Diploma thesis

Gene Expression Profiling and Pathway Scoring

By	Marius Dürr
Supervised	Anja Wille, Eckart Zitzler
Professor	Eckart Zitzler

Content

1. Introduction

In the last few years new technologies in the field of genetics allowed a closer look into the metabolism of cells and also into the functionality of the DNA. This revealed a large amount of data for metabolic networks and also for the underlying genetic level by using DNA-chips.
To properly understand and analyze the global properties of a metabolic network, methods for rationally representing and quantitatively analyzing its structure are needed. Also its relationship to gene expression data should be examined.

The analysis of the transcriptional network of *Escherichia coli* revealed a high occurrence of certain small substructures [5]. This work tries to evaluate whether such substructures exists also in the metabolic network of the plant *Arabidopsis thaliana.*

Recent research in *Saccharomyces cerevisiae* revealed that many genes which are arranged along the central part of a pathway are coregulated [6]. It is an open question whether this coregulation exists also on other substructures of the network.

Also we like to model the data of *Arabidopsis thaliana* which has been studied and analyzed in a wide range because of its simplicity. There is gene expression data for 793 genes as well as the metabolic network containing 524 enzymes and 566 metabolites.

The work is divided into 3 parts. At first we choose an appropriate data structure for integrating the available data. It preserves all required information and allows us to work efficiently on the data. This data structure is similar to the graph-like appearance of the metabolic network itself. Nodes represent the different types of entities like enzymes, metabolites, genes and reactions. Links represent relationship between nodes, whereas the type of the nodes and the direction of the link determine its biological interpretation.
In a second step, we extract interesting information from the data structure. Therefore we search the metabolic network for small substructures. There are several different substructures [Figure 1.1]. We test for cliques, where each pair of nodes is connected through a link. We test for paths, where each node has only 2 neighbors. We test for hubs, which are single nodes that are defined by the number of links they have. And we test for unspecified modules that are defined by the connectivity of the according nodes. Further we test which of those substructures are characteristic for the network.

Finally we are looking for correlations between substructures and gene expression data. The aim is to construct a scoring function for to gene expression data, which allows conclusions about the structure of the metabolic network. Such a scoring function would simplify the work of biologists who are investigating the metabolic network.

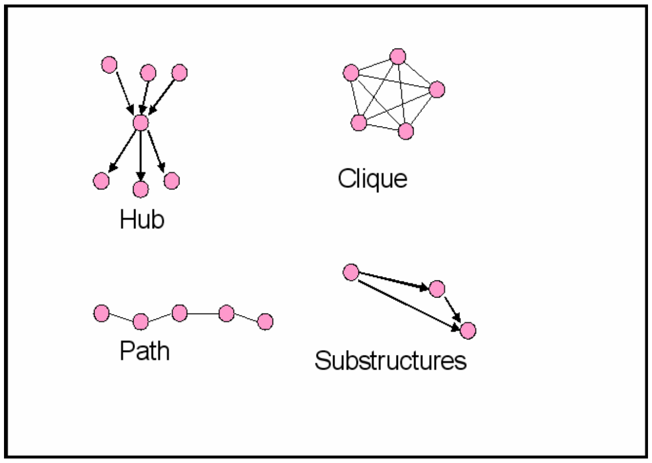

Figure 1.1: Substructures: hub, path, clique, unspecified module

1.1 Biological Backgrounds

We know that genes get transcript into proteins. This transcription still depends on mostly unclear factors, but its amount – the *activity* of a gene, called gene expression – is measurable and depends on the cell's state. The new technology of DNA-Chips tells us at once the activity of a wide range of genes. The large amount of generated data may enhance our understanding of the interactions and collaborations of the genes and their according proteins and give us some deeper insights into the functionality of cells at all.

Proteins can serve as enzymes. In this case they catalyze reactions which transform one metabolite into another. Enzymes, metabolites and their relationships are together represented in metabolic networks. Often a certain metabolite is transformed several times by different enzymes. Such chains of reactions are interpreted as metabolic *pathways*. Pathways can branch into more than just one reaction chain, but mostly there is one central part where the main reactions happen.

2. Modeling

2.1 Model

As starting point we have a database of a metabolic network consisting on 524 enzymes and 566 metabolites. Enzymes and metabolites are represented as nodes, whereas reactions of metabolites catalyzed by an enzyme are represented as links between these nodes. Some links are directed and indicate whether a metabolite is an educt or a product of a reaction. Each link belongs to a certain pathway, and there can be more than one link per connection between two nodes if this connection is part of a couple of pathways.

Finally we like to search for substructures within a graph-like model of the metabolic network. There are multiple possibilities to model the data, and we have to choose one to work with. There is a trade-off between simplicity and expressiveness of the model, and because at the time of modeling we do not know the final requirements, we decide to model also the reaction flow in contrast to the original data where the reactions are ambiguous [Figure 2.1.1].

An enzyme could be modeled as link between two metabolite-nodes; vice versa a metabolite could be modeled as link between two enzyme-nodes. Such specialized solutions should be used if we know that we do not need any other information. A more complex representation like the used database models both enzymes and metabolites as nodes. But it has no information about whether two metabolites are part of the same reaction. They also can belong to two different reaction catalyzed by the same enzyme. This information can be important in various contexts, for example in [3] metabolites correspond to nodes and reactions are represented as links between them. For keeping the model general and flexible, we decided to model the reactions too. To integrate the gene expression data we also model genes. A gene could produce one or more enzymes.

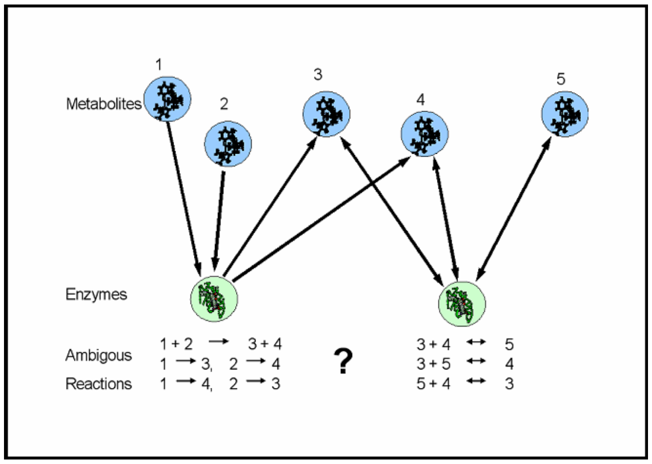

Figure 2.1.1: database-model: ambiguous context

There are several possibilities to model reactions. We could model educts and products of a reaction by using an educt and a product node to which the reaction and the reactants are connected. We can use a directed link to indicate the reactant's relationship to the reaction. Or we can omit the information whether a reactant is an educt or a product of a reaction. Since using directed links seems to be reasonable for diverse purposes, we model enzymes, metabolites, genes and reactions as nodes and the direction of a link indicates the reactant's state.

