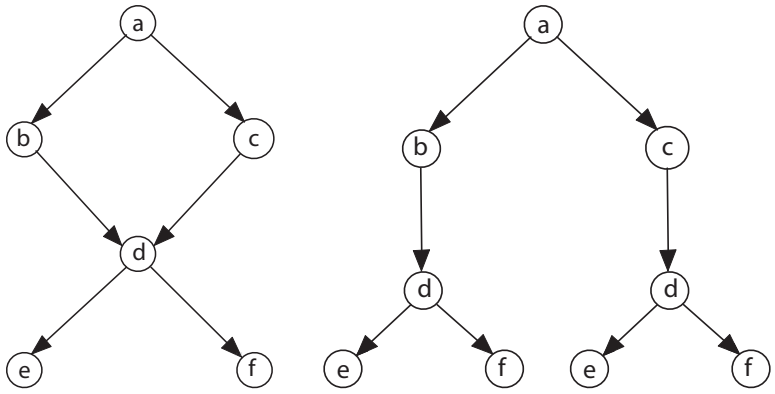


Figure 3.2: Screenshot from www.geneontology.org: How the GO graph is explored.

3.3.1 Converting a DAG into a tree

As indicated by their name, expanding trees are used for displaying hierarchies with tree structure. So the first step towards using such a method is to create a tree "equivalent" to the G.O. dag. The term "equivalent" is used to declare that all information in the dag is still present in the tree. The process to obtain such a tree is a two step procedure where all that is required is to copy every vertex with incoming-degree $deg \geq 2$, deg times. Each such copy is then assigned a unique parent, one of the deg incoming neighbors. Figure 3.3 shows an example.

The idea behind the algorithm is the following: Initially the root of the tree is placed. It is the same as the first vertex (with incoming degree 0) of the dag. Then for every vertex



(a) the initial DAG (b) the "equivalent" tree

Figure 3.3: DAG to tree conversion.

v in the dag, a recursive method *insertNode* is called. This method inserts the necessary copies of a vertex in the tree after ensuring that all the ancestors have been inserted. So if v has been inserted *insertNode* immediately returns, otherwise it adds all the incoming neighbors of v in the tree (recursive calls for the parents) one by one. Once a recursive call for an incoming neighbor u terminates all ancestors of v in the respective subgraph have been inserted into the tree and multiple copies have been created if necessary. Now we can insert copies of v to be children of u as many times as the copies of u are. Repeating this for every parent of v in the dag results in the necessary insertions of copies in the tree.

Since many copies of a vertex can exist and we have to perform the same action on all of them, we keep a hashtable where the key is a vertex and the value is a list with all of its copies. If the list is empty the vertex has not been inserted in the tree.

```
DAG2TREE( Directed acyclic graph G)
 1  Hashtable vertexCopies //the hashtable with a list of copies for every inserted vertex
 2  treeRoot = G.vertexWithInDegree0
 3  vertexCopies.put(treeRoot, treeRoot)
 4  for each vertex v in G
 5     do INSERTNODE(v)

INSERTNODE(v)
 1  if not vertexCopies.get(v).isEmpty() //if it has been inserted
 2     then
 3          return
 4
 5  for each incoming neighbor w of v  //Otherwise work for the parents
 6     do
 7        INSERTNODE(w)
 8        for each copy w_copy in vertexCopies.get(w)
 9           do
10              make a new copy v_copy of v
11              put v_copy in the tree as child of w_copy
12              append v_copy in vertexCopies.get(v)
```

This conversion of the G.O. dag produces a tree with a very large number of vertices, but it is the only way that allows us to visualize this hierarchy efficiently. The careful management of synonyms which was discussed in Section 1.4.2 is important since it prevents further, useless increase of the tree. If all synonyms are represented with different vertices, the resulting tree has $\sim$ 150.000 nodes. On the other hand, if all synonyms of a term are collapsed to one vertex, the number of nodes decreases to $\sim$ 100.000. The number is still very large (but is 33% lower than before). The size is not a severe problem for this browser since the user can hide all irrelevant branches and focus on a small area of interest. On the other hand, for the drawing technique which is analyzed later, this reduction is very important and leads to considerably improved results.

3.3.2 User Interface for navigating the Gene Ontology & communication with the microRNA network interface

The tree is computed when the application starts, but does not become visible until some G.O. term is clicked (box 4, Figure 3.1). When some term is selected the tree appears and all paths that lead to copies of the selected term are expanded. The copies themselves are highlighted for easier identification. The result of selecting the G.O. term "*cell growth and/or maintenance*" appears in Figure 3.4.

In the figure, apart from the tree itself one can see the set of options available. The tree expands and shrinks based on a user's request by clicking on the plus/minus boxes. When some term is right clicked, information about it appears in the bottom component of the window. Namely the information consists of:

1. **Accession**: the unique G.O. identifier that has been assigned to the term
2. **Instance**: The number of copies this term has in the G.O. tree. This is additional information, extracted during the dag-to-tree transformation
3. **name**: the name with which the term is stored in the G.O.
4. **definition**: some more detailed description of the selected term

On left click however, while most applications expand the term, in this tool some set of genes related to the clicked term replaces the set of genes in the main list of the microRNA network panel. The set is determined by the two groups of checkboxes in the middle of the window, which provide four combinations in total. One group determines the genes to associate with the term. They can be either the set of genes that are directly related to the selected term, or the set of genes that are related to at least one of the G.O. terms in its subtree. For example if the user chooses to display all terms directly related to the root of the tree, no gene will be selected since the root is artificial, with no biological meaning and thus no gene is directly related to it. On the other hand, almost all genes are related to at least one G.O. term so in case of asking for all terms related to some descendant of the root almost all genes will be selected. Information about the cardinality of these sets is available next to the name of each term with two numbers. The first is the number of genes directly related to the term and the second the number of genes related to some term in the subtree.

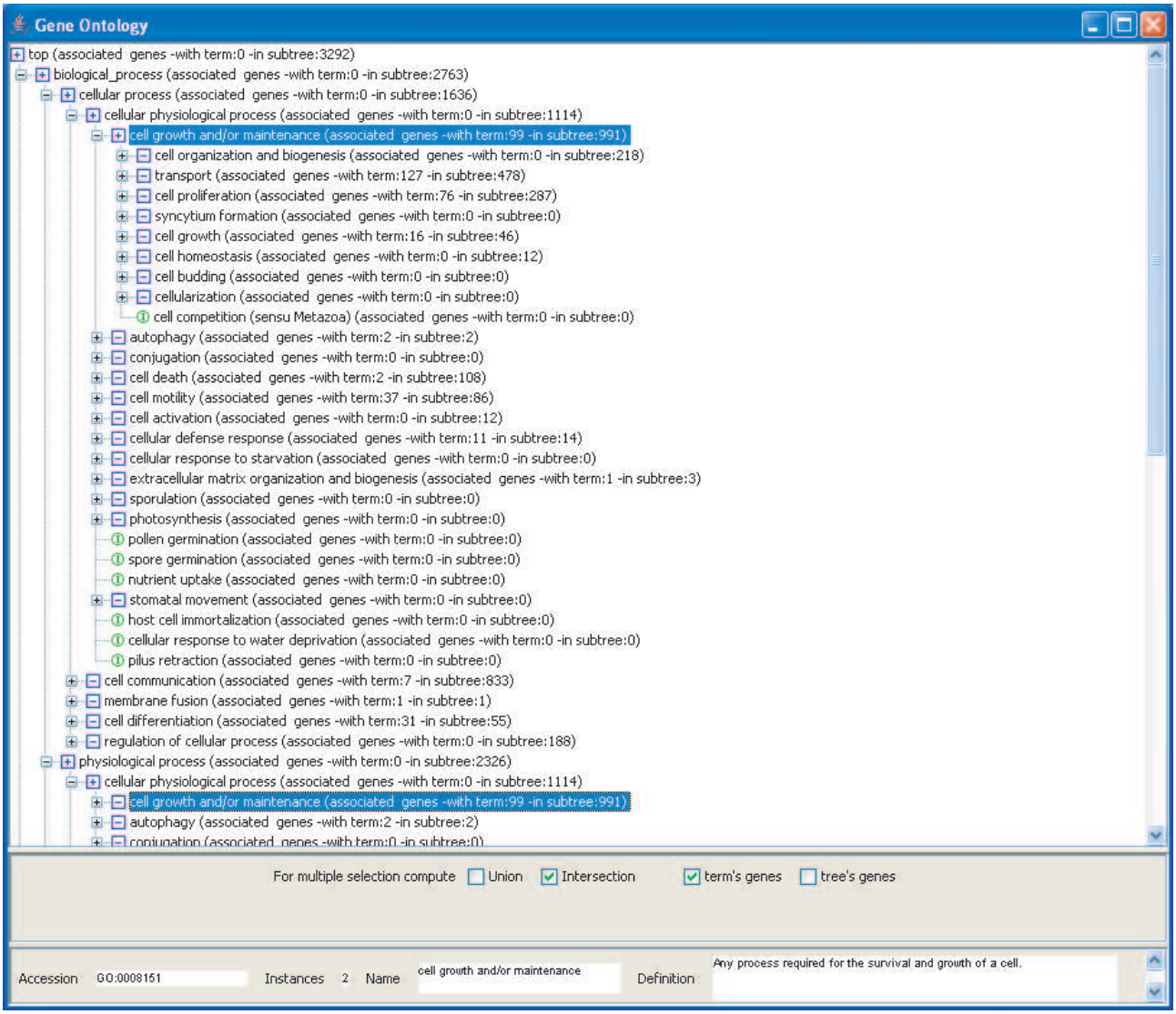


Figure 3.4: The Gene Ontology tree after enabling term "cell growth and/or maintenance".

Multiple selection of terms is also supported, through the second group of checkboxes. According to a user's wish again, the set of genes can be either the union of the terms' respective sets, if information about all terms is required or the intersection of them, if one wishes to find the genes that are related to all the selected sets of terms. The set of genes that is associated to every term is computed initially and stored for every term. Storing in efficient structures (HashSets, Java built-in data structures which are based on hashing) allows for the fast computation of the intersection and union of these sets.

The already mentioned "Filter microRNAs" option in the microRNA user interface is also useful in this context since it allows the user to limit only to the microRNAs that are related to the set of genes under examination at the time.

3.4 Summary

In this section a graphics user interface for navigating through the microRNA's network and the Gene Ontology was presented. Using this tool, a number of interesting queries can be answered for various sets of genes, microRNAs and/or G.O. terms. This application can also be used for simply traversing through the G.O. hierarchy or the microRNA network. The main limitation in this case is that the whole network cannot be visible at a time, but only small areas can be efficiently displayed. This is useful for local searching, but becomes inefficient if a large part has to be viewed. In the next Chapter, graph drawing techniques which deal with this problem are discussed.

Chapter 4

Graph Drawing Methods for the Visualization of the microRNA Regulatory Network and the Gene Ontology

In the previous Chapter a user interface that allows one to have access to all information is described. Only the links between one microRNA and the associated genes (and vice versa) are visible at a time. Similarly, only a small part of the G.O. DAG can be efficiently explored, since the expanding tree grows very fast. In this Chapter, graph drawing algorithms that allow for a more global view of the microRNA network and the G.O. DAG on user's demand are described.

4.1 Drawing the microRNA network

4.1.1 Properties and needs

MicroRNAs are expected to be a small portion of the number of genes(predicted to be somewhere between 1-10%). Following this, only some hundreds of microRNAs are predicted to be regulating some of the thousands of genes that are available. Thus, the two partitions have a very considerable size difference and this makes drawing the network difficult. In our datasets for example, there exist 4369 genes and 207 microRNAs. The first attempt towards solving this problem is to remove from our data all genes with no relation to any G.O. term and all genes that are not regulated by any microRNA (recall that this is based on the threshold value). Even after this removal, the problem remains since we have 1073 genes and the same 207 microRNAs. The second step towards simplifying the structure is to remove the edges that show the "gene contains in introns microRNA" relation. They are only a few dozens and can be easily and efficiently explored with the application that was described in the previous section. In addition, this removal leads to only one type of edges so we do not need a way to distinguish between them. After this pre-process we come to the question of what do we need to see for this network. The main requirement is that the bipartite structure must be clear. Also since a prediction- score is assigned to each regulation we also want to see this information.

Small bipartite graphs are typically drawn using two columns as in Figure 1.1, and this

was our first approach. The difference in the cardinalities of the two partitions was the reason we decided to divide the genes' set into two smaller sets. This lead to three smaller columns. The column with the microRNAs can be placed in the center and on both sides we can put the two sets with genes. The placement of roughly 500 genes and 200 microRNAs in columns is still inefficient as shown in Figure 4.1. since the objects can not be distinguished.

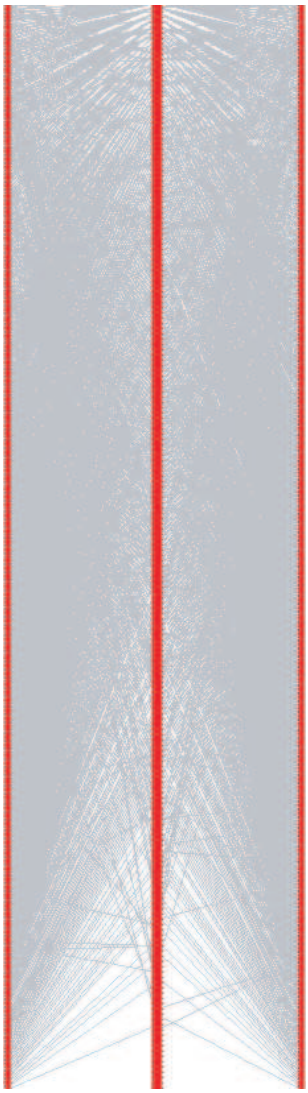


Figure 4.1: Draw the microRNA network with three columns.

The figure reveals an additional problem: due to the large degree of the microRNA-vertices, there are many edge crossings and the result is practically unreadable. But this is not the only difficulty. As discussed later, most of the available drawing area is occupied by the G.O. , thus the space that is available for the microRNA regulatory network is limited. This is a significant constraint. Furthermore, general graph drawing techniques (force directed, orthogonal, etc.) cannot reveal the bipartite structure of the network.

While ideally one would like to have a visualization of the whole regulatory network where the bipartite structure and other properties are clear, due to the aforementioned problems this can not be achieved. However, we can overcome these difficulties with an

interactive visualization technique, where the user specifies what he wants to see. Towards this direction we provide two visualization methods, one microRNA and one gene-oriented.

4.1.2 microRNA oriented drawing

In order to draw the microRNA regulatory network interactively two lists are used. One contains the genes and the other the microRNAs. If some microRNA is selected it is drawn using a small circle, and all genes that are regulated by it are placed on the periphery of a circle whose center is the microRNA. This way, we avoid drawing lines for many edges and obtain a much clearer result. An edge is drawn only if some gene has already been drawn. This happens when the user selects to display some microRNA which regulates a gene that is also regulated by some other previously selected microRNA (and is drawn on its periphery). In addition, once some microRNA m is selected a list of "similar" microRNAs appears and the user can select to display any of them. We define a microRNA m' to be similar to m if more than 50% percent of the genes regulated by m are also regulated by m'. The microRNAs are placed in two columns, in the order they are selected. The first microRNA in the left column, the second in the right, the third in the left under the first and so on. The names of the microRNAs are printed in the interior, while genes names are shown when the mouse is over them. A screenshot appears in Figure 4.2.

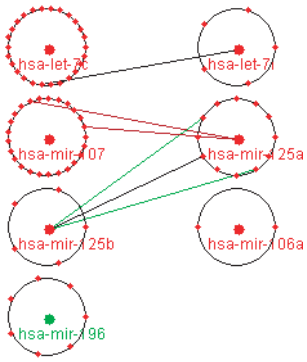


Figure 4.2: The visible part of the microRNA network, after selecting hsa-let-7c, hsa-let-7i, hsa-mir-107, hsa-mir-125a, hsa-mir-125b, hsa-mir-106a, hsa-mir-196.

