The DM Algorithm: A Causal Search Algorithm for the Discovery of MIMIC Models, with an Attempt to Recover a Protein Signalling Network from a High-Dimensional Ovarian Cancer Dataset

Alexander Murray-Watters

August 21, 2014

Abstract

Latent variables have long confounded attempts to determine causal structure when experiments cannot be conducted. While some methods exist for dealing with exogenous latent variables, endogenous latents remain neglected. This thesis presents a new algorithm (the DM algorithm) designed to discover causal structure for a restricted class of models when endogenous latents are present. The algorithm is non-parametric, and in simulations outperformed one of the most popular methods for handling endogenous latents (namely, factor analysis). As the DM algorithm is also capable of handling a surprising number of variables, the algorithm was run on a high-dimensional genomic dataset. Popular methods in genomics lack the ability to address large numbers of variables and provide less information about the latent structure than the DM algorithm, so this represents an improvement on the state-of-the-art.

Acknowledgements

I'd like to begin by thanking my parents and twin for their tolerance of all my annoying questions over the years, as well as their willingness to listen while I worked through problems out loud.

Many thanks to Clark Glymour for allowing me to research a problem on which little progress might be made and for preventing me from falling down the various crevices that appear during research.

Many thanks also to Richard Scheines for his time and very helpful suggestions(such as on factor analysis), despite his new administrative burdens.

Needless to say, I'm also very grateful to both Clark and Richard for their willingness to be my advisors.

I'm grateful to Xinghua Lu for both providing the ovarian cancer dataset used in the data analysis and for kindly taking the time to answer questions about its contents.

I'd also like to thank Joe Ramsey for providing a great deal of assistance in dealing with some computational difficulties.

Finally, I'd like to express apprection of my fellow graduate students for all of the interesting and informative discussions we've had over the past year, as well as their support during stressful periods.

Contents

List of Figures

List of Algorithms

List of Source Code Files

Chapter 1

Introduction

One of the enduring features of science is the discovery of novel variables and their causal relations with each other and with known variables. The histories of physics, chemistry and biology all illustrate such discoveries, from novel fundamental particles, to atoms and their weights, to genes and the processes through which they produce proteins. As these examples illustrate, the novel variables are often unmeasured and unmeasurable at the time of their discovery. Similar issues of discovery arise in psychology, economics, and the many social sciences where the means of discovery, such as they are, have been primarily statistical, as in psychometrics.

In many of these domains an important research question concerns the identification of unmeasured variables that are causal intermediaries between measured variables, the selection of clusters of input and output variables that share a common intermediate, and the causal relations among the unmeasured intermediates. Causal hypotheses of these forms are often referred to as "MIMIC" models, short for Multiple Indicators Multiple Input Causes.

Several statistical techniques have been proposed or applied for the purpose

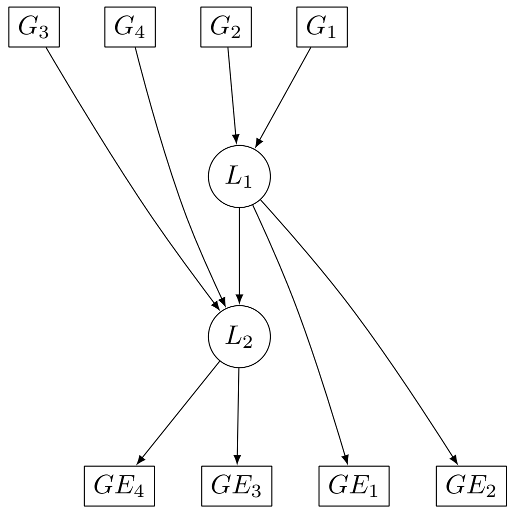


Figure 1.1: The structure of a MIMIC Model. $G_1 - G_4$ and $GE_1 - GE_4$ are observed variables, while $L_1 - L_2$ are unobserved variables.

of finding MIMIC models, each with its special limitations. The aim of this thesis is to describe, illustrate and prove correct (under restrictive assumptions) a new, non-parametric method for identifying a sub-class of MIMIC models. The thesis introduces prior methods used to discover hypothesized MIMIC models in data and explores their limitations. It then introduces the necessary conceptual and theoretical ideas, before discussing the new algorithm. This discussion includes both a worked example of the algorithm in practice, as well as a proof of correctness. Simulations are then discussed, comparing the performance of the new algorithm with that of factor analysis. A data analysis using the algorithm is then conducted on a genomic dataset and the results summarized. Finally, future avenues for research are discussed.

The very idea of algorithmic discovery of theories, or "models" has been challenged in the philosophical and statistical literatures. Notably, Carl Hempel, among the most eminent philosophers of science of the 20th century, denied the possibility on the very grounds that no algorithm could correctly discover unmeasured, novel properties(Hempel, 1985). Developments in the late 20th century and this century show that Hempel's opinion should be clustered with Kant's endorsement of phlogiston in the first edition of his *Critique of Pure Reason*, and with Hegel's (possibly apocryphal) announcement that there are necessarily seven planets (Craig and Hoskin, 1992). Improvements in automated causal inference now appear almost monthly, and all manner of problems (cyclic feedback structures, latent variables, latent variables and feedback, non-linear systems, and non Gaussian variables) that were once thought insuperable have been solved.