There are two types of reactions. "single-way reaction" can run only in one direction, i.e. A+B=>C+D. "two-way-reaction" can also run backwards, i.e. A+B=>C+D, C+D=>A+B or A+B⇔C+D. A metabolite is then an educt as well as a product of the enzyme's reaction. Such a two-way-reactions could be modeled with a bidirectional link between metabolite node and reaction node. Since all reactants would get a bidirectional link, we would not know which reactants belongs together on one side of the reaction equation. Therefore we split up a two-way-reaction into two symmetric single-way-reactions. This eliminates any ambiguities as shown in figure 2.1.2.

Links between nodes indicate a relationship between two nodes. All links are unidirectional. There are three sorts of links. We have links between genes and enzymes, links between enzymes and reactions and links between reactions and metabolites.
A link from a gene node to an enzyme node indicates that the gene produces the enzyme. A gene can produce more than one enzyme. An enzyme can be produced by more then one gene.
A link from an enzyme node to a reaction node indicates that the enzyme catalyzes the reaction. An enzyme can have more than one reaction, but each reaction belongs to exact one enzyme.
A link from a metabolite to a reaction indicates that the metabolite is an educt of the reaction. Vice versa a link from a reaction to a metabolite indicates that the metabolite is a product of the reaction. On the educt side as well as on the product side of the reaction there can be one or more reactants.

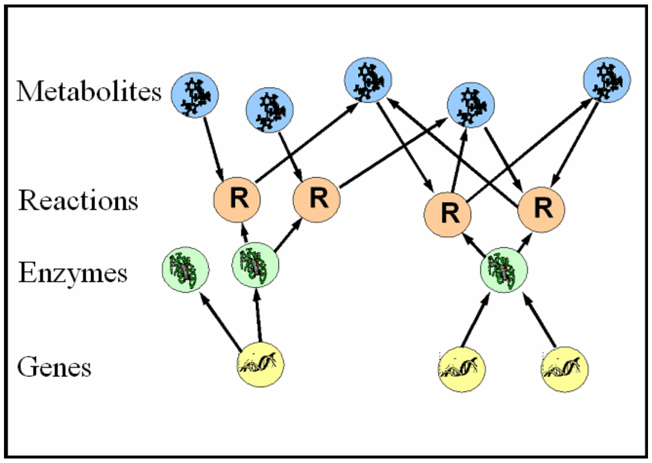

Figure 2.1.2: used model

2.2 Data Structure

The usefulness of a sophisticated data structure grows with the size of the data. Although computing performance has been steadily increasing over the last decades, it still maters whether an algorithm runs in linear, quadratic or even higher time. Data structures should be optimized with respect to the used algorithms. Simultaneously a flexible and reusable data structure can save programming time and source code size.

The presented new model of the metabolic networks is already in a graph-like form. So we choose a graph as data structure where each node manages its links by itself. This minimizes look-up time when traversing the graph.
Another approach is a set of tables like in a database. One table holds all connections between nodes. Tables have an easy handling, but look-up time is logarithmic to the number of nodes. This would lead to an immense increase of calculation time for almost all algorithms used in this work.

Within the constructed graph the algorithms are just analyzing its structure. Once the data is read, the structure of our graph does normally not change anymore. Accessing nodes and links should be cheap while manipulating the graph by inserting and deleting nodes or links may be expensive.

The links can be represented in various ways. A link may be directed, undirected or bidirectional. We can allow more than one link between two nodes or just one. Since in the model we use only directed links, we use them also for the data structure. To keep the data structure flexible we allow existing more than one link between two nodes.

Since each link is directed, it always has an origin (further “from-node”) and a destination (further “to-node”). From the view of the from-node the link is an outgoing link (further “out-link”). From the view of the to-node the link is an incoming link (further “in-link”). Depending on the algorithm we like to access only in-links, only out-links or all links of a node. So the data structure provides separated sets for in- and out-links to speed up calculation time. Simultaneously there is a set of all links where we can access the next node without caring about the direction of the link.

For fast and easy access the content of the data structure can be stored and reloaded. A file begins with the numbers of nodes and the number of links. Each node is written to one line, first a unique ID, additional information on its type and content are separated with tabs. In the second section of the file each link is written to a line, consisting only of the IDs of the two connected nodes.

2.3 Results

The conversion from the database into the data structure produced several problems. Within the original data there were links between metabolites as well links between enzymes. Some metabolites had no names which confused the database parser. Since each link belongs to a certain pathway there were multiple links for the same connection between two nodes.

The reactions and their according metabolites are not part of the original database. In simple cases an enzyme posses only two or three metabolites. We can conclude that all of them belong to the same reaction. If there are more than three metabolites connected to an enzyme we need additional knowledge to fill our data structure. For receiving this missing information we used the online database KEGG [7].

2.4 Discussion

The presented model demonstrated to be flexible enough for all tasks of this work. Due to the well defined structure, transformations into other representations are easy to implement. This is necessary for implementing fast algorithms or if we are only interested in subparts of the metabolic network, for example the relationships of the enzymes without any information about the metabolites.

2.5 Future Work

In the original database each link belongs to a certain pathway. If this information is not used, a link contains no information at all and does not have to be an independent object since it has no properties except the relationship to the connected node. Then all links of a node could be represented with one table and handful iterators instead of pointers to link-objects. The entries in the table are the connected nodes. The table consists of 4 parts. In the first part are all neighbors with an undirected connection. The second part contains the neighbors of inlinks. Part three stores bidirectional links and the last part is filled with outlinks. Iterators administer the positions of the four parts. There are iterators for the single parts as well as such for a combination of several parts. An all-links iterator covers the range of the whole table; an incoming-links iterator provides access to inlinks and bilinks etc.
This data structure is very fast and able to manage useful subsets of links.