The user can also select some gene from the respective list. If the gene, has already been drawn on the periphery of a cycle, all microRNAs that regulate it and have not been yet drawn are placed in the lists and the respective lines are drawn to indicate the relationship. On the other hand, if the gene has not been already drawn, all regulating microRNAs are drawn and the gene is placed on the periphery of the first of them. The newly added microRNAs are not expanded (no genes are placed on their periphery and no additional lines are drawn) in order to keep the image as clear as possible. The list with the names of the genes, is also the way to identify some specific gene fast. If the gene has been already drawn, it will be highlighted. Extending the network of Figure 4.2 by selecting the gene ENSG00000048740 leads to the drawing in Figure 4.3.

For genes with many related microRNAs a large number of new cycles can be produced. Thus, after a right click on a microRNA, an option to remove it from the network appears. The next microRNA to draw will be placed in the space that has been released.

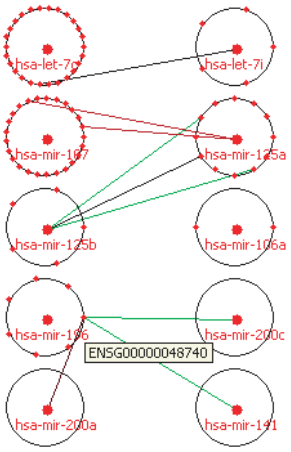


Figure 4.3: Expand the network of Figure 4.2 by selecting gene ENSG00000048740.

The last selected term (either gene or microRNA) is colored with green, while all other terms are in red. Drawing the network this way offers many advantages. First of all, the user sees only the portion of the network that interests him. Secondly, since this image is being created in a step-by-step interactive process, it is easy to remember where each term is and thus, maintain a mental map of the network. The bipartite structure of the network is obvious: microRNAs appear in the middle of the cycles and are a bit larger, while genes appear on the periphery. Due to the convention that genes on the periphery are regulated by the microRNA in the center, many edges are implied and do not have to be drawn. This leads to a much clearer image. With some other drawing technique which displays all edges for the part of the network in Figure 4.2, 82 more edges would have to be drawn. In addition this method is fast since all that is required is to determine if some term has been already displayed or it has to be placed due to a user's selection. Storing all drawn terms in a hashtable solves this problem and since the number of visible elements is relatively small, updating and repainting are performed in real time, which is of major importance. Finally, storing and restoring the image can be performed efficiently. We do not have to save the coordinates for the painted vertices. It suffices to store the sequence of selections made by the user, since the placement is predetermined. This allows us to save the produced image in a mall text file when the user exits the application. When the application restarts, the last-used image is restored and the user can continue from the point he left. Of course an option to clear everything and start from scratch is also supported.

4.1.3 Gene oriented drawing

A similar approach has been adopted to create a gene-based image. Only this time genes are placed in the center of cycles and regulating microRNAs are placed on the periphery. This leads to less compact results. While 207 circles (one for every microRNA) are enough to display the whole network using the microRNA-oriented approach, with this approach 1073 circles (as the number of genes) have to be used. On the other hand, for the same number of cycles we expect far fewer edges, since most genes have only a few regulating microRNAs (rarely more than 5, maximum 11). A typical embedding appears in Fig. 4.4.

Respectively, if some microRNA is selected, all genes that are regulated by it become

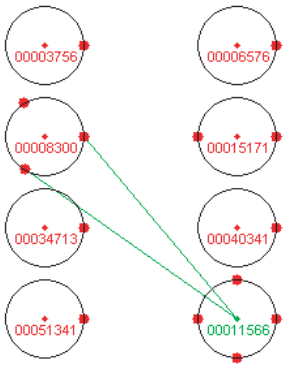


Figure 4.4: The network, after choosing genes with id 3756, 6576, 8300, 15171, 34713, 40341, 51341, 11566.

visible. Since the average degree of a microRNA is much larger(maximum 49, average 11) we expect relatively more new cycles with genes to appear. This is illustrated in Figure 4.5

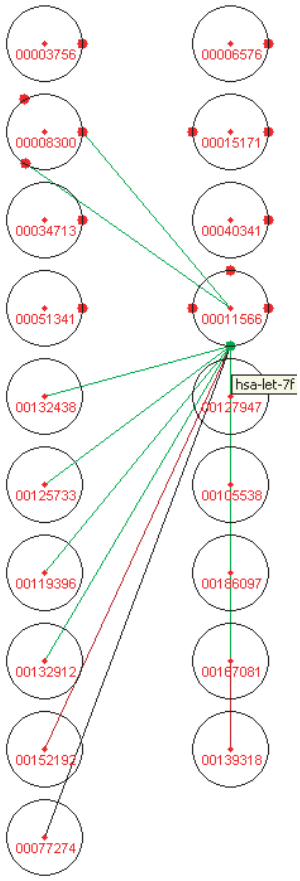
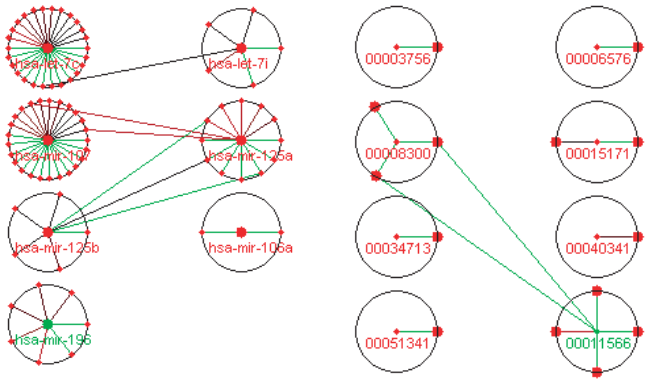


Figure 4.5: Expansion of the network in Figure 4.4 with microRNA hsa-let-7f.

4.1.4 Weighted edges

As discussed in Section 1.2 every prediction has an associated score, which represent its "strength". For the edges in the network, it is important to show this information. The median value of the scores for the edges is computed. All edges that have score below the median are colored in red and the brightness reflects the value. The lower the score, the brighter the color. For edges that have score equal to the median black is used, while edges with higher score are colored using green. Again, the brightness reflects the value. Since viewing edges is important, we have added some features to help the user. By clicking on some vertex, all the associated edges which are initially visible are hidden. So even if a complex image has been created, the user has the ability to unclutter it. By clicking again, the edges are revealed. For the edges that are implied by the placement of vertices on the periphery of a cycle, a similar feature is available. By right-clicking on the vertex in the center of the cycle, an option to show all these edges becomes available. In Figure 4.6 this is depicted for the already shown networks.



(a) for Figure 4.2 (b) for Figure 4.4

Figure 4.6: include in-cycle edges.

4.2 Drawing the Gene Ontology DAG

4.2.1 properties and needs

In order to display the G.O. dag efficiently one has to satisfy three main goals:

- Display the whole graph
- Maintain the hierarchy
- traverse through the graph easily

The G.O. dag contains about 20000 terms and is regularly updated. Any traditional graph drawing technique would fail to draw such a large graph in a comprehensive manner within the limited size of a typical computer monitor. In Chapter 3 a method to obtain a tree equivalent to the G.O. dag was presented. This process creates a tree with a much higher number of vertices ($\sim$ 100.000) but enables us to use treemaps which is among the most space-efficient techniques to draw large trees.

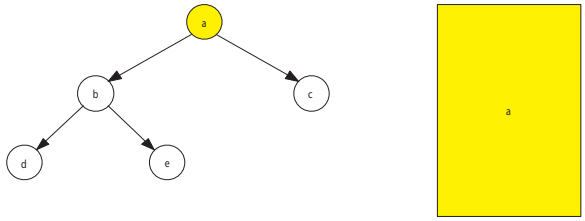
4.2.2 Treemaps

Treemaps were originally proposed by Johnson and Shneiderman in 1991 [18],[19]. Every node of the tree is represented by some rectangular area. The main idea is to place each vertex inside its parent's rectangle starting from the root and proceeding in a depth-first-search way. The area of one vertex is distributed to its children. During this process, layers of hierarchy are created. On the upper, visible layer only the leaves appear. The description of the main treemap algorithm is the following:

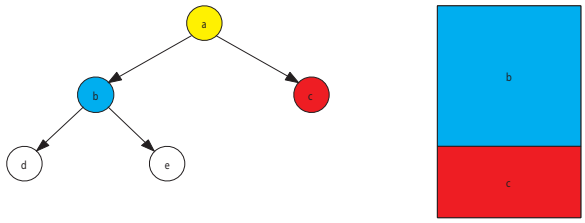
TREEMAP(r,area)
1 r.area=area;
2 divide *area* to the children of r
3 **for each** child c of r
4 **do** TREEMAP(c,$c's$ computed area)

CALLTREEMAP(treeRoot,drawingArea)
1 TREEMAP(treeRoot,drawingArea)

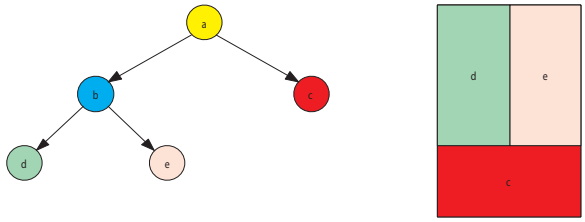
An example of drawing a simple tree using treemaps appears in Figure 4.7.



(a) Assign the area to the root a



(b) Divide a's area to b and c

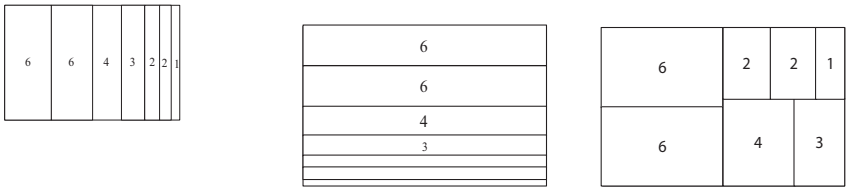


(c) Divide b's area to d and e

Figure 4.7: Draw a tree with treemaps.

If instead of the root some other vertex is used for calling the algorithm, the subtree rooted under this vertex is shown. This means that the algorithm itself can be used for zooming-in and out. Depending on the way the space division in step 2 is accomplished, many variants have been proposed. The original idea [18],[19], also known as *"slice and dice"* was to cut the area and divide it to slices with size proportional to some metric defined for

every vertex. This metric typically is the weight in case of weighted trees and the number of leaves in the subtree for non-weighted trees. For better visualization results the orientation of the slices alters from horizontal to vertical between subsequent layers. *Slice and dice* tends to create long, thin rectangles for vertices with many children. In an attempt to avoid this, Bruls, Huizing and Wijk [20] introduced *squarified treemaps*. They divide the area using a heuristic which aims to create rectangles with *aspect ratio=length/width* as close to 1.0 as possible. In Figure 4.8 an example of how 6 leaves with weights {6,6,4,3,2,2,1} are placed over their common parent appears.



(a) horizontal slice and dice

(b) vertical slice and dice

(c) squarified

Figure 4.8: (a) horizontal slicing (b) vertical slicing (c) squarified.

In [18] the authors propose nesting in order to show internal vertices. An Additional advantage of nesting is that the hierarchy is revealed and some space is left unoccupied, so that labels can be placed. Nesting, is accomplished in the following manner. Whenever some area is assigned to a vertex, a small border is placed on its boundary and only its interior is given as drawing area for the children (Figure 4.9). With nesting, the treemap for the tree of Figure 4.7 is as in Figure 4.10.

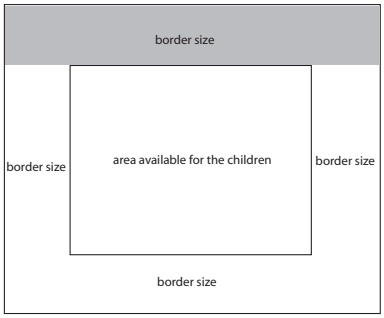


Figure 4.9: Addition of a border: The gray area can be used to place a label for the vertex. Only the interior of the border is made available for the children.

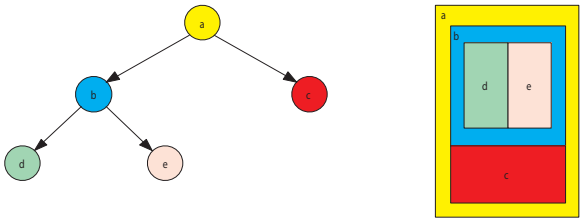


Figure 4.10: The treemap for the tree of Figure 4.7.

4.2.3 Features available for the Gene Ontology treemap

The *squarified treemaps* algorithm has been implemented. The *slice and dice* variant could have also been used but for such large trees (where the number of children varies widely) it has practical disadvantages (placement of labels), which are discussed here.

Borders are supported according to user's request, and their size can be adjusted by the user. Borders are only placed if the available interior area is non zero, in other words if there is enough space to place all the descendants of a node. For borders with sufficient size (9 pixels or more), large enough letters can be placed in the upper side, so the names of the terms are also displayed. The need to use borders for displaying names, was the main reason to discard the " slice and dice" algorithm, since long and thin rectangles would appear and the use of horizontal borders would become impossible. An additional obstacle towards the placement of labels is the vertical slicing, which gives almost zero-width rectangles.

Zooming in and out on specific subtrees is supported and works in the manner already described. If some vertex is clicked, the treemap algorithm runs using this as root, and the subtree under the vertex is displayed in the whole drawing area.

Due to its large size, displaying the whole G.O. at once assigns very small rectangles to vertices that are deep in the tree even if small borders are used (Fig. 4.11). The result is that these vertices can not be easily found , even if their location is known. What is clear though in this case, is the hierarchal structure of the G.O. The first layers are also clearly visible, so the build up of a a mental map about the backbone of the G.O. is possible.

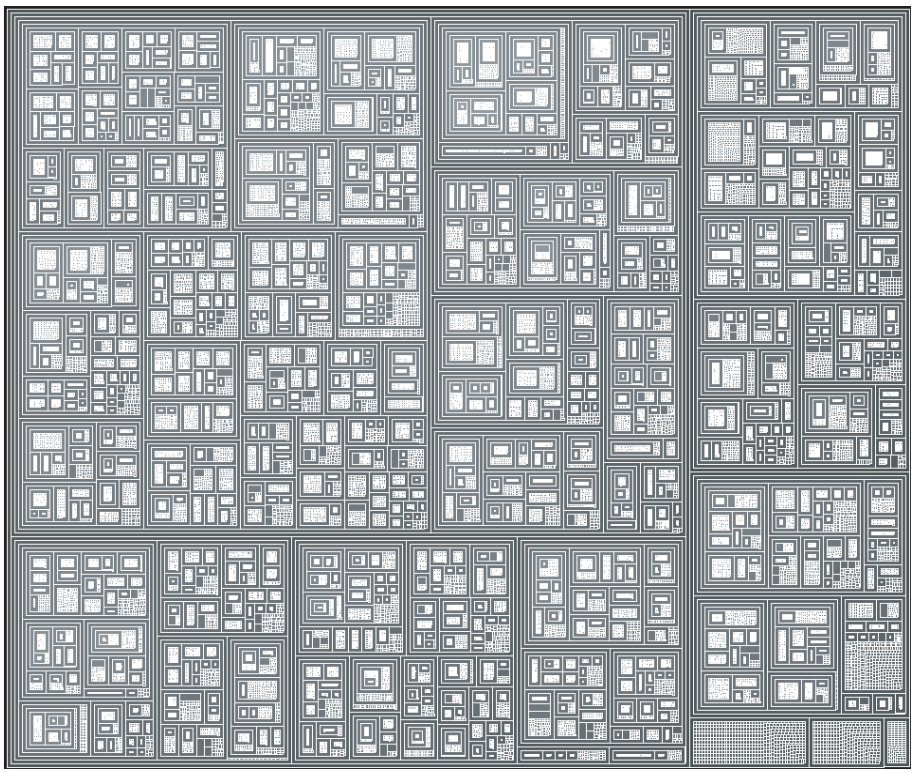


Figure 4.11: The whole G.O.: It is difficult to select vertices that are deep in the hierarchy.

To overcome the difficulty of excessive detail, the user can set the number of visible layers. While this depends on the size of the drawing area, displaying 6 or 7 layers at a time seems to provide reasonable browsability. Vertices that are below this depth can be revealed

with zooming. In Figure 4.12 the first 6 layers are visible. After clicking on *"metabolism"*, vertices which are deeper in this subtree become visible (Fig. 4.13).