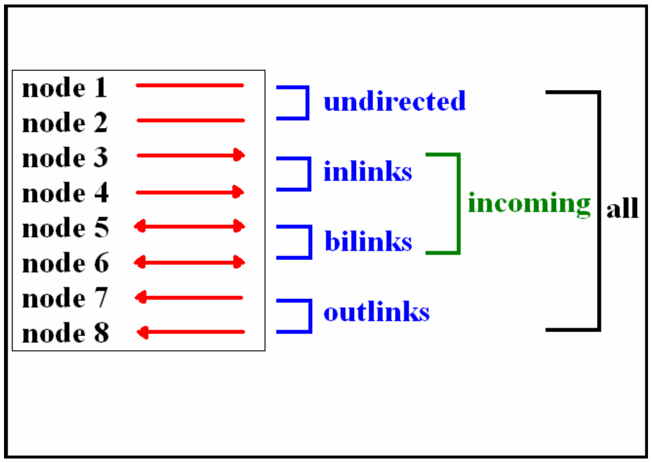

Figure 2.4.1: Improved table

3. Graph Search

The structure of a metabolic network is defined by its functionality. When considering such a network the question arises whether there are substructures which are special to this kind of metabolic representation. Knowledge about frequent substructures or modules can improve our understanding of the metabolic network. As mentioned in the introduction, similar work in genetic regulatory networks as already been done by [5]. They found several modules that are more frequent in a network than in a random graph with equal numbers of nodes and links.

The question arises whether such modules can also be found in the metabolic network. Since our model consist of 4 levels containing metabolites, enzymes, reactions and genes we transform the representation to one single level where we search for modules. For determining the significance of the module's frequencies, we need a appropriate random graph which allows us meaningful comparisons.

3.1 Methods

3.1.1 Transformation

Searching for substructures can be done in various representations of the metabolic network. Depending on the model we get different results which have to be interpreted differently. The graph contains different types of nodes. There are enzymes, metabolites, reactions and genes. Since we finally like to find correlations between gene expression data and substructures, we are looking for structures of gene nodes. The information about enzymes, metabolites and reaction is used to determine the relationships between the gene nodes. A link between two nodes, i.e. genes, represents a metabolite which is a reactant of the gene's enzymes. Since the gene nodes in our graph model are not connected to each other but only to their corresponding enzymes we have to transform the graph into the appropriate representation as shown in Figure 3.1.1.

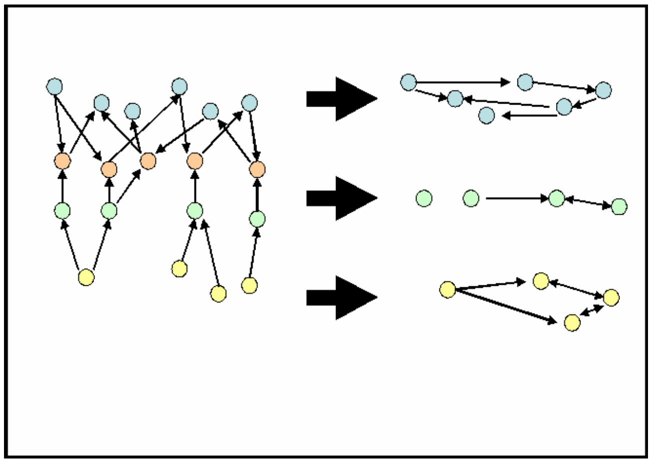

Figure 3.1.1: Transformation: The 4-Level-Model on the left side can be transformed to a single level as shown on the right according to the transformtaion rules.

3.1.2 Randomization

When we want to know whether a module is significantly frequent within the graph, we need to compare the frequency with those of random graphs. Randomization of the consisting graph returns such a random graph.

If we would only choose a random number of nodes and links, the results would not be comparable to our metabolic graph. The frequencies of the modules seem to be defined through the size of the graph and the average number of links per node.

One approach of a appropriate randomization is to keep the numbers of inlinks and outlinks per node and only change the destinations of these links. When the original graph has x nodes with i inlinks and o outlinks, the randomized graph will also have x such nodes. This is useful for significance testing in most cases of considered modules. As we have seen in chapter 2 some connections are bidirectional, but the links are not. Without special treatment the randomization destroys most of the bidirectional connections which leads to an overrepresentation of modules containing bidirectional connections.

If we want to test the significance of the connectivity itself we have to choose another randomization than the presented above because that conserves all connectivities. Anyway the randomized graph should keep the general properties of the original graph like the number of nodes, the number of bidirectional and unidirectional links.

3.2 Implementation

3.2.1 Transformation

The result of a transformation is a graph which nodes are connected if the nodes in the original graph are in a well defined relationship. A gene node A is connected with a directed link to a gene node B if there is a metabolite which is an educt of a reaction catalyzed by an enzyme A' and simultaneous the product of a reaction catalyzed by an enzyme B', where enzyme A' is produced by gene A and enzyme B' is produced by gene B.

1. First a new graph G' is created with the same number of nodes as gene nodes in the original graph G. Each original node is mapped to a new one in G' which is saved in an according map M.
2. Iteration over all gene nodes in G is started.
   a. For each enzyme of a gene all metabolite products of the according reactions are collected.
   b. Where this metabolite is an educt, all reactions and their enzymes are collected.
   c. Finally the gene nodes of these enzymes are collected.
   d. The gene nodes in G' that correspond to the resulting nodes are received from the map M.
   e. Directed links from the origin node to each destination node are inserted in G'.

3.2.2 Randomization

The randomization that conserves the connectivity tries to save also bidirectional connections. The two unidirected links are replaced by one bidirectional link. Now each node is characterized through a defined number of inlinks, outlinks and bilinks.