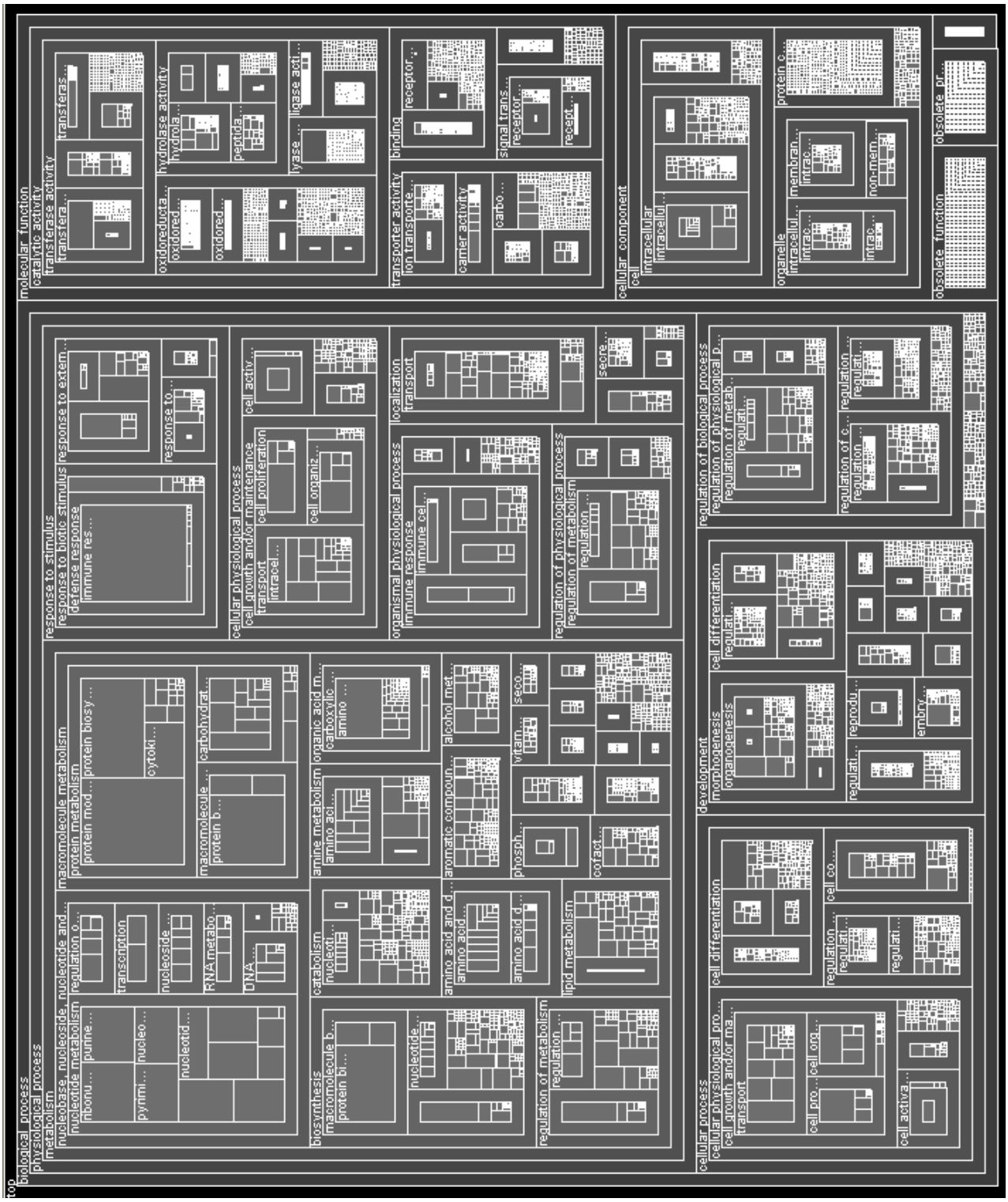


Figure 4.12: The first 6 layers of the Gene Ontology.

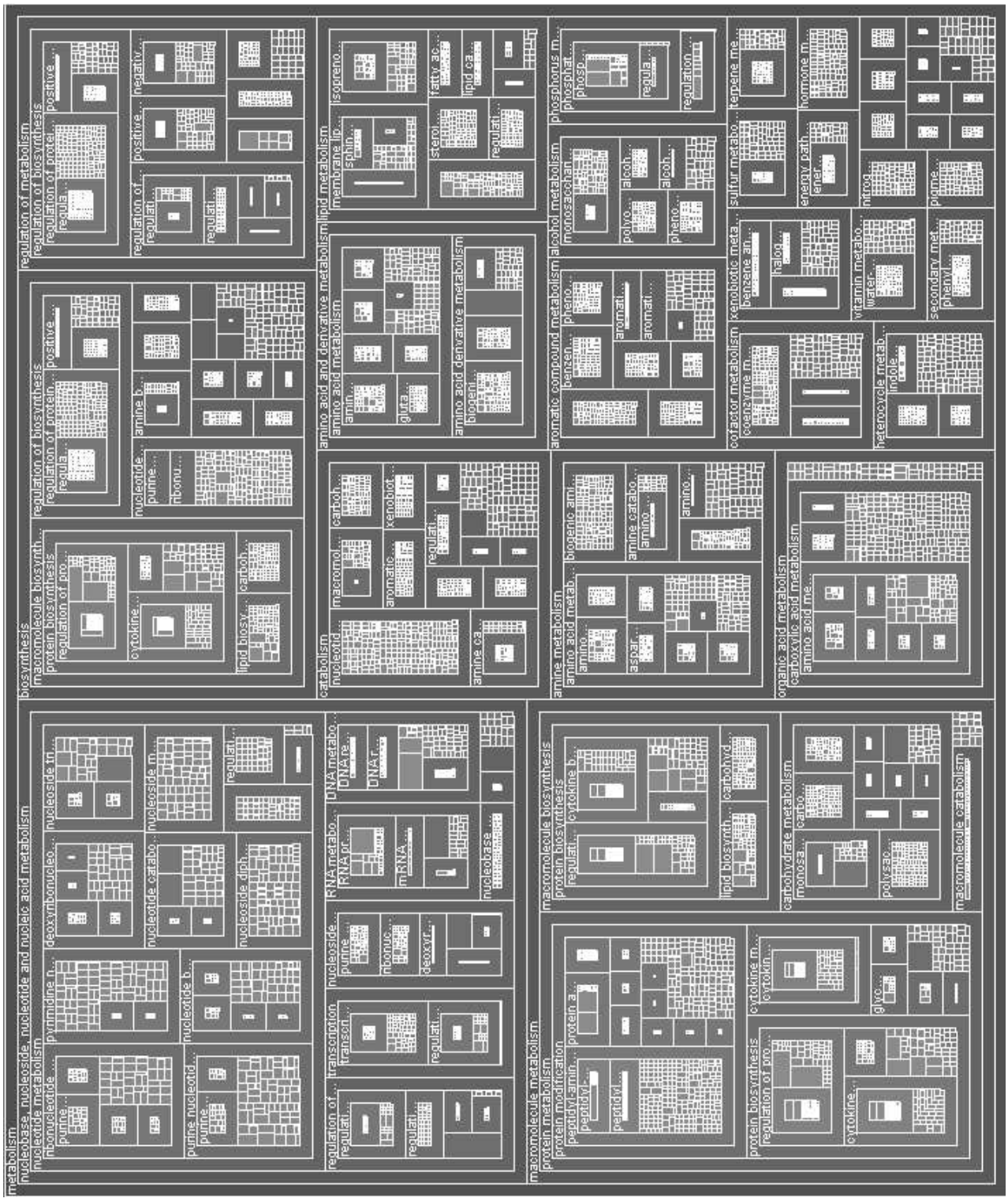
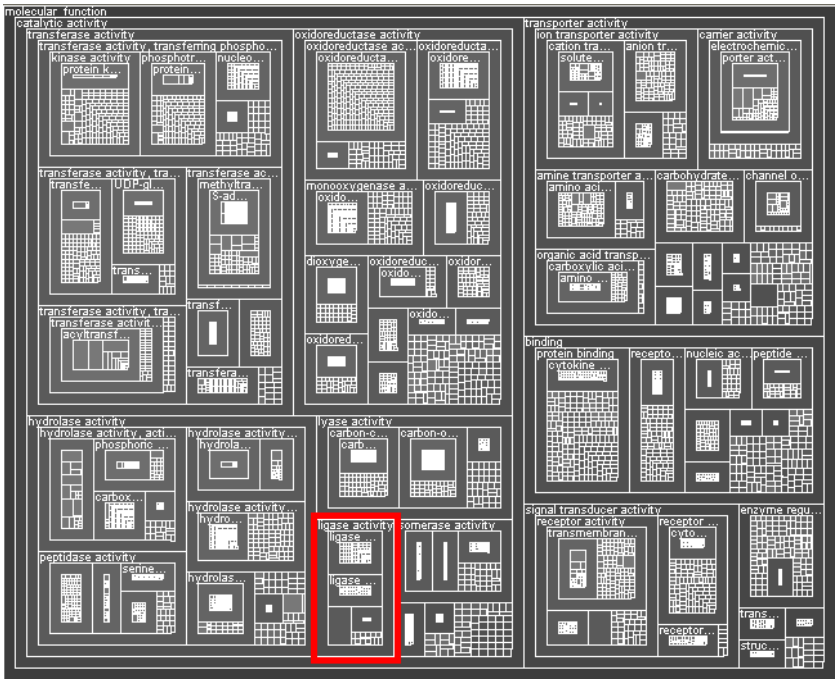
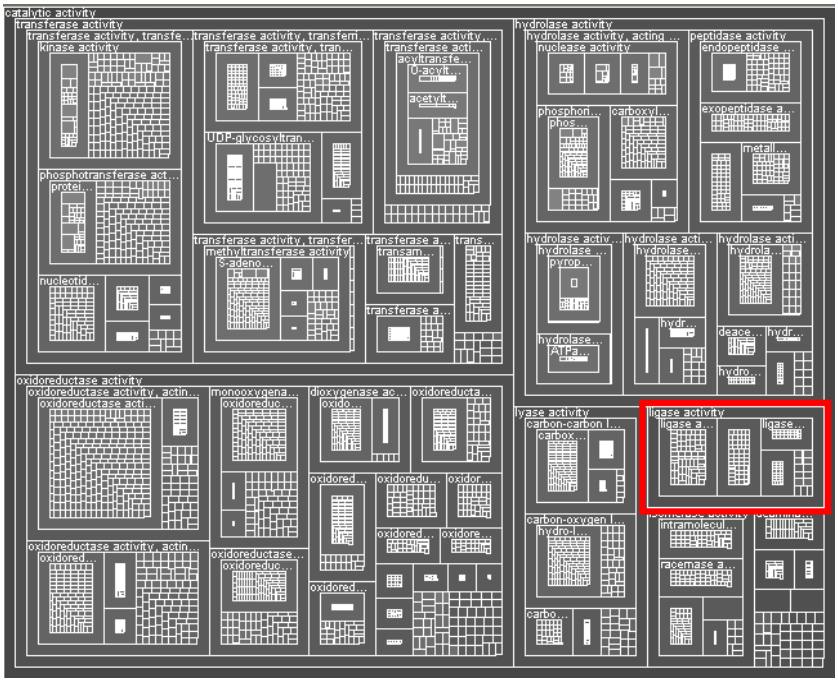


Figure 4.13: After zooming on *metabolism*: 6 first layers.

During zooming-in and -out the position of vertices alters, and following some specific term can become difficult(Figure 4.14). The *slice and dice* layout maintains the ordering and overcomes this difficulty, but the discussed problems, make it inefficient for trees of large size.



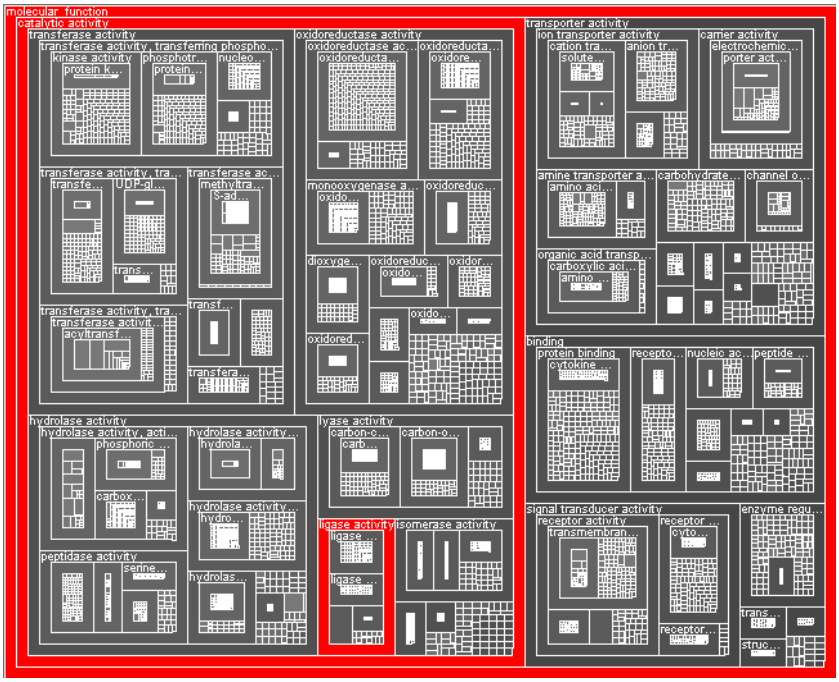
(a) *"ligase activity"* in the *" molecular function "* branch



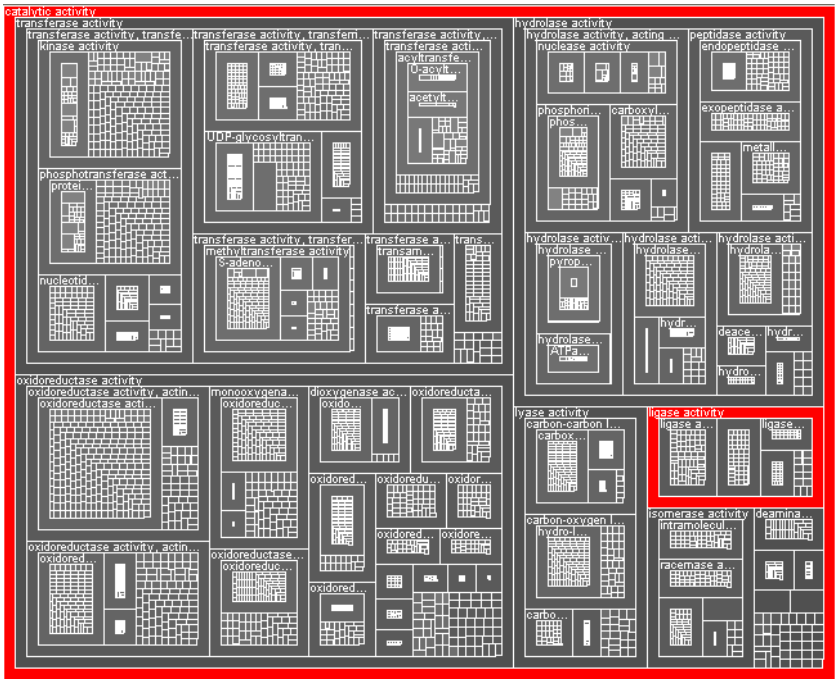
(b) *"ligase activity"* in the *" catalytic activity "* branch

Figure 4.14: The position of a term can alter while zooming.

An alternative solution to the problem of locating some term is highlighting. By right clicking on some vertex of the treemap, a number of options becomes available. "Highlight term" causes the path from the root to the specific vertex to be colored in red. The term of interest can be easily identified at any level(Figure 4.15).



(a) *"ligase activity"* in the *"molecular function"* branch



(b) *"ligase activity"* in the *"catalytic activity"* branch

Figure 4.15: Using highlight, terms are easily identified at any layer.

Exploration of deeper layers of the tree is straightforward and is supported by clicking on terms, as described. Returning to upper layers though cannot be performed in the same way since all vertices that are higher in the hierarchy are not visible. This action is supported through three options, which become available by right-clicking anywhere in the treemap region.

- **go to parent.** The visible subtree alters to the subtree rooted under the current root's parent.
- **go two levels up**. Similar to the previous but goes two layers up. Useful for faster traversing.
- **go to root.** Return to the original root of the tree and display the whole G.O.

So with treemaps, we can provide an efficient way for the visualization of very big trees, like the G.O. Additionally, as described, the image can be adjusted (size of treemap, size of borders and number of visible layers) to better match the user's needs. With drawing techniques for the microRNA regulatory networks and the G.O. available, the next step is to combine these drawing and give answers to more important biological questions.