1. First a new graph with an equal node set but without any links is created. Each new node has a pendant in the original graph.
2. Three collections are created: “need bilink”, “need inlink” and “need outlink”.
3. Depending on their connectivity, the nodes are put in some of these collections.
4. A node is removed from a collection as soon it has as many links from the collection type as its counterpart in the original graph.
5. As long the collections contain some nodes, we choose two nodes from the collections and connect them with an according link. Either we take both nodes from the “need bilink”-collection or one from “need inlink” and one from “need outlink”. If we take the same node twice within such a step, we discard it because

we do not allow self references. If we always take the same node twice because the collections contain only one and the same node, we stop the algorithm.

Finally in the resulting randomized graph at most two links are missing which is negligible compared to the size of the graph.

The randomization that does not conserve the connectivity is much simpler

1. First a new graph with an equal node set but without any links is created. Each new node has a pendant in the original graph.
2. The same number of unidirectional and bidirectional links are chosen as in the original graph.
3. These links are randomly distributed to the nodes.

The result is a graph which nodes have random characterizations while keeping the graphs general properties as number of nodes or number of a sort of links.

3.2.3 Search

We are interested in the structure of the transformed graph. Therefore we try to find typical patterns. Additional to the modules found in [5] we analyze the graph for hubs, paths and cliques, which seem to be a important part of the graph. We are looking for the statistical significance with respect to the occurrence of different substructures of the graph. Each search target (module, path, hub, clique) requires a special search algorithm. Independent from the target we are interested for the same information of this target. We want to know the frequency of a target and the frequency within a randomized graph. We want to know the involved nodes of a target and its specific structure (i.e. there are more than 10 variations of 3-node-modules and we are interested in the organization and frequency of every one).
All algorithms have similar outputs and equal tasks to fulfill which is expressed in the interface of abstract search classes.

Each search happens in 3 steps as shown in figure 3.2.3.1. In the first step we choose the graph where the search should run. This can be either a graph from a file or a randomized graph. The second step is to choose the search itself. There are searches for modules, hubs, paths and cliques. In the third step each found target has to be processed in one of many ways. Targets can be counted, listed or identified.

These three steps are represented in three objects. The first object is the main loop, where initialization, running and evaluation of the chosen search algorithm happens. Randomizations can be applied to the given graph and the search can run on such a randomized graph. The second object is the implementation of the search algorithm. Each search algorithm has methods for initialization, running and evaluation. The third object is the target processor which is applied for each item found by the search algorithm. A target processor has methods for initialization, target processing and evaluation.

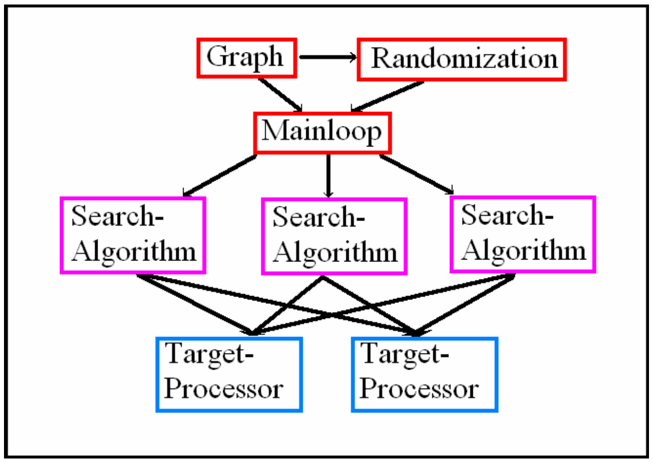

Figure 3.2.3.1: Structure of the search. Different algorithms can use the same target processor.

Figure 3.2.3.2: Pseudo source code of the code.

```
class MainLoop
{
public:
     MainLoop( SearchAlgorithm, int nRandom=0, int
RandomType=RT1 );
};
class SearchAlgorithm
{
public:
     SearchAlgorithm( TargetProcessor );
     void begin();
     void search( Graph* );
     void end();
};
class TargetProcessor
{
public:
     TargetProcessor();
     void begin();
     void process( NodeList* );
     void end();
};
```

3.2.3.1 Hubs

Searching for hubs is very simple, since a hub is defined by its links. We choose a node and collect all other nodes that are directly connected to it. The ID of the hub depends on the number of connected nodes.

3.2.3.2 Paths

A path is a chain of nodes. Each node that is part of the path has exactly two neighbors. At the beginning of the path search, all nodes are marked as unvisited. We choose an unvisited node and mark it as visited. If this node has exactly two neighbors, it is the beginning of a new path. We traverse along both of the two nodes as long the traversed nodes have exactly two neighbors. We mark each traversed node as visited and add it to the created path. When the traversal is finished, we pass the collected nodes in the path to the target processor. Afterwards we search for the next path by choosing another unvisited node.

3.2.3.3 Cliques

We choose a node and we choose the largest possible size N of a clique. Initially this is equal to the number of connected nodes. Then we choose N nodes within the connected ones. For each of these nodes we test whether it is connected to all other chosen nodes. If this test succeed we found a clique and we call the Target Processor. If one such connection is missing, we choose another set of nodes. If we have tested all sets, we decrease the maximum clique size to N-1.

3.2.3.4 Modules

The idea of the module search algorithm is to return all available modules of a certain size N fixed at initialization. A module of size N consists of N nodes that form a connected graph. Each found module get identified by a first target processor according to its structure. Two modules with the same structure get the same ID. The idea of the module search algorithm is choose a node, process all modules that contain this node, mark this node as visited and go on with the next node.

1. All nodes are marked as unvisited and left-nodes-in-module is set to N.
2. The first unvisited node is marked as in-use and is added to a part-of-module-collection. Since this node is now certainly within the module, left-nodes-in-module is decreased to left-nodes-in-module - 1.
3. All unvisited neighbors of nodes in the part-of-module-collection which are not in-use are marked as in-use and collected in the neighborhood-collection.
4. From the neighborhood-collection between 1 and N nodes are chosen. The other nodes are deleted from the neighborhood–collection. The chosen nodes are added to the part-of-module-collection.
5. If the size of the part-of-module-collection is N, the algorithm returns a found module, else step 3 is called again

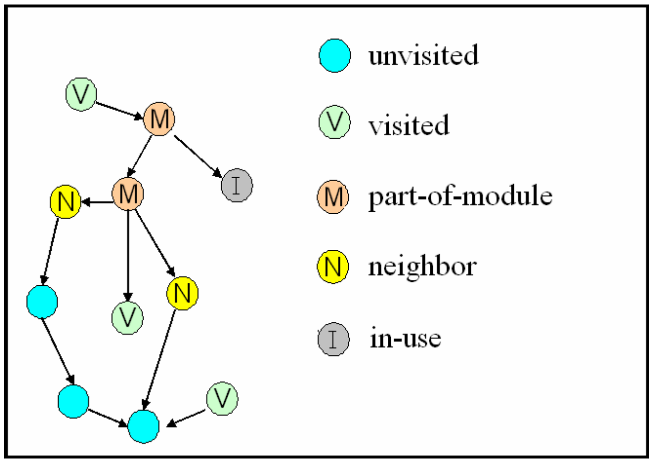

Figure 3.2.3.4.1: The module search algorithm: All nodes in neighborhood to part-of-module-nodes are visited, in-use or get part of the neighbor-collection.

The module search algorithm produces all existing modules within the graph. For further processing we need to identify each module and assign a unique ID to it. Comparing such two modules is very expensive in calculation time. Instead of full comparison of the graph we first pre-classify it. This pre-class is defined by the number of nodes and the number of links within the module. We sort the nodes of the module, where the sorting criteria are the number of inlinks and outlinks of a node. Then the number of inlinks and outlinks of the nodes are written to a string. This defines the so called header-string. Two equal modules have the same header-string. If two modules have different header strings they are not equal. Else we do not know it and are forced to compare the modules.

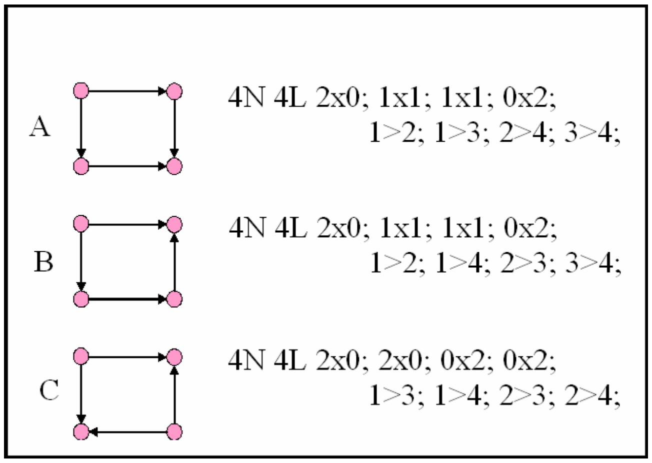
A
4N 4L 2x0; 1x1; 1x1; 0x2;
1>2; 1>3; 2>4; 3>4;
B
4N 4L 2x0; 1x1; 1x1; 0x2;
1>2; 1>4; 2>3; 3>4;
C
4N 4L 2x0; 2x0; 0x2; 0x2;
1>3; 1>4; 2>3; 2>4;

Figure 3.2.3.4.2: Two modules (A and B) contain an identical beginning of the header string.

3.3 Results

3.3.1 Hubs

P-value	Count	Average	Hub-Type
0.010000	43	33.820000	hub with 0 link
0.910000	70	80.199997	hub with 1 link
0.000000	233	106.919998	hub with 2 links
1.000000	53	103.080002	hub with 3 links
1.000000	35	82.849998	hub with 4 links
1.000000	16	54.910000	hub with 5 links
1.000000	18	32.480000	hub with 6 links
1.000000	8	16.219999	hub with 7 links
0.000000	16	7.770000	hub with 8 links
0.070000	7	3.550000	hub with 9 links
0.040000	4	1.320000	hub with 10 links
0.100000	2	0.540000	hub with 11 links
0.000000	4	0.230000	hub with 12 links
0.040000	1	0.050000	hub with 13 links
0.000000	3	0.030000	hub with 14 links
0.000000	3	0.020000	hub with 15 links
0.000000	3	0.000000	hub with 16 links
0.010000	1	0.010000	hub with 17 links
0.000000	2	0.000000	hub with 21 links
0.000000	1	0.000000	hub with 23 links
0.000000	1	0.000000	hub with 28 links

Table 3.3.1.1: P-Value: Significance of a module. Count: The occurrence of the module in the original graph. Average: The average occurrence of the module in the randomized graphs. Hub-Type: size of the hub.

The network seems to consist of large hubs which are significant even they are not so frequent. Hubs without any links are significant because there is a lot of reaction data missing which leads to many nodes without any neighbor.
The randomization produces graphs which have nodes containing two or three links in average. As expected, the frequency of random hubs decreases with their size. Significant hubs contain null or two links or more than ten. Hubs with three to nine links are not significant.

3.3.2 Paths

P-value	Count	Average	Path-Type
1.000000	112	165.820007	path with 1 node
1.000000	23	47.299999	path with 2 nodes
0.870000	15	17.959999	path with 3 nodes
0.170000	9	6.480000	path with 4 nodes
0.000000	8	2.310000	path with 5 nodes
0.000000	6	0.880000	path with 6 nodes
0.220000	1	0.250000	path with 7 nodes
0.000000	3	0.040000	path with 9 nodes
0.010000	1	0.010000	path with 11 nodes

Table 3.3.1.1: P-Value: Significance of a module. Count: The occurrence of the module in the original graph. Average: The average occurrence of the module in the randomized graphs. Path-Type: size of the path.

Although the metabolic network consists of pathways, the short paths are very frequent but not significant. Paths with a length bigger than four nodes tend to be significant but not very frequent. Paths of a certain length do not include paths with a larger length; each one is counted just once.

3.3.3 Cliques

P-value	Count	Average	Clique-Type
0.000000	1050	113.820000	clique with 3 nodes
0.000000	1144	44.939999	clique with 4 nodes
0.000000	1035	4.730000	clique with 5 nodes
0.000000	570	0.060000	clique with 6 nodes
0.000000	161	0.000000	clique with 7 nodes
0.000000	16	0.000000	clique with 8 nodes

Table 3.3.4.1: P-Value: Significance of a module. Count: The occurrence of the module in the original graph. Average: The average occurrence of the module in the randomized graphs. Clique-Type: Size of the clique.