4.2.4 Integrated visualization of the microRNA regulatory network and the Gene Ontology

Up to now we have discussed how the user interacts with the microRNA regulatory network and can obtain information about it. We have also described how one traverses the G.O. with treemaps. The most interesting and important part of this work though, is the ability to explore the two graphs concurrently. This can be performed either based on specific G.O. terms that are of interest or some genes and microRNAs the user focuses on. The first direction is important when the biological concept (related to the G.O.) is set and we search for genes and microRNAs that are related to it. On the other hand, we may want to search for the G.O. terms that are related to some gene or microRNA. MicroRNAs are not related directly to the G.O. , only through the genes they regulate. But since they control genes, it is essential to take this indirect relation also into consideration.

Gene Ontology based analysis of the microRNA regulatory network

Every gene in the network is related to at least one term in the G.O. Genes that do not have this property have been removed (4.1.1). On the other side, for many G.O. terms there is a number of genes that are related to them. The most obvious requirement is that the user can ask to see these genes. For every term, a list of genes that are related to it has been created initially in order to ensure a fast response to this very common question. In order to ask this question, the user has to right click on some term and select the "Show Genes" option. All genes that are related to the selected G.O. term and belong to the visible portion of the network are highlighted (colored in green) in the left pane of the user interface. An instance is illustrated in Fig. 4.16.

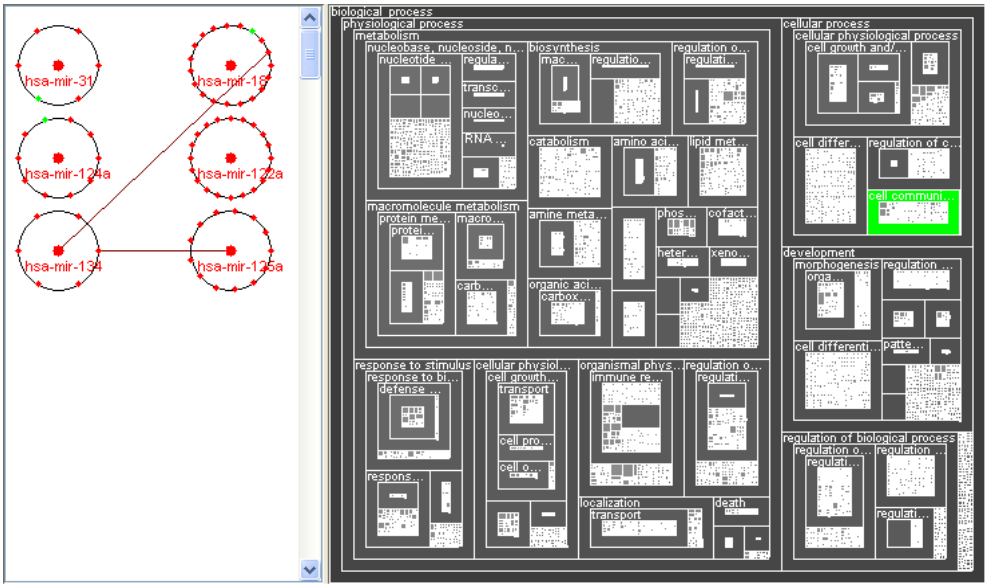


Figure 4.16: Highlighted genes on the cycles on the left(87470, 68383, 101384) are related to "cell communication".

With the drawing method used for the microRNA regulatory network, only a portion of the regulatory network is visible in the left pane. This way the user has the ability to see only the part of interest and not the whole complex network. This means that it is possible that some genes related to a selected G.O. term are hidden. Since it can also be the case that the user wants to see the sub-network that is related to some term in the G.O., an option to "Expand & show Genes" that are related to it before highlighting is also available(Fig. 4.17).

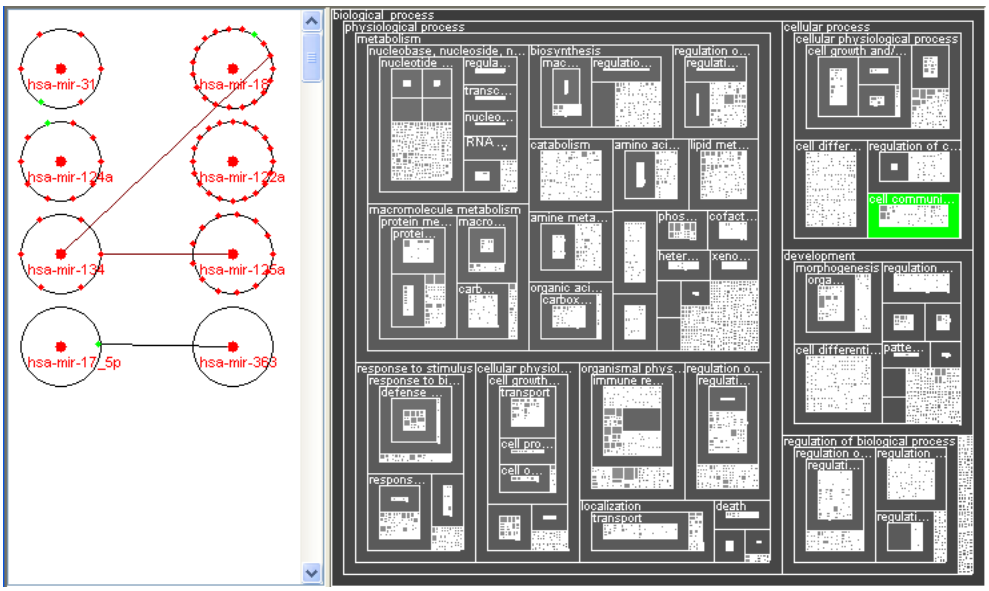


Figure 4.17: The gene with id 112577, not visible in Fig. 4.16 is drawn.

While microRNAs that regulate highlighted genes can be easily spotted if genes are on the periphery of the circle surrounding it, for microRNAs that are connected to genes with

lines this can be difficult, especially if the user has selected to hide some edges. Thus, we provide an additional option called "Show Genes and microRNAs" to highlight all microRNAs that regulate genes associated to a given G.O. term as well (Fig.4.18).

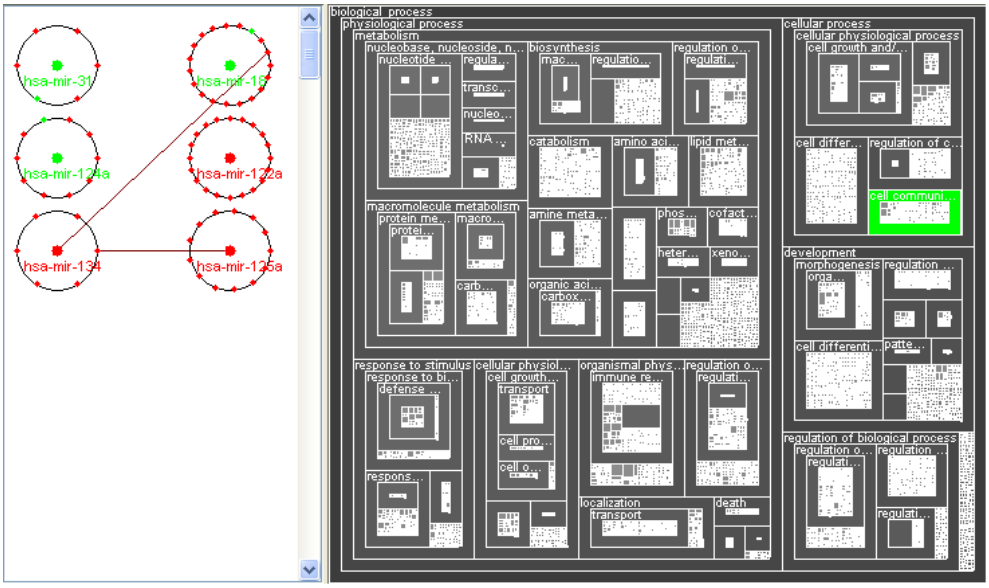


Figure 4.18: Highlight visible genes and microRNAs related to *"cellular communication"*.

As in the previous case, some genes or microRNAs may not be present in the currently visible part of the network. Using the same option ("Expand & show Genes") as before, the user can expand all genes and make visible all microRNAs that are related to the term (*"cellular communication"* for example) before highlighting with the "Show Genes and microRNAs" option and finally obtain a result similar to that of Figure 4.19.

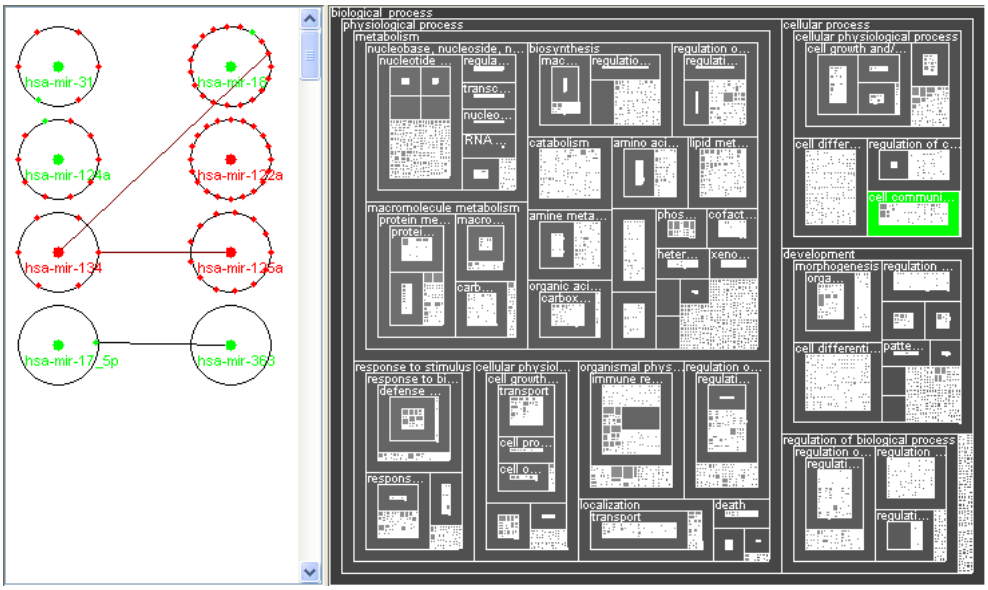


Figure 4.19: All genes and microRNAs that are related to *"cellular communication"*.

In order to store as much information as possible for the behavior of a gene, the set of terms that are related to it are as specific (and thus in the lower layers of the G.O. hierarchy) as possible. Thus, almost no gene is related to any G.O. term that belongs to the first 3 or 4 for layers of the hierarchy. These layers are mainly used to create the various categories where more specific terms are placed. In a typical scenario the user would be interested in terms in the first layers of such a category, rather than some very specific term. Thus, there is also an option to see not only genes (and microRNAs) that are related to some term, but genes (and microRNAs) that are related to any G.O. term in a specific branch. In Figure 4.20 for example, all genes (in the existing portion of the network) that are related to some term in the branch of *"biological process"* are highlighted. While no gene is directly related to the very general term *"biological process"*, the vast majority of the genes have something to do with a term in the branch under it.

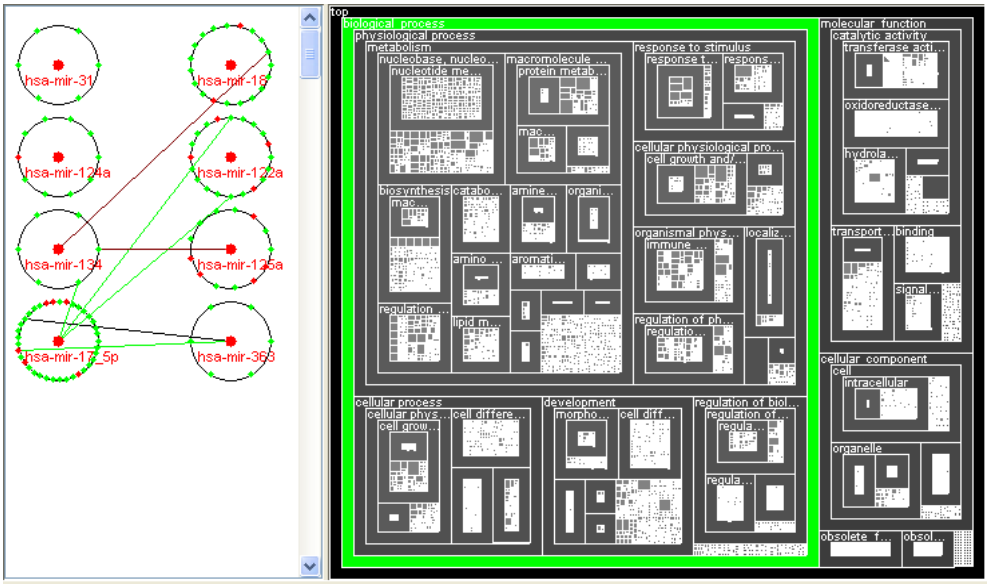


Figure 4.20: All visible genes that are related to some term in the branch of *"biological process"*.

Again, similar options to "Expand & show genes under the term" and "Show Genes and microRNAs under the term" are also available.

Search for G.O. related properties of the microRNA regulatory network

While we can see the genes and microRNAs that are related to some G.O. terms, this is not the only way to combine these objects. The opposite direction which relates one gene or microRNA to many G.O. terms, is also important. We want to answer the question: which G.O. terms are related to some given gene/microRNA. If some gene is selected it is expanded (if not already) and highlighted, as described earlier. In addition to this, the set of G.O. terms that are related to it are highlighted using bright green. Because of the option to show only some layers or because of very small size, some terms may not be visible . In this case, the closest visible ancestor is highlighted using dark green (Fig. 4.21). This allows the user to know that some term in the highlighted subtree is related to the clicked gene and if the branch appears to be interesting the user can zoom in to identify the exact term, which has been assigned bright green color(Fig. 4.22).

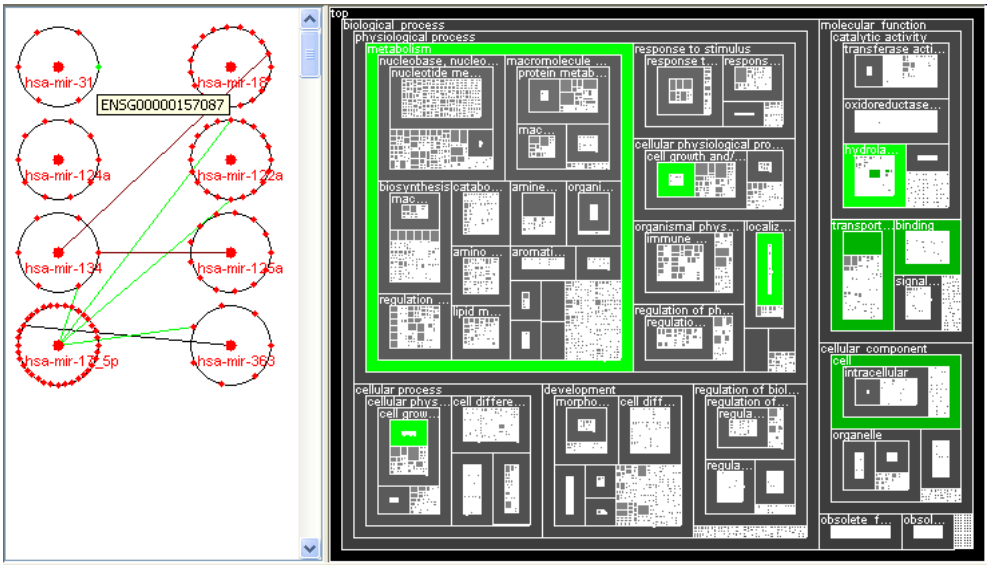


Figure 4.21: All terms related to gene *ENSG00000157087*. Note that "cell" is in dark green.

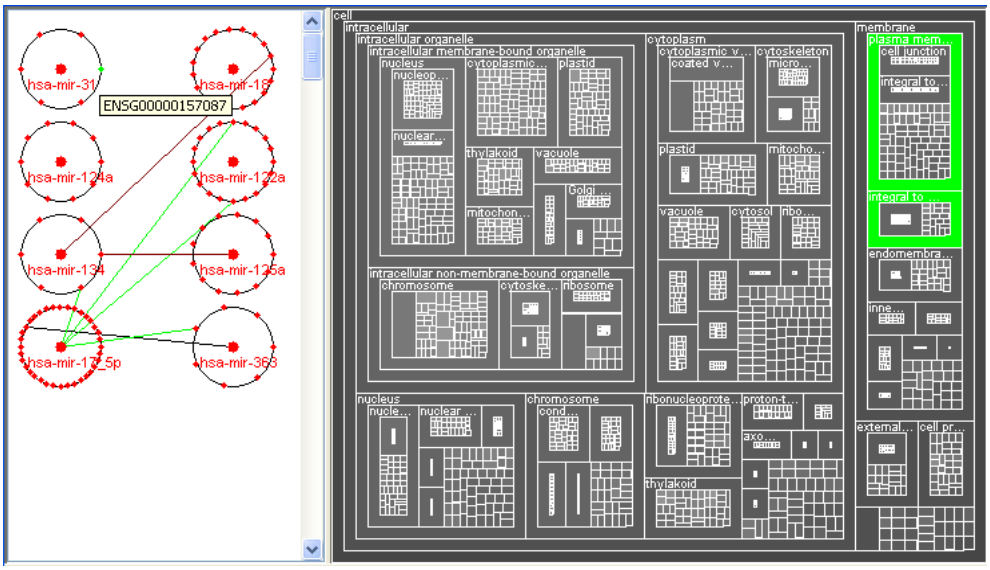


Figure 4.22: All terms that were not visible and caused *"cell"* to be colored in dark green in Fig 4.21, are clearly shown after zooming-in *"cell"*.

In the same sense, for some microRNA one can be interested to see the G.O. terms that are related to the genes it regulates. Similarly, if some microRNA is selected it is expanded(if not already), highlighted and the set of G.O. terms that are related to at least one of the genes it regulates are colored. The coloring is the same as in the previous case. For *hsa-mir-17_5p*, the microRNA with the maximum number of predicted regulating genes, the related part of the G.O. appears in Figure 4.23.

For microRNAs with many regulating genes, a large portion of the G.O. is highlighted. This feature allows the user to see the terms that are related to the genes, but doesn't give any quantitative idea about the number of genes that are related to any term. This

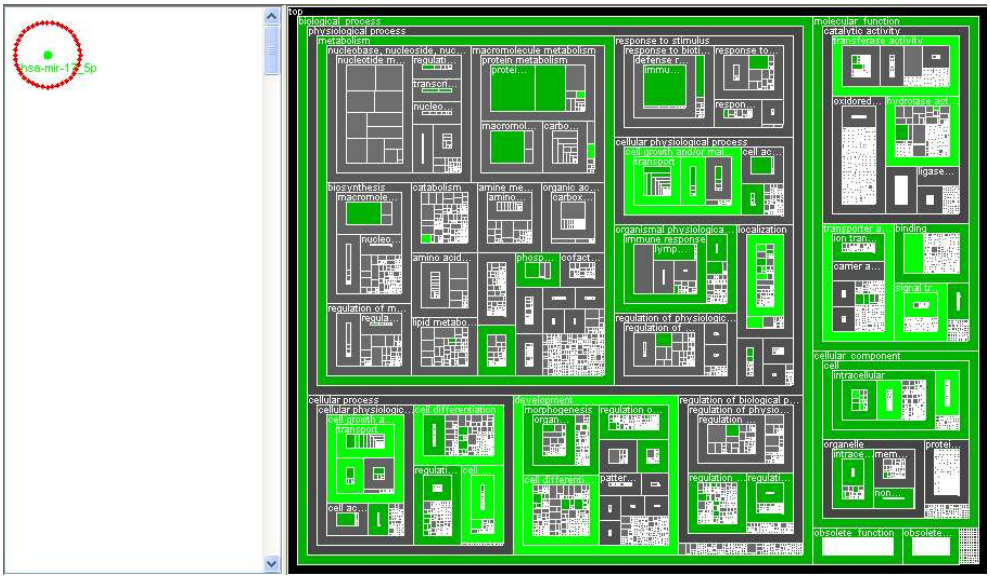


Figure 4.23: All G.O. terms that are related to microRNA *hsa-mir-17_5p*.

is important since the microRNA regulatory network is based on predictions, thus a false prediction gives false G.O. relations. In the analysis of their predictions [2], [6], [7], the authors point out that genes being predicted to be targeted several times by microRNAs are significantly less likely false positive predicted. Hence, if there are more than one regulating genes that are related to some term, we can be more confident that this term is really related to the microRNA of interest. So the user is provided with an option to iterate through the genes that are regulated by some microRNA and for each of them display the related G.O. terms with some animated highlighting. Now, the user can see how often some term is highlighted and obtain the additional information. Using this feature for example it has been found that 17 out of the 21 genes that are regulated by *hsa-mir-98* are related to *"binding"* - or some more specific term in its branch. This observation makes the hypothesis that *hsa-mir-98* is important for binding more trustworthy.

4.3 microRNA co-regulation network: a new graph

An interesting, indirect relationship that exists between microRNAs comes through the genes they regulate. It is biologically useful to know the common gene targets (if any) two microRNAs have. Apart from giving some information, this could potentially define some metric of similarity between microRNAs. In order to obtain and store this information, we create a new graph, which we call *microRNA co-regulation network*:

- **Vertices:** the set of microRNAs
- **Edges:** for every pair of microRNAs with common targets, an edge is inserted. All edges have as attribute, the set of common gene targets between the two end points

4.3.1 Structure of the graph

The graph consists of 207 vertices, the microRNAs, and 893 edges. There exist 27 microRNA-vertices with no neighbors, which can be divided in two classes.

- **Isolated:** the set of microRNAs that regulate no gene at all:
  {*hsa-mir-147, rno-mir-129*, hsa-mir-223, hsa-mir-366, hsa-mir-364, hsa-mir-126, hsa-mir-99b, hsa-mir-99a, hsa-mir-208, hsa-mir-187, hsa-mir-299* }

- **Unique regulators of genes:** The set of microRNAs that are the unique regulators of the genes they regulate (so no edge is added). This is a much more interesting branch. While the straightforward comment that these genes can only be regulated by the specific microRNAs is not expected to be true, we can conclude that at least for the genes whose predicted regulators are the following microRNAs, the prediction is probably incomplete:
  {*rno-mir-329, rno-mir-347, hsa-mir-362, hsa-mir-137, hsa-mir- 142-5p, hsa-mir-212, hsa-mir-215, hsa-mir-216, hsa-mir-341, hsa-mir-127, hsa-mir-138, hsa-mir-126*, hsa-mir-193, hsa-mir-218, hsa-mir-344, hsa-mir-139*}

For the remaining 180 microRNAs we have the 893 edges, each of which represents at least one and at most 11 common genes. The remaining subgraph is very dense, and thus an efficient visualization is hard to be obtained. We have adopted a number of different graph drawing techniques which accompanied by user interaction can give good results.

4.3.2 A supergraph based on vertex- disjoint cliques

A graph with such density probably contains a number of smaller cliques. Since the simplification of the structure is important to visualize this graph we identified all vertex disjoint cliques and constructed a supergraph in the following manner:

- **Vertices:** For every microRNA that does not belong to any identified clique, a vertex is inserted. For every clique, a superVertex that represents the clique is also inserted.

- **Edges:** For every pair of microRNAs that regulate one or more common genes an edge is inserted between them. Similarly for every two cliques C_a and C_b with vertices $v_1 \in C_a$ and $v_2 \in C_b$ such that v_1 and v_2 regulate common genes an edge in inserted. Finally, an edge is also inserted between a vertex v and a superVertex sV if v shares a common gene with some vertex in sV.

We simplify the discrimination by coloring simple vertices using red and superVertices using black. This supergraph consists of 133 vertices 21 of which are cliques and 431 edges. The sizes of the cliques vary from 3 to 11. The subgraph that contains all simple vertices and no superVertex is very sparse. There are only 52 edges between the 112 vertices and the structure is that of a forest. In order to display this portion of the graph clearly we place all vertices on the periphery of a cycle. The order they appear on the cycle is based on a depth first traversal which guarantees the absence of edge crossings. With this ordering, we also manage to place the edges close to the periphery and leave the interior empty. The result appears in Figure 4.24

In the interior, we place a second, co-centric cycle with all the superVertices-cliques on its periphery. This subgraph is very dense and almost any ordering of the vertices gives a very cluttered image. We try to improve the quality of the result by reducing the number of edge crossings. We do this by applying the circular drawing algorithm that has been proposed in [36]. The main idea of this algorithm is to find an ordering for the vertices by removing edges that will belong in the interior of the cycle. After this we are left with

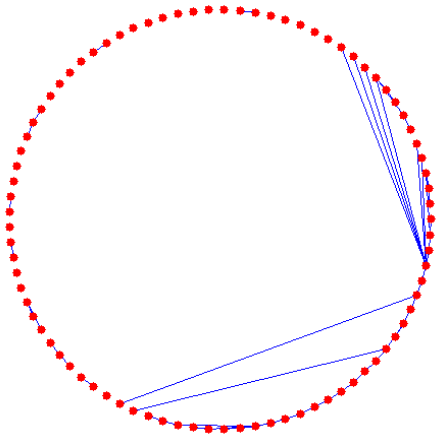


Figure 4.24: The subgraph which contains only microRNAs and no cliques is a forest.

a sparser graph and the vertices are placed using a longest path heuristic (like a depth first search traverse). The most important property of this algorithm (which was actually the inspiration for designing it) is that, in case of a biconnected outerplanar graph, the subgraph obtained after the edge removal phase is a hamilton path and the placement of the vertices in this order gives a cross-free result. The algorithm in pseydocode is presented here.

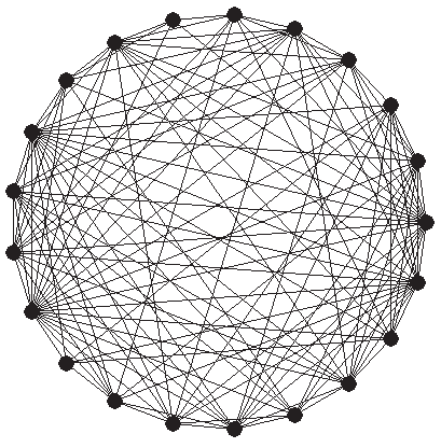
CIRCULAR(Graph $G(V, E)$)

```
1   bucket sort the vertices of G by ascending degree into a table T
2  while G has more than three vertices
3     do
4          find a lowest degree vertex u, prefer a neighbor of an as recently removed vertex as possible
5          visit u's adjacent nodes consecutively
6         for  every pair of nodes
7             do if  an edge between them exists
8                    then  place the edge in the removalList
9                    else  place an edge between the pair and also in the removalList
10         remove u from G
11         update T
12  restore G to its original topology
13  remove from G all edges in the removalList
14  perform a dfs on G
15  place the longest path on the embedding cycle
16  place the remaining vertices next to as many neighbors as possible
```

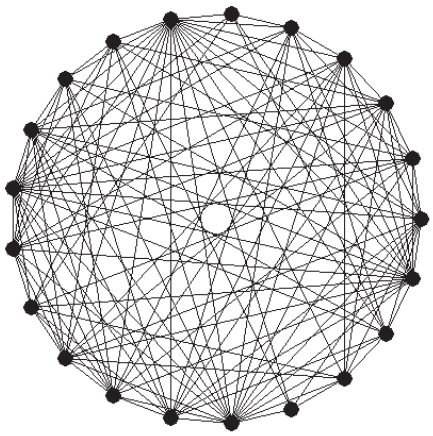
Table T is used to accomplish step 4 faster. The time complexity of the algorithm is linear to the number of edges in the graph. While the algorithm runs for all graphs, in [36] the authors suggest its use for biconnected graphs only. They also provide a framework for the visualization of non-biconnected graphs which makes use of the described algorithm for the biconnected components and gives better results. Details and complexity analysis can also be found in [36]. In Appendix A we present experimental evaluation of the algorithm. We also discuss two variants we have developed for the execution time reduction.

The fact that the graph of interest is very dense, means that the graph after the removal phase will be also relatively dense. So the ordering that is aimed by the edge removal phase is quite possible to be missed by the longest path heuristic. Thus, we have attempted to repeat the edge removal phase for as long as edges are actually removed, hoping to better guide the longest path. This has the risk that different orderings can be targeted by various iterations of the removal phase. But it is also possible that the new ordering which has information about a good ordering from the removal of edges in previous iterations will be a good one as well. During the implementation we have constructed the graph in two different ways and obtained two different adjacency lists. For the original *Circular* algorithm the numbers of edge crossings were 2014 and 1898 while for the recursive-edge-removal variant the results were 1890 and 1948. The resulting images have obvious differences. For different datasets it can also be the case that one of the two methods gives considerably better results, so we allow the user to choose among them.

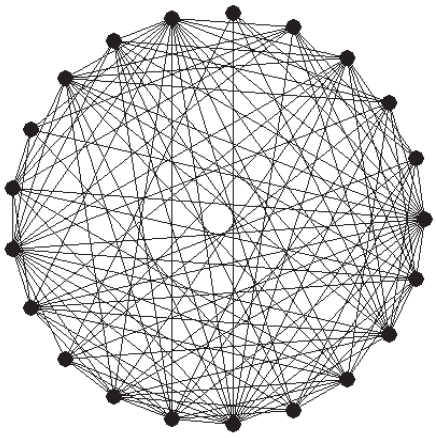
A third method for obtaining a circular embedding for the superGraph is to place the vertices on the periphery using a heuristic that tries to spread the degrees. This approach aims to create a visualization where the density does not vary for different regions of the embedding. The results for all three methods appear in Figure 4.25



(a) Circular



(b) Recursive Circular



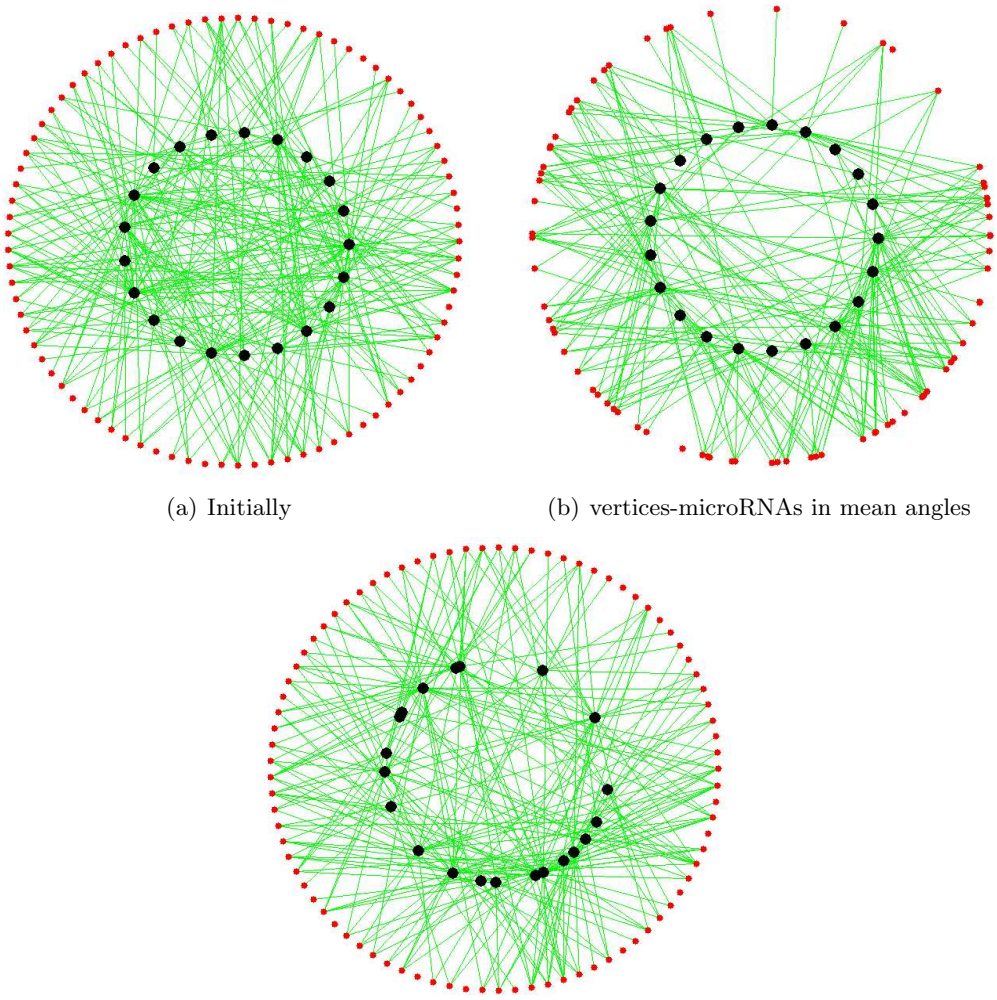
(c) Spread degrees

Figure 4.25: The three circular methods.