All found cliques are significant and also frequent. Interesting is that there are more cliques of size 4 than cliques of size 3, even also subcliques are counted. This is because of the overlapping of some cliques. For instance two cliques of size 4 can share 3 nodes which lead to fewer cliques of size 3 than one might expect.
The huge cliques of sizes up to 8 are probably a result of reactands like ATP which are part of many reactions. Removal of such general metabolites could clarify the results.

3.3.4 Modules

P-value	**Count**	**Average**
0.000000	25	0.930000
0.000000	111	0.950000
0.000000	53	2.060000
0.000000	97	2.040000
0.000000	54	0.360000
0.000000	6	0.640000
0.060000	4	1.740000
0.960000	134	196.520004
0.980000	684	749.789978
1.000000	129	208.419998
1.000000	382	478.540009
1.000000	236	362.799988
1.000000	210	358.420013

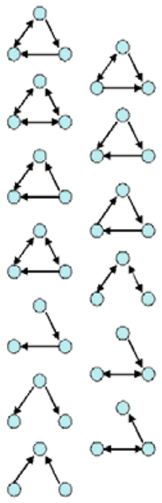

Table 3.3.4.1: P-Value: Significance of a module. Count: The occurrence of the module in the original graph. Average: The average occurrence of the module in the randomized graphs. Graph-String: A string representation of the module.

All significant modules with 3 nodes are fully connected. In the randomized graphs they occur very rarely. This can be explained by the destruction of this full connectivity. Otherwise all modules that are frequent in the randomized graph are neither fully connected nor significant in the original graph.

When looking at module with 4 nodes we see that they consist mostly on a 3-nodes-module with full connectivity which explains the significance. The fourth node is often connected to just one other node. There are just a few modules with 4 nodes that have a higher complexity.

It is an open question whether there is a clear interpretation of the overrepresented modules which is more specific than the high-connectivity-explanation.

P-value	**Count**	**Average**	**Graph-String**
0.000000	24	1.630000	4n; 5l; 1*2; 1*2; 2*1; 1*0; 0>3; 0>2; 1>0; 1>2; 2>1;
0.000000	203	4.920000	4n; 5l; 1*3; 1*2; 2*0; 1*0; 0>3; 0>2; 0>1; 1>0; 1>2;
0.000000	187	17.570000	4n; 8l; 3*3; 2*2; 2*2; 1*1; 0>3; 0>2; 0>1; 1>2; 1>0; 2>0; 2>1; 3>0;
0.000000	295	5.470000	4n; 6l; 2*3; 1*2; 1*1; 2*0; 0>2; 0>1; 0>3; 1>0; 1>3; 2>0;
0.000000	32	0.710000	4n; 6l; 2*2; 1*2; 2*1; 1*1; 0>3; 0>2; 1>0; 1>2; 2>1; 3>0;
0.000000	108	1.630000	4n; 5l; 2*2; 0*2; 1*1; 2*0; 0>2; 0>3; 1>0; 1>3; 2>0;
0.000000	142	1.590000	4n; 6l; 3*2; 2*1; 0*2; 1*1; 0>3; 0>1; 1>0; 2>0; 2>1; 3>0;
0.000000	161	16.219999	4n; 7l; 2*3; 2*2; 2*2; 1*0; 0>3; 0>2; 0>1; 1>0; 1>2; 2>0; 2>1;
...			
0.130000	2	0.650000	4n; 4l; 1*1; 1*1; 1*1; 1*1; 0>1; 1>3; 2>0; 3>2;
0.140000	3	1.230000	4n; 5l; 1*2; 2*1; 1*1; 1*1; 0>1; 0>2; 1>0; 2>3; 3>1;
0.160000	9	4.630000	4n; 6l; 2*3; 1*2; 2*1; 1*0; 0>2; 0>1; 0>3; 1>2; 1>0; 2>0;
0.160000	1	0.280000	4n; 6l; 1*3; 2*1; 2*1; 1*1; 0>2; 0>1; 0>3; 1>2; 2>1; 3>0;
0.250000	1	1.170000	4n; 5l; 0*3; 2*1; 2*1; 1*0; 0>2; 0>1; 0>3; 1>2; 2>1;
0.250000	8	5.170000	4n; 7l; 3*3; 1*2; 2*1; 1*1; 0>2; 0>1; 0>3; 1>2; 1>0; 2>0; 3>0;
...			
1.000000	127	251.910004	4n; 4l; 2*1; 1*1; 1*1; 0*1; 0>2; 1>0; 2>0; 3>1;
1.000000	820	1182.380005	4n; 3l; 0*2; 1*1; 1*0; 1*0; 0>1; 0>3; 1>2;
1.000000	471	1007.609985	4n; 3l; 0*2; 2*0; 0*1; 1*0; 0>1; 0>3; 2>1;
1.000000	49	314.529999	4n; 4l; 2*1; 0*2; 1*1; 1*0; 0>2; 1>0; 1>3; 2>0;
1.000000	37	125.900002	4n; 4l; 2*1; 2*1; 0*1; 0*1; 0>1; 1>0; 2>0; 3>1;
1.000000	149	401.179993	4n; 6l; 2*2; 2*2; 1*1; 1*1; 0>2; 0>1; 1>3; 1>0; 2>0; 3>1;
1.000000	122	472.609985	4n; 5l; 2*2; 2*1; 1*1; 0*1; 0>2; 0>1; 1>0; 2>0; 3>1;

Table 3.3.4.2: Some modules consisting of 4 nodes. P-Value: Significance of a module. Count: The occurrence of the module in the original graph. Average: The average occurrence of the module in the randomized graphs. Graph-String: A string representation of the module.

3.4 Discussion

We saw that in the network there is a significant representation of big hubs, long paths, cliques and high connected modules. Some of these results might be caused by high connected metabolites as ATP. Removing such metabolites could give us clearer insights in the network.

The randomization leads to a more uniform distribution of the links. Therefore more complex substructures like cliques, long paths and large hubs get destroyed. We do not found an intuitive interpretation for the significance of the resulting substructures except that they have to be characteristic for this metabolic network.

In a different context it could be interesting to search for substructures within another representation of the metabolic network than the described one. For example we like to find substructures within a model where a node stands for a metabolite and a link between two metabolites means that they belong to the same reaction.