It is obvious that we have three sets of edges.

- between microRNAs, which are colored using blue
- between superVertices-cliques, which we draw using black colour
- between a microRNA and a superVertex, which are painted in green

The user can select to see only one of these sets or any of their combinations. For the first set, a dfs based ordering of the vertices gives a clear visualization. For the second set, the circular drawing algorithm that was described earlier manages to give an as clear as possible result. For the third set, the user has the option to move either the microRNAs or the supervertices to the mean value of the angles that are created by their neighbors and the horizontal line that passes from the (common) center of the cycles. This appears in Figure 4.26



(a) Initially

(b) vertices-microRNAs in mean angles

(c) superVertices-cliques in mean angles

Figure 4.26: Place vertices or superVertices in mean angles.

If some combination of sets of edges is required, then the user can select any of the algorithms for the cycles, but -most important- he has the ability to interact and move any

vertex to some position that he finds more appropriate.

4.3.3 The full co-regulation graph

Moving the mouse over a superVertex one can see the microRNAs it consists of. A more detailed view can be obtained if the user selects to "Change Mode" and expand all superVertices. Then the microRNAs contained in every superVertex are placed on the periphery of a circle in the position the superVertex was. As already said the density of this network is very high, and creates a very complex image. Thus, we have chosen to hide all edges initially, and allow the user to expand the edges that are related to microRNAs of interest. A small rectangle is drawn in the middle of every edge. If the user moves the mouse over some rectangle, the genes that are co-regulated by the two microRNAs-end points are shown (Fig. 4.27).

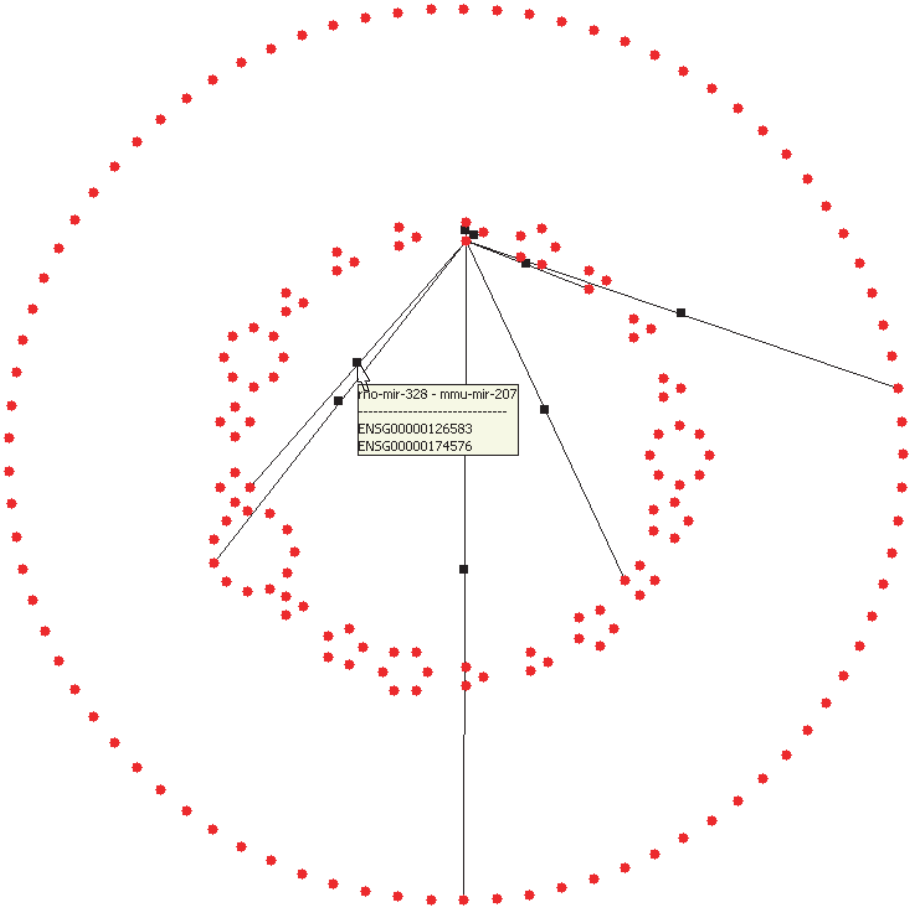


Figure 4.27: The common genes of two microRNAs.

A click on the rectangle, causes the terms in the G.O. treemap that are related to these genes to be highlighted. In case more than one common gene exist, we iterate through these genes and use animated highlighting as described earlier.

Again the user can interact with the system and alter the positions of the vertices. He can also place the vertices on a grid, based on their current positions. This method makes use of the whole drawing area and guarantees that no two vertices are placed annoyingly

close. For every vertex we also add a small random displacement from the center of its cell. We do this to avoid overlapping edges between vertices in the same row or column of the grid (Fig. 4.28).

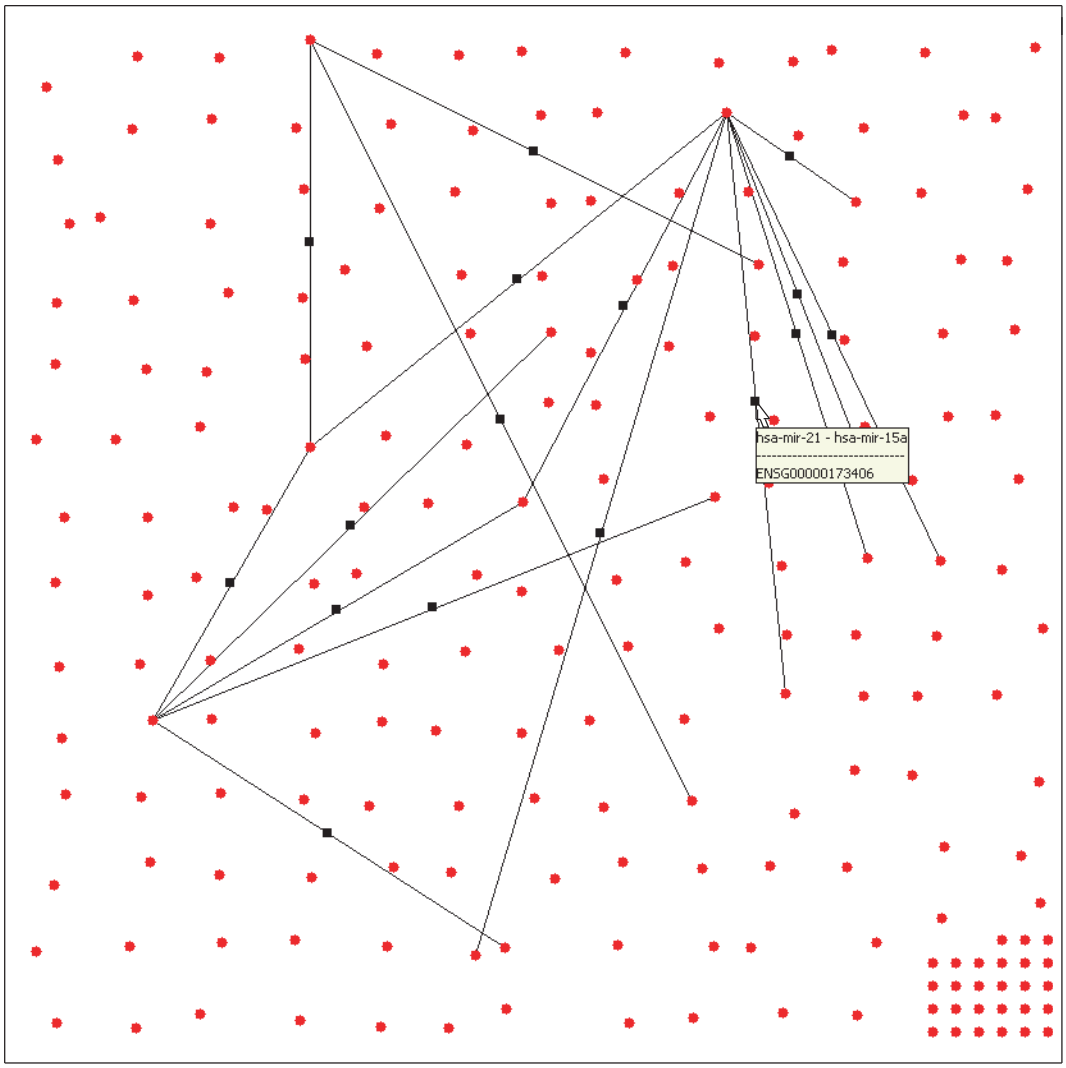


Figure 4.28: Use a grid for the placement of vertices.

4.3.4 Summary

In this Chapter we have presented graph drawing techniques that draw the Gene Ontology hierarchy and the microRNA regulatory network. Some methods have been chosen and others designed based on the structure of the graphs and the properties we want to emphasize. We also allow the user to interact with the system and modify the embeddings so that he obtains a result closer to his needs and aesthetic criteria. Finally, we have introduced a new graph, which is constructed based on correlations between microRNAs and suggested various algorithms for its visualization.

Chapter 5

Conclusions and Future Work

We present a tool for the analysis and the visualization of a new, promising concept in the field of biology: predicted microRNA regulatory networks. The structures and properties of these networks have not been analyzed. With this work we aim to provide a tool that allows researchers to search for such properties. Since microRNAs regulate genes, more information about the biological role of microRNAs can be extracted through the G.O. With knowledge about the functionality of genes stored in the G.O., we want to search for unknown properties of microRNA regulatory networks based on it. We have designed and implemented two different tools for these goals. The first makes use of common G.U.I. elements such as lists and expanding trees. With this tool, the user can easily apply a number of filters on the data set and get genes and microRNAs of interest. For these, the answers to a number of questions about the relations with the G.O. and other related genes and microRNAs are available on request. The second tool draws the G.O. hierarchy and the regulatory network based on graph drawing techniques in a manner that can be easily modified by the user to match his needs. Advantages of this tool are: a) the ability to interactively select and draw the desired part of the microRNA network in full detail and hide all other information, b)the ability to visualize functional properties of multiple sets of genes in a standardized and understandable way, c) the efficient browsing of the G.O. hierarchy, d) the use of animation to show indirect relations between microRNAs and G.O. terms and finally, e)the fact that the user has rather than text, a visual representation, which is helpful in understanding the overall structure and much easier to remember.

For both techniques, we have tried to produce self-explanatory, user friendly environments which allow the user to easily obtain the information of interest and search for new properties. Responding to any action in real time was another important goal we managed to achieve.

We also introduce as a new proposal the construction of the co-regulation network for the microRNAs. This graph is much smaller in size and stores information about the genes that are simultaneously regulated by pairs of microRNAs. This can be interpreted as some kind of similarity in the functionality of microRNAs.

The most useful test with this tool is to see if the network that is created by the known predictions actually is in some way related to the G.O. Importing this information in the tool can be used for answering the question and this would transfer the contribution of this work from predicted networks to reality.

Currently we support data in the format presented by John et. al. in [2]. Supporting other formats which are used by other predictors and probably include other organisms

would be a useful addition. After this, it would be interesting to define some metric for the comparison of different predicted networks.

Some graph-theoretic metric for the comparison could be a function of the percentage of common edges, the size of the maximum common subgraph and others. Such a procedure could lead to useful results. For common subgraphs between different prediction algorithms for example, the belief that the predicted regulations actually exist becomes stronger. Notice that since all vertices are labelled with the respective genes/microRNA names a 1-1 mapping for vertices between different networks exists. This means that we avoid difficult problems like that of graph isomorphism.

Another interesting approach to this comparison would be to define some G.O.-based metric. The basis for such a metric could be the number of G.O. terms that are common for all genes that are regulated by a single microRNA or, more general, the multiplicity of appearances of G.O. terms in the genes related to some microRNA. Working with such a metric on the microRNA co-regulation network which was introduced and analyzed in Section 4.3 can give interesting results . These results could be used in addition to those obtained by the predicted microRNA regulatory network and could help us reach useful conclusions. Even if it is unlikely that this metric can be helpful as an evaluator of the predictions (which would not be the goal) it could give useful clues about the existence of relationship between the microRNA regulatory networks and the G.O. hierarchy.

In terms of visualization, it would be useful to adopt other graph drawing techniques for the microRNA regulatory network and compare the results. A different approach towards the visualization of the network can be based on families of microRNAs, i.e. groups of microRNAs that are defined based on similarities in the sequences of the mature microRNAs.

Another set of regulations that can be used in parallel with those predicted, is the regulations that have been experimentally verified. Even if for the time being their number is very limited, an option to show only these in the G.U.I. would be useful in the future as an evaluator of the predictor's accuracy.

An important property of microRNAs is their tissue specific expression, [60]. The data available from the related gene expression measurements could be collected and the visualization of this feature should be realized.

Finally, it is challenging to discover new graph drawing techniques that will be able to visualize hierarchies with the size of the G.O. Dag or tree.

Bibliography

[1] R.C. Lee, R.L. Feinbaum, and V. Ambros. C. elegans heterochronic gene lin-4 encodes small rnas with antisense complementarity to lin-14. *Cell*, 75:843–854, 1993.

[2] B. John et al. Human microrna targets. *PLoS Biology*, 2, 2004.

[3] Alan Dove. Opening a portal: The strange new world of microrna. *Genomics and Proteomics*, 2005 June.

[4] B.P. Lewis et al. Prediction of mammalian microrna targets. *Cell*, 115:787–798, 2003.

[5] S. Pfeffer et al. Identification of virus encoded micrornas. *Science*, 304:734–736, 2004.

[6] A.J. Enright et al. microrna targets in drosophila. *Genome Biol.*, 5, 2003.

[7] A. Stark et al. Identification of drosophila microrna targets. *PLoS Biology*, 1, 2003.

[8] M. Kiriakidou et al. A combined computational-experimental approach predicts human microrna targets. *Genes dev.*, 18:1165–1178, 2004.

[9] D. Gruen et al. microrna target predictions across seven drosophial species and comparison to mammalian targets. *PLoS Comp. Biol.*, 1, 2005.

[10] A. Krek et al. Combinatorial microrna target predictions. *Nature Genetics*, 37:495–500, 2005.

[11] M. Rehmeister et al. Fast and effective prediction of microrna/target duplexes. *RNA*, 10:1507–1517, 2004.

[12] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[13] G. Di Battista and R. Tamassia. Algorithms for plane representations of acyclic graphs. *Theoret. Computation Scien.*, 61:175–198, 1988.

[14] R. Tamassia G. Di Battista and I. G. Tollis. Constrained visibility representations of graphs. *Informat. Process. Letters*, 41:1–7, 1992.

[15] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE, Transact. Syst. Man. Cybern*, 11(2).

[16] D. J. Gschwind and T. P. Murtagh. *A Recursive Algorithm for Drawing Hierarchical Directed Graphs*. Department of Computer Science, Williams College, 1989.

[17] M.J. Carpano. Automatic display of hierarchized graphs for computer aided decision analysis. *IEEE, Transact. Syst. Man. Cybern*, 10(11):705–715, 1980.

[18] B. Johnson and B. Shneiderman. Treemaps: a space-filling approach to the visualization of hierarchical information structures. In *Proc. of the 2nd International IEEE Visualization Conference*, pages 284–291, October 1991.

[19] B. Shneiderman. Tree visualization with tree-maps: a 2d space-filling approach. *ACM Transactions on Graphics*, 11(1):73–78, September 1992.

[20] D. M. Bruls, C. Huizing, and J.J. van Wijk. Squarified treemaps. In *Proceedings of the joint Eurographics and IEEE TVCG Symposium on Visualization*, pages 33–42, October 2000.

[21] W. T. Tutte. How to draw a graph. *Proceedings London Mathematical Society*, 13:743–768, 1963.

[22] P. Eades. A heuristic for graph drawing. *Congr. Numer.*, 42:149–160, 1984.

[23] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Softw.-Pract. Exp.*, 21(11):1129–1164, 1991.

[24] A. Frick, A. Ludwig, and H. Mehldan. A fast adaptive layout algorithm for undirected graphs. In R. Tamassia and I. G. Tollis, editors, *proc. of GD'94, LNCS*, volume 894, pages 388–403.

[25] R. Davidson and D. Harel. Drawing graphics nicely using simulated anealing. *ACM Trans. Graphics*, 15(4):301–331, 1996.

[26] Janet M. Six and Ioannis G. Tollis. A framework for user-grouped circular drawings. In G. Liotta, editor, *Proc. GD'03, LNCS*, volume 2912, pages 135–146. Springer-Verlag, 2004.

[27] Janet M. Six and Ioannis G. Tollis. Effective graph visualization via node grouping. In *Proceedings of the IEEE Symposium on Information Visualization 2001 (INFOVIS'01)*, 2001.

[28] U. Doğrusöz, B. Madden, and P. Madden. Circular layout in the graph layout toolkit. In *Proc. GD"96, LNCS*, volume 1190, pages 92 –100. Springer-Verlag, 1997.

[29] B. Madden G. Kar and R. Gilbert. Heuristic layout algorithms for network presentation services. *IEEE Network*, 11:29–36, 1988.

[30] Q. Feng P. Eades and X. Lin. Straight-line drawing algorithms for hierarchical graphs and clustered graphs. In *Proc. GD"96, LNCS*, volume 1190, pages 113–128, 1997.

[31] P. Eades and Q. W. Feng. Multilevel visualization of clustered graphs. In *Proc. GD"96, LNCS*, volume 1190, pages 101–112. Springer-Verlag, 1997.

[32] M. L. Huang and P. Eades. A fully animated interactive system for clustering and navigating huge graphs. In *Proc. of GD"98, LNCS*, volume 1547, pages 107–116. Springer-Verlag, 1998.

[33] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J.Comput*, 16,no3:421–444, 1985.

[34] U. F*ö*ssmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F. J. Brandenburg, editor, *Proc. of GD'95, LNCS*, volume 1027, pages 254–266. Springer Verlag, 1996.

[35] J.S. Vitter R. Tamassia, I. G. Tollis. Lower bounds and parallel algorithms for planar orthogonal grid drawings. In *Proc. IEEE Symposium on Parallel and Distributed Proceedings*, pages 386–393, 1991.

[36] Janet M.Six and Ioannis G. Tollis. A framework and algorithms for circular drawing of graphs. *Journal of Discrete Algorithms*, page to appear, 2005.

[37] J. M. Six and I. G. Tollis. Circular drawings of biconnected graphs. In *Proc. of ALENEX'99, LNCS*, volume 1619, pages 57–73. Springer verlag, 1999.

[38] E. Mă kinen. On circular layouts. *International Journal of Computer Mathematics*, 24:29–37, 1988.

[39] M. Baur and U. Brandes. Crossing reduction in circular layouts. In *proc. of WG'04, LNCS*, volume 3353, pages 332–343. Springer Verlag, 2004.

[40] Byong-Hyon Ju and Kyungsook Han. Complexity management in visualizing protein networks. *Bioinformatics*, 19:177–179, 2003.

[41] Moritz Y. Becker and Isabel Rojas. A graph layout algorithm for drawing metabolic pathways. *Bioinformatics*, 17(5):461–467, 2001.

[42] Alkiviadis Symeonidis and Ioannis G. Tollis. Visualization of biological information with circular drawings. In J.M. Barreiro et al., editor, *proc. ISBMDA'04, LNCS*, volume 3337, pages 468–478. Springer Verlag, 2004.

[43] Adam Arvelakis, Martin Reczko, Alexandros Stamatakis, Alkiviadis Symeonidis, and Ioannis G. Tollis. Using treemaps to visualize phylogenetic trees. In José Luís Oliveira, Victor Maojo, Fernando Martín-Sánchez, and António Sousa Pereira, editors, *proc. ISBMDA'05,LNCS*, volume 3745, pages 283–293, 2005.

[44] Ying Tao, Yang Liu, Carol Friedman, and Yves A. Lussier. Information visualization techniques in bioinformatics during the postgenomic era. *Biosilico*, 2(6):237–244, 2004.

[45] Baehrecke et al. Visualization and analysis of microarray and gene ontology data with treemaps. *Bioinformatics*, 5:84, 2004.

[46] `www.cytoscape.org`.

[47] `www.spotfire.com`.

[48] `http://avadis.strandgenomics.com`.

[49] `http://www.affymetrix.com/analysis/index.affx`.

[50] `http://www.genmapp.org/`.

[51] M. Scena, D. Shalon, R.W. Davis, and P.O. Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270(5235):467–470, 1995.

[52] J. Quackenbush. Computational analysis of microarray data. *Nature Reviews Genetics*, 2:418–427, 2001.

[53] Ramón Díaz-Uriarte, Fátima Al-Shahrour, and Joaquin Dopazo. Use of go terms to understand the biological significance of microarray differential gene expression data. In K. F. Johnson and S. M. Lin, editors, *Methods of Microarray Data Analysis III, papers from Camda '02*, pages 233–247. Kluwer, 2003.

[54] Paul Pavlidis et al. Using the gene ontology for microarray data mining: A comparison of methods and application to age effects in human prefrontal cortex. *Neurochemical Research*, 29(6):1213–1222, 2003.

[55] "http://llama.med.harvard.edu/~berriz/GoFishWelcome.html.

[56] http://www.godatabase.org/cgi-bin/amigo/go.cgi.

[57] http://cgap.nci.nih.gov/Genes/GOBrowser.

[58] http://sourceforge.net/project/showfiles.php?group_id=36855.

[59] Eric H Baehrecke, Niem Dang, Ketan Babaria, and Ben Shneiderman. Visualization and analysis of microarray and gene ontology data with treemaps. *BMC Bioinformatics*, 5, 2004. Available on-line: http://www.biomedcentral.com/1471-2105/5/84.

[60] Tomas Babak et. al. Probing micrornas with microarrays: Tissue specificity and functional inference. *RNA*, 10:1813–1819, 2004.

[61] Sandra L. Mitchell. Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Inf. Process. Lett.*, 9(5):229–232, 1979.

Appendix A

Circular Drawings of Graphs

During this work, a graph drawing problem that we were concentrated on was the placement of the vertices of a graph on the periphery of an embedding cycle in such a way that a small number of edge crossings appears. The starting point for our research was the *Circular* algorithm proposed by Six and Tollis in [36].

A.1 The original Algorithm

Six and Tollis presented in [36] an algorithm for drawing *biconnected* graphs based on Mitchell's algorithm [61] for the recognition of *outerplanar* graphs. By definition, *outerplanar* are these graphs that can be drawn on the plane with all vertices on a common face in such a way that no two edges cross. Equivalently, *outerplanar* are the graphs whose vertices can be placed on the periphery of a cycle in such a way that no edge-crossings appear. Mitchell's Theorem for the recognition of *outerplanar* graphs is based on the iterative removal of vertices, which decomposes an *outerplanar* graph, until it becomes a triangle:

Theorem A.1 *([61])*
A graph G with n nodes is outerplanar if and only if either G is a triangle or

1. *G has at most* $2n - 3$ *edges*
2. *G has at least two degree two nodes*
3. *no edge of G lies on more than two triangles, and*
4. *For any degree two node u which is adjacent to nodes v and w, the graph G minus node u plus the edge (v,w) if not already in G, is also outerplanar* □

Biconnected outerplanar graphs contain exactly one *hamilton* circuit. The placement of this circuit on the periphery of a cycle is the only way to put all vertices on the periphery with no edge crossings. They can not contain more than one since if that was the case one of the two *hamilton* circuits would have to be placed partially in the interior of the cycle thus no cross-free embedding could be obtained. But this contradicts the definition of *outerplanarity*. So at most one *hamilton* circuit exists. On the other hand, one *hamilton circuit* must always exist since the only way to draw a *biconnected outerplanar* graph in

a circular manner with no crossings is to have an edge between every pair of successively placed vertices - this is the *hamilton* circuit. In Figure A.1 an optimal (zero-edge cross) embedding is drawn and vertices u, v do not have an edge between them. If we traverse the neighbors of u in a clockwise manner we will stop at some vertex w. Similarly for the neighbors of v in a counter- clockwise traversal the last vertex is some vertex z which can not belong in the counterclockwise-arc $(\widehat{u,w})$, otherwise we would have two edges crossing. In the worst case z is the same as w. Then the two arcs are disconnected (due to outerplanarity edges (u,w) and (v,z) act as frontiers for the two arcs), with the only possible exception of w. But the removal of w definitely disconnects the graph, thus it can not be biconnected, which contradicts the hypothesis. Concluding, every *biconnected outerplanar* graph has exactly one *hamilton circuit* whose placement on the periphery on an embedding cycle gives the only zero-edge crossings result.

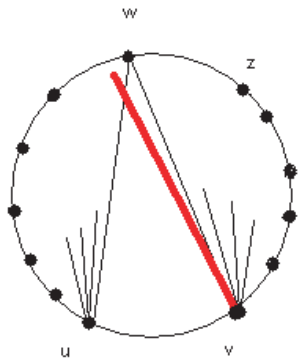


Figure A.1: If for a pair of successive vertices (u,v) in an optimal embedding an edge does not exist, then there exists a vertex w whose removal disconnects the graph.

One can find this unique *hamilton* circuit graphs by marking the edges that do not belong to it during a decomposition similar to Mitchell's. The removal of these edges leaves only the *hamilton circuit*, whose placement on the periphery of an embedding cycle gives a drawing where no two edges cross. This is the idea for the *Circular* algorithm. If G is not *outerplanar*, instead of a *hamilton circuit*, a sparser graph is obtained and some *longest path* heuristic is adopted to place vertices on the cycle. Vertices not included in the longest path are finally placed next to as many neighbors as possible. The main goal is to place edges towards the periphery of the embedding cycle. With the longest path, we ensure that many edges are placed on the periphery and the heuristic approach for the remaining vertices places even more edges towards the periphery. The pseydocode for the algorithm has been presented in Section 4.3.2. The algorithm works for all graphs, but as stated in [36] it is better to use only for *biconnected* graphs. For $non-biconnected$ graphs, better results can be obtained if the algorithm is applied on their *biconnected* components and the results are merged using other techniques.

A.2 Two variants

Here, we present two variants of the original algorithm. The goal is to reduce the execution time of the algorithm, without affecting the quality of the drawings in terms of

edge crossings. We are concentrated in the edge addition phase. The algorithm presented in [36] places triangulation edges if they do not exist for pairs of vertices that are adjacent to the selected, lowest-degree vertex. We modify this policy in such a way that fewer edges are added. Since this addition is crucial for the optimal result in case of outerplanar graphs, we do not want to destroy this. So we modify the algorithm only if we realize that the graph is not outerplanar. By Mitchell's Theorem we know that a graph is not outerplanar if at some step during the decomposition the lowest degree vertex has more than two neighbors.

A.2.1 Variant 1

The first approach is to stop adding edges as soon as we find that the lowest degree vertex has degree more than 2, which means that the graph is not *outerplanar*. We have chosen to do this aiming to minimize the insertion of edges whose contribution is doubtful. While edges that are required to guide the algorithm for *outerplanar* graphs are always inserted, if this is not the case no more edges are inserted. The edges that have been inserted in the meanwhile, serve as a guide for a good embedding for -at least- the subgraph that has been removed before determining non-outerplanarity.

A.2.2 Variant 2

A second policy we suggest for the edge addition phase, which is a bit more conservative, is the following: As already stated, in order to guarantee an optimal result for *outerplanar* graphs as long as the lowest degree vertex that is selected has degree 2, edges have to be inserted. Since for degree 2 vertices the edge addition seems to provide a good guidance we propose to insert an edge every time the selected vertex has degree 2. This is a different approach since for relatively sparse graphs, during the decomposition a degree 3-vertex can be selected during the decomposition, in which case no edge is added, but due to the removal of vertices, in some subsequent iteration a degree-2 vertex can appear again. While the first variant would not add a new edge, this variant does. As a result, it inserts more edges than the previous variant, but still significantly less than the original algorithm. On the other hand, for degree-2 vertices where the edge insertion seems useful, an edge is always inserted.

The complexity of the original algorithm is linear to the number $|E|$ of edges, $O(|E|)$. This is due to the number of additional edges and the longest path heuristic. In [36] the authors prove that during the decomposition, at most $2 * |E|$ edges are inserted. For the variants, at most one edge is inserted per iteration. Since for each iteration one vertex is removed and the algorithm stops when the remaining graph is a triangle, we have $n - 3$ iterations thus, at most $n - 3$ edges are inserted. Still though, we cannot guarantee that the decomposition phase will require less than $O(|E|)$ due to the number of existing edges that are marked for removal. So, the complexity remains $O(|E|)$ due to the total cost of the decomposition phase and the longest path heuristic. But decreasing the number of edge insertions reduces the running time.

Note that for very dense graphs the variants insert almost no edges, since no degree-2 vertex is expected to be found, so the longest path heuristic is not well-guided. This is the reason we do not use these variants for the very dense graph we presented in Section 4.3.2.

A.2.3 Outerplanar Graphs

The two variants behave identically to the original algorithm for *outerplanar* graphs. This can be justified using Mitchell's theorem(A.1). The decomposition phase in *Circular*

is identical to that of Theorem A.1. The difference is that the edge between the neighbors of the degree-2 vertex (existing or inserted) is also marked so that it is removed later. During the decomposition phase, after the removal of a vertex, which is followed by a potential edge addition- the same edge the *Circular* algorithm adds- the graph remains *outerplanar*. So always at least two degree-2 vertices exist. This means that the lowest degree vertex always has exactly two neighbors, thus always an edge is inserted. So, the behavior of the variants is identical to that of the original algorithm if the input graph is *outerplanar*. An example of the execution of the algorithm (or any of the variants) for an *outerplanar* graph is illustrated in Figure A.1.