4. Coexpression

After identifying characteristic substructures within the metabolite network we are interested how far their occurrences correspond to patterns in the gene expression data.

4.1 Methods

Since each node represents an enzyme, we have to transform the substructure to the according genes which are related to the gene expression data. This transformation is very simple: For each gene connected to the first enzyme node, we introduce a new copy of the substructure containing this gene instead of the enzyme. This is applied recursively to all nodes. The number of resulting transformed substructure is the product of the number of genes per enzyme node.

The gene expression data consist of vectors with 154 entries per gene. First we copy the vectors for the genes in the substructures into a matrix. To reduce the amount of data we calculate the correlation matrix and extract the eigenvalues. We receive as much eigenvalues as there are nodes in the considered substructure. The ratio between the first eigenvalue and the sum of all represents the proportion of coexpression accounted for by the first principal component which is an indicator for the coexpression level. The proportion of coexpression values of a certain substructure type are collected. Together they form a density curve which simplifies interpretation.

4.2 Implementation

The implementation is done in R [8], a freeware statistic tool available for windows and linux. Due to the low complexity of the task, the code is very straightforward. In contrast to C/C++, R has a weak OOP-support. Classes exist, but they have a noticeable overhead expressed in producing slowness. Therefore we used less sophisticated data structures as lists and arrays for processing data. To reduce the code written in R we used file formats which are already in appropriate forms for further computations.

We need two different data structures for calculating the coexpression. The first contains the gene expression data from the DNA-chip. The second contains the genes of the substructures and some additional information like number of nodes or type of the substructure. Also the resulting proportion of coexpression is saved in the second data structure. After loading the appropriate files all different substructures are processed by one single function that calculates the proportion of coexpression. To present the results we use the plot and density functions of R.

4.2 Results

4.2.1 Links

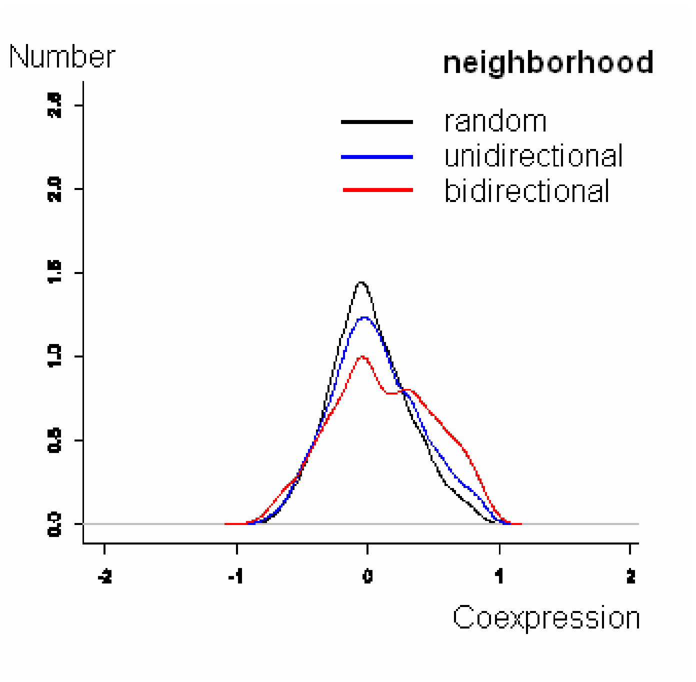

Figure 4.2.1.1: Density curve of proportion of coexpression. The coexpression variance of unidirectional connected nodes is significant higher than random chosen nodes. Bidirectional connected nodes have even a significant higher coexpression variance.

Bidirectional links lead to a significant greater diversion in coexpression between the nodes as unidirectional links, whereas unidirectional linked nodes have a significant higher coexpression as not connected nodes.

4.2.2 Modules

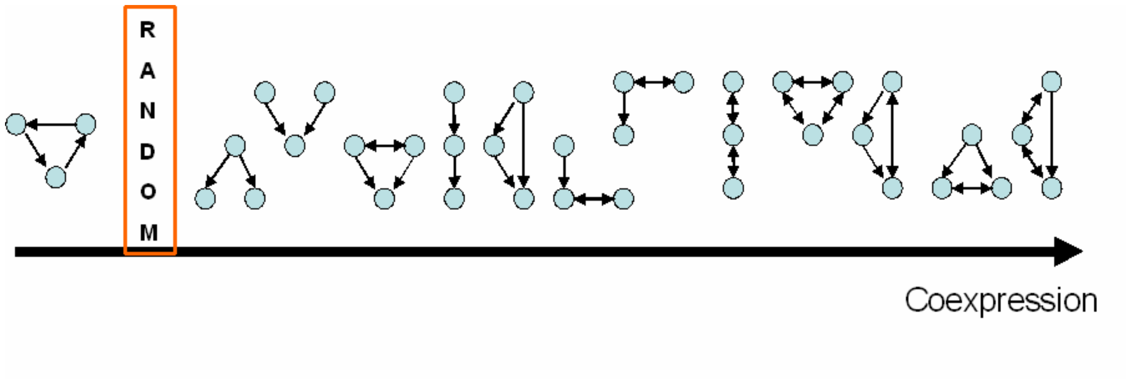

Figure 4.2.2: Modules with 3 nodes ordered by their average proportion of coexpression

Even the average coexpression value leads to a certain order of the 3-nodes-modules, it seems not possible to find a rule for this distribution. We can see a tendency that coexpression grows with higher connectivity of the modules.

4.2.3 Hubs

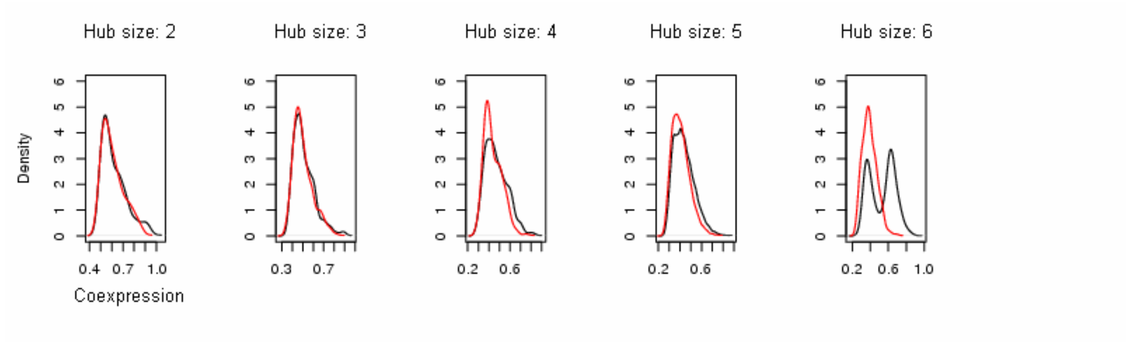

Figure 4.2.3.1: Coexpression density curve. The red line is the coexpression distribution of random chosen nodes; the black line is the coexpression distribution of nodes in a hub.

Most interesting in the distribution are the two peaks in the diagram on the right side, even it is not clear how these peaks could be interpreted.
Although the curves look very similar, the hubs have a significant coexpression.

4.2.4 Cliques

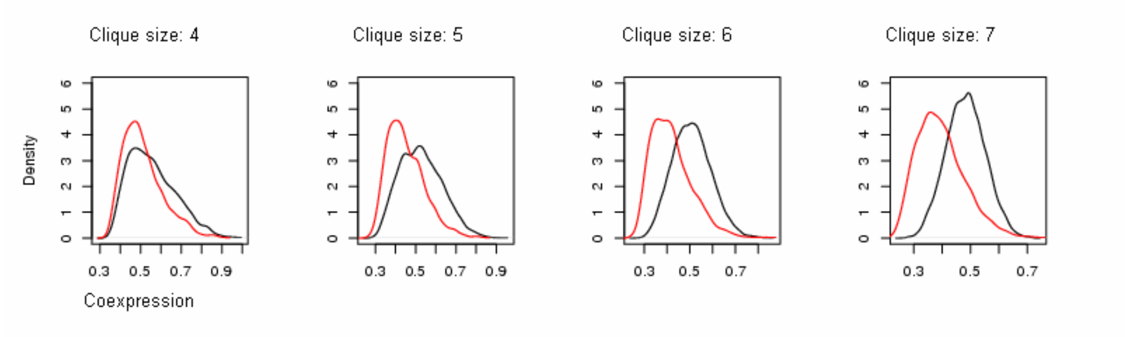

Figure 4.2.4.1: Coexpression density curve. The red line is the coexpression distribution of random chosen nodes; the black line is the coexpression distribution of nodes in a clique.

The distribution of the clique coexpression shows a clear difference to the random distribution. One might expect that the coexpression decreases with the size of the clique because of redundancy or complexity of a huge structure. As we saw in chapter 3, there are only few large cliques which perhaps depend on a common metabolite as ATP.
Removing such metabolites could change the coexpression pattern significantly.

4.3 Discussion

The results suggest that the level of coexpression of the gene expression data depends on the connectivity of the according substructure. The genes of substructures with a high links per node ratio are significant more coexpressed than such with a low ratio.
Additional reaction data and removing general metabolites that are part of various reactions could strongly affect the results.

5. Summary and future work

The chosen model was useful for all applications in this work. Reaction information is missing in the pathway database, and even we used KEGG to complete the data, we did not find some reaction information. An improved database would help to get more exact results from analysis and coexpression searching.

The found substructures like modules, paths, hubs and cliques give us an insight into the construction of the metabolic network, but for a useful interpretation we would need a deeper analysis. For example we can search substructures in the metabolic level of the network, where nodes represent metabolites and links between them represent genes respective enzymes. Coexpression patterns are related to the links instead of nodes, which perhaps will be expressed in other significant substructures. This allows another interpretation of the gene expression data.
Also the chosen substructures are just a subset of various possibilities. Searching for other substructures could reveal new network properties. Examples are structures where each node has exactly 3 neighbors (instead of paths, where each node has exactly 2 neighbors), or substructures containing exactly n links (instead of modules, which consist of n nodes).

The coding of the model and the search algorithms lead to a clear OOP-structure which should be easy to extend for various similar applications. For minimizing calculation time the classes would have to be broken up since the code is optimized for nice design and not for speed.

The coexpression patterns of the substructures show that the number of links per nodes in a substructure enlarges the coexpression level significantly. Even this is a nice result; it's possible that there are more complex and more interesting relationships between gene expression and substructures. Instead of finding such relations by hand, an evolutionary algorithm could return a function that calculate possible substructures for a given coexpression pattern.

Frequency as well as coexpression pattern of the substructure could change significantly if we remove often used metabolites as ATP from the network.

The title of this diploma thesis is “gene expression profiling and pathway scoring”. In contrast to that we did not found a scoring function. The proportion of coexpression in the expression patterns reveals not enough information for generating such a function.
Nevertheless we found interesting network properties in form of significant substructures and we saw that there is at least a very coexpression pattern within gene expression data.

References

[1] Hanisch D, Zien A, Zimmer R, Lengauer T. Co-clustering of biological networks and gene expression data.
Bioinformatics. 2002 Jul;18 Suppl 1:S145-S154.

[2] Ku_ner R, Zimmer R, Lengauer T. Pathway analysis in metabolic databases via di_erential metabolic display (DMD).
Bioinformatics. 2000 Sep;16(9):825-36.

[3] Ma H, Zeng AP. Reconstruction of metabolic networks from genome data and analysis of their global structure for various organisms.
Bioinformatics. 2003 Jan 22;19(2):270-7.

[4] Milo R, Shen-Orr S, Itzkovitz S, Kashtan N, Chklovskii D, Alon U. Network motifs: simple building blocks of complex networks.
Science. 2002 Oct 25;298(5594):824-7.

[5] Shen-Orr SS, Milo R, Mangan S, Alon U. Network motifs in the transcriptional regulation network of Escherichia coli.
Nat Genet. 2002 May;31(1):64-8.

[6] Ihmels J., Levy R., Barkai N. Principles of transcriptional control in the metabolic network of Saccharomyces cerevisiae.
Nat. Biotech. 2004 Jan, 22: 86-92

[7] Kyoto Encyclopedia of Genes and Genomes.
www.genome.ad.jp/kegg

[8] The R Project for Statistical Computing
www.r-project.org