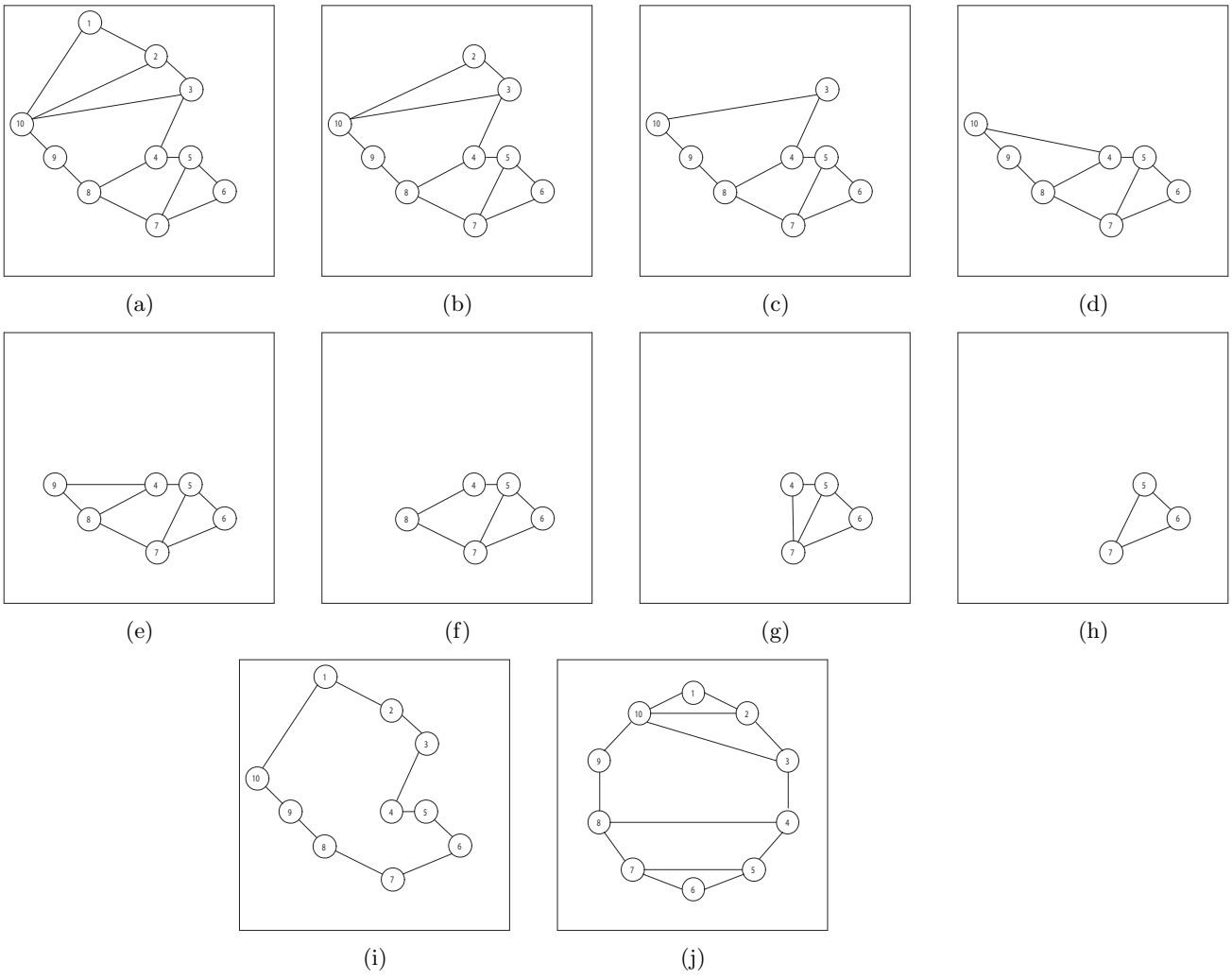


Figure A.2: (a) Select 1. Check 2,10. It exists. Mark it. Remove 1. (b) Select 2. Check 3,10. It exists. Mark it. Remove 2. (c) Select 3. Check 4,10. It does not exist. Add it. Mark it. Remove 3. (d) Select 10. Check 4,9. It does not exist. Add it. Mark it. Remove 10. (e) Select 9. Check 4,8. It exists. Mark it. Remove 9. (f) Select 8. Check 4,7. It does not exist. Add it. Mark it. Remove 8. (g) Select 4. Check 5,7. It exists. Mark it. Remove 4. (h) There are three vertices left. Stop (i) Restore graph. Remove all marked edges. A hamilton circuit is left. (j) Place all vertices in the order they appear in the hamilton circle. No two edges cross.

A.3 Experimental Results

We have implemented the original algorithm of [36] and the two discussed variants using Java 2,version 1.4. The experiments were conducted on a computer running windows^{XP}, with a Pentium 4 at 2.8 GHz and 1 GB of RAM. For all the experiments we recorded the real time requirement, the number of edge crossings and the number of added edges. We tested the algorithms on randomly generated biconnected graphs with n= 30, 50, 80, 100, 150, 200 vertices and density $d = \frac{m}{n}$=1.5, 1.75, 2, 2.25, 2.5, 2.75, 3, 3.25, 3.5, 3.75, 4. 500 biconnected graphs of each category were produced and the three algorithms were executed on all graphs. For measurements closer to the CPU time, each graph was given as input 300 times, and the lowest time response was recorded. The mean value for time, added edges and edge crossings over the 500 graphs of each category were calculated and are presented here.

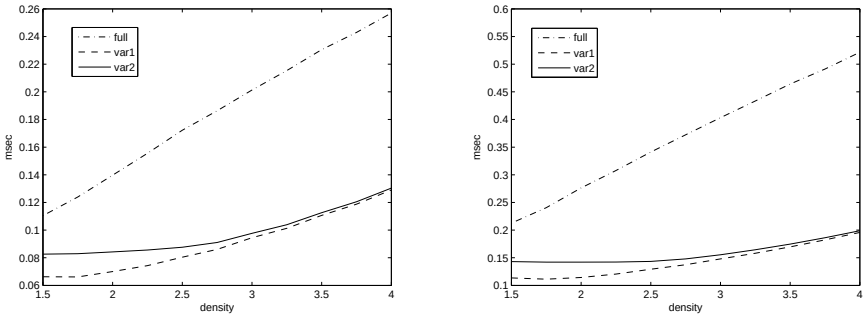
In order to randomly generate a biconnected graph with n vertices and density d the following procedure was followed: First we construct n vertices, select randomly two of them and add an edge between them. These two vertices now form a tree, which is grown by iteratively picking an isolated vertex and a vertex from the tree arbitrarily and adding an edge between them. When no isolated vertices are left, we have a tree with n vertices. Next we randomly order the leaves, and add edges between successive leaves. Now, we have a biconnected graph. The last step is to randomly add edges until we reach the required density.

As already discussed, the two variants aim to reduce the time requirement by not inserting as many edges as the original algorithm does. Comparing the two variants, the first can never add more edges than the second, while the opposite can be true as already described. This means that we expect the two variants to be much faster than the original algorithm and we also expect to see that the first variant is a bit faster than the second. Since for graphs with high density almost no edges are inserted by the variants, we can also reach conclusions about the portion of overall time that is required for them.

The behavior of the number of inserted edges is easily predicted. For the original algorithm we expect this number to increase as the density increases. On the other hand, for the variants since fewer degree-2 vertices can be found the number decreases. One important point is to see which is the density that makes both variants behave almost identically, in terms of additional edges.

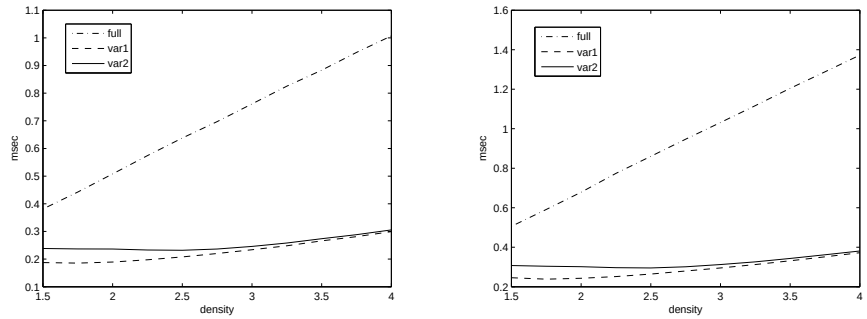
The most important parameter though, is the number of edge crossings. Even if the time reduction is significant, considerably poorer results in terms of edge crossings would make the two variants practically useless and their only contribution would be theoretic. Since the two variants modify parts of the original algorithm, the comparison of the numbers of edge crossings can reveal the most important parts of the algorithm. We can determine for example if the insertion of edges in all cases affects the quality of the result or if these insertions are useful only for degree-2 vertices.

Here we present the time versus density behavior of the three algorithms. In Figure A.3 we see the respective plots for 30, 50, 80, 100, 150 and 200 vertices. As expected, the two variants are much faster than the original. For low densities, one can observe that the second variant is slightly slower than the first. For densities ≥ 3, almost no edges are added, so the time is practically the same, and increases linearly to the size of the graph. What is important to notice is the relatively high cost of the edge addition. It leads to a larger number of neighbors to search in subsequent iterations and a larger number of edges to remove later. The considerable difference between the original algorithm and our variants shows this. Moreover, observe the second variant for $n \geq 80$. While for denser graphs we would expect that more time is required, the time remains almost constant until $d = 3$, which is due to the fact that fewer edges are added.



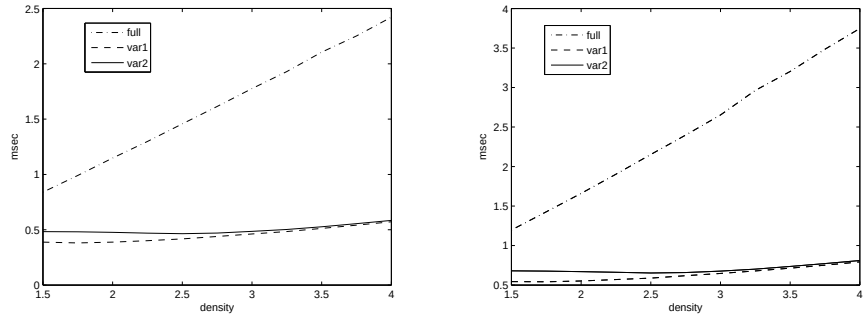
(a) $n = 30$

(b) $n = 50$



(c) $n = 80$

(d) $n = 100$

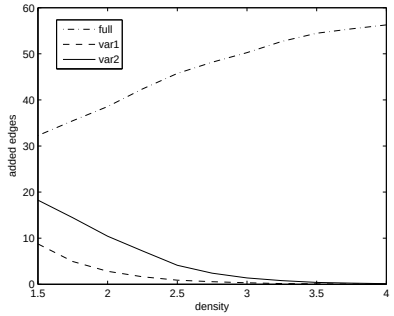


(e) $n = 150$

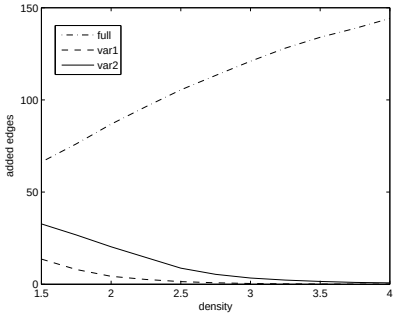
(f) $n = 200$

Figure A.3: Time vs density plots.

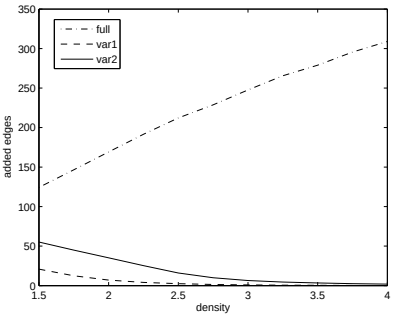
Now we discuss the number of additional edges inserted in each of the three algorithms, as a function of density. The results appear in Figure A.4. The most important observation is that as density increases, the number of edges added by the original algorithm increases as well. On the contrary, our variants add fewer edges. As more edges exist, a chosen vertex has more neighbors. The original algorithm adds more edges, since more neighbors exist, but our variants do not. As density increases fewer degree-2 vertices can be found during the decomposition, and non-outerplanarity can be determined earlier. This explains why we add fewer edges. In addition, we can observe that if the density exceeds some value (close to 3,5) both variants add very few edges. That is why the execution time for both variants is practically the same, as stated in the previous paragraph.



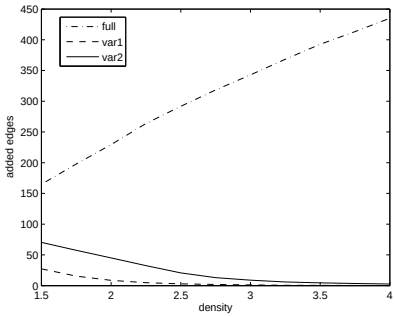
(a) $n = 30$



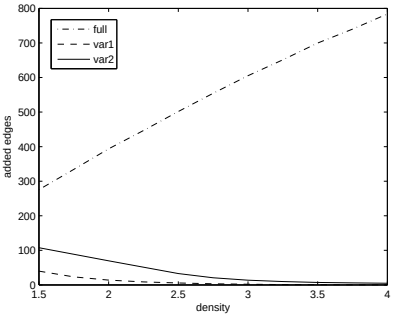
(b) $n = 50$



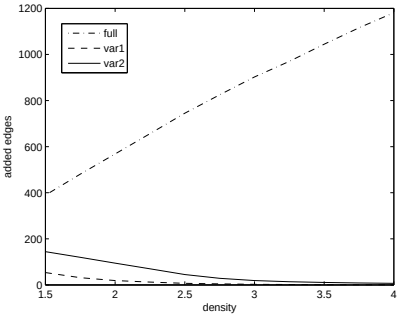
(c) $n = 80$



(d) $n = 100$



(e) $n = 150$



(f) $n = 200$

Figure A.4: Edges vs density plot.

Finally, we compare the two variants and the original algorithm in terms of crossings. Table A.1 shows the numbers of crossings in our experiments. One can see that the number of crossings is quite similar in all cases, while in some of them the result in the variants is lower than that of the original algorithm.

n	**d**		**original**	**var1**	**var2**	**n**	**d**		**original**	**var1**	**var2**
30	1,5		51,44	50,016	51,268	100	1,5		522,502	511,864	504,322
30	1,75		98,328	98,728	100,188	100	1,75		1053,87	1055,046	1055,252
30	2		167,5	168,996	170,472	100	2		1798,684	1806,116	1796,002
30	2,25		249,044	250,992	248,878	100	2,25		2776,526	2763,964	2771,506
30	2,5		359,276	358,208	357,214	100	2,5		3908,418	3914,41	3906,94
30	2,75		470,33	472,692	472,77	100	2,75		5286,564	5293,054	5307,042
30	3		615,77	620,228	618,14	100	3		6874,398	6885,434	6871,92
30	3,25		761,948	769,948	770,882	100	3,25		8647,648	8685,948	8706,068
30	3,5		949,524	959,89	960,448	100	3,5		10690,654	10671,948	10678,94
30	3,75		1132,434	1144,136	1143,712	100	3,75		12775,222	12904,668	12858,642
30	4		1362,634	1375,254	1376,2	100	4		15209,846	15254,63	15264,01
50	1,5		142,798	149.01	141,466	150	1,5		1163,444	1172,124	1134,572
50	1,75		272,602	277,478	271,486	150	1,75		2338,672	2335,024	2324,098
50	2		464,936	473,882	469,056	150	2		4030,312	4026,76	4028,704
50	2,25		689,52	702,9	700,38	150	2,25		6090,706	6120,21	6122,884
50	2,5		991,278	995,666	998,242	150	2,5		8714,342	8715,076	8729,962
50	2,75		1315,484	1325,228	1315,632	150	2,75		11787,514	11790,346	11797,076
50	3		1713,468	1722,97	1726,212	150	3		15346,128	15326,148	15299,664
50	3,25		2145,196	2148,884	2149,604	150	3,25		19324,106	19316,576	19334,348
50	3,5		2626,212	2652,54	2656,916	150	3,5		23906,758	23854,25	23841,854
50	3,75		3178,28	3192,85	3205,53	150	3,75		28696,464	28666,616	28630,45
50	4		3819,65	3818,764	3810,77	150	4		34218,774	34279,532	34272,294
80	1,5		341,98	344,028	337,612	200	1,5		1996,15	2084,338	2034,604
80	1,75		682,884	684,42	684,188	200	1,75		4116,95	4101,88	4121,742
80	2		1150,424	1153,184	1159,716	200	2		7012,814	7037,284	7048,016
80	2,25		1763,26	1771,088	1762,076	200	2,25		10829,744	10904,418	10831,07
80	2,5		2504,864	2504,876	2502,842	200	2,5		15427,068	15545,176	15608,032
80	2,75		3366,244	3384,958	3390,902	200	2,75		20725,938	21119,75	20996,27
80	3		4376,138	4366,39	4383,21	200	3		27264,7	27336,696	27252,882
80	3,25		5506,476	5529,596	5538,896	200	3,25		34434,028	34506,24	34421,012
80	3,5		6764,588	6772,768	6764,49	200	3,5		42258,122	42734,466	42469,648
80	3,75		8161,74	8209,9	8201,276	200	3,75		51151,458	51324,516	51275,662
80	4		9674,624	9698,608	9715,602	200	4		60891,504	60811,47	60779,344

Table A.1: Numbers of edge crossings for the three methods.

A.4 Conclusion

The fact that the two variants produce very similar numbers of edge crossings indicates that adding triangulation edges between neighbors of vertices with degree larger than 2 does not help much. Furthermore, from dense graphs we observe that the important action is the removal of pair edges, since the results are similar for all algorithms, even if in the variants almost no edges are added. Thus, we conclude that the key point of the algorithm is the removal of pair edges and not the addition of triangulation edges when they do not exist. The first action, removes possible shortcuts that could make the longest path heuristic (dfs) give a poor result. The addition of edges in the first iterations is useful because it guides the decomposition. The respective subgraph that will be obtained after the removal of edges will be a tree. Thus the longest path heuristic will definitely find the "correct" ordering. After a degree 3 vertex has been found, the reduced subgraph will not necessarily be a tree, so the guidance will not be clear. This means that the edges that are added for degree 3 or more vertices does not contribute. Concluding the most important parts of the algorithm are the insertion of edges until (if) the graph is found to be non-outerplanar and the removal of existing pair edges which can confuse the depth first search.