1.1 MIMIC Models in Practice

A hypothesized MIMIC model consists of three parts: a set of unobserved (latent) variables, a set of observed effects of the latents (or "outputs"), and a set of observed causes of the latents (or "inputs").

The ability to discover a MIMIC structure is useful in a number of situations. In genomics, some researchers are interested in discovering how the effect of a genetic mutation propagates through the protein signalling network, resulting in (possibly) different observed frequencies of various proteins. Genes are read and transcribed into mRNA, which is then translated into various proteins. Genes influence both which, and in what quantity, proteins are made. A genetic mutation (say, one that causes cancer) can lead to differences in the resulting levels of protein species, protein combination, and folding configurations. How the effects of mutation propagate through the network is of fundamental biological importance.

In neuroscience, researchers are interested in understanding how a signal (i.e., a stimulus), propagates through the brain's neural,network, and ultimately how these processes produce behavior. fMRI resolution is about two seconds

(Logothetis, 2008)[pg. 3], and since the signal propagates through the network at a significantly faster rate, it is currently not possible to directly observe the propagation of a signal through the brain. An important problem is to reveal the network structure from indirect imaging data clustered into "Regions of Interest," and MIMIC models, if discoverable, offer the possibility of identifying unmeasured intermediates between Regions of Interest.

In order to estimate the size of the shadow economy (the portion of the economy not captured by GDP or other government statistics), economists have made use of MIMIC models. Bühn and Schneider (2008) used MIMIC models to examine economic loss attributed to the shadow economy in France. Giles (1999) employed a MIMIC model to create a time-series view of the shadow economy in New Zealand. Tedds (1998) also estimated a MIMIC model in order to determine the size of the shadow economy in Canada. DellAnno and Schneider (2006) published an article advocating further use of MIMIC models in economics.

In Lester (2008), the researcher used a MIMIC model to determine what factors related to the successful settlement of immigrants to Australia. Subjects were non-labor force participants, as well as economic and non-economic immigrants. Indicators of successful settlement included mental health, belief that the decision to migrate was correct, encouraging others to migrate to Australia, as well as reported level of life satisfaction.

Ríos-Bedoya et al. (2009) used MIMIC models to examine the strength of association between two latent factors (pleasant/unpleasant early smoking experiences) and current smoking status.

1.2 Current Methods for Discovering MIMIC Models

The most common method of specifying a MIMIC model is simply to make one up and test it statistically. As is well-known (Mayo, 1996), there are typically a multitude of alternative models that can pass standard tests of fit for a data set. We have no reason for confidence that the tacit search, whatever it is, that an investigator goes through in proposing a model is a reliable procedure that adequately considers the alternative. For that kind of confidence, statistical search methods are needed that can be shown to be at least asymptotically correct under explicit assumptions or, failing that, will at least search a broad space of alternative models.

Previously developed methods used to discover a posited MIMIC structure are problematic in several ways. In the case of factor analysis, the method is unreliable (which is clearly shown in the simulations reported in chapter 6). In others, the method is computationally intractable for most interesting problems (Markowetz et al., 2007; Tresch and Markowetz, 2008), or makes highly restrictive assumptions(Brodie, 2014).

The methods proposed to find MIMIC models range from the recent (nested effect models), to the cutting-edge (sparse endogenous latent search), to adaptations of the procedure that inaugurated algorithmic search early in the last century (factor analysis).

1.2.1 Factor Analysis

Factor Analysis models data (**P**) by multiplying a principal component matrix (**w**) by the projections of the data onto the principle components (**F**). Any residual differences between the data and the result of the matrix multiplication are accounted for by an error term, ϵ. $\mathbf{X} = \mathbf{F}\mathbf{w} + \epsilon$

For example, imagine we have run factor analysis on a dataset (made up), and requested a model with 2 factors. We get back the following loading matrix:

	Factor1	Factor2
Var1	0.49	0.44
Var2	0.16	0.82
Var3	0.30	0.86
Var4	0.21	0.36
Var5	0.7	0.38
Var6	0.78	0.12

A "loading" is the correlation between a variable and a given factor. So Var1 and Factor1 have a correlation of .49. The matrix is then converted to a graph by choosing some cutoff loading (commonly .3), and drawing an undirected edge between a factor (latent) and a variable only if their loading is greater than the cutoff. So in the example, Factor1 has edges to Var1, Var3, Var5,and Var6, while Factor2 has edges to Var1, Var2, Var3, Var4, and Var5. Figure 1.2 depicts an example resulting from such a conversion.

Factor analysis is often used to perform dimension reduction, or in other cases, to infer causal structures when there are unobserved (latent) variables present.

When being used to infer a causal structure, proponents of factor analysis distinguish between two kinds of analysis: exploratory and confirmatory. In exploratory factor analysis, the number of latents to be used is not decided prior to looking at the data. Instead, the data is used to select the number of latents (often by looking at a Scree plot). Confirmatory factor analysis begins with a number of different models already specified (i.e., the number of latents is already specified for each model), and uses a chi-square goodness of fit statistic to select which of the models will be the final model. This distinction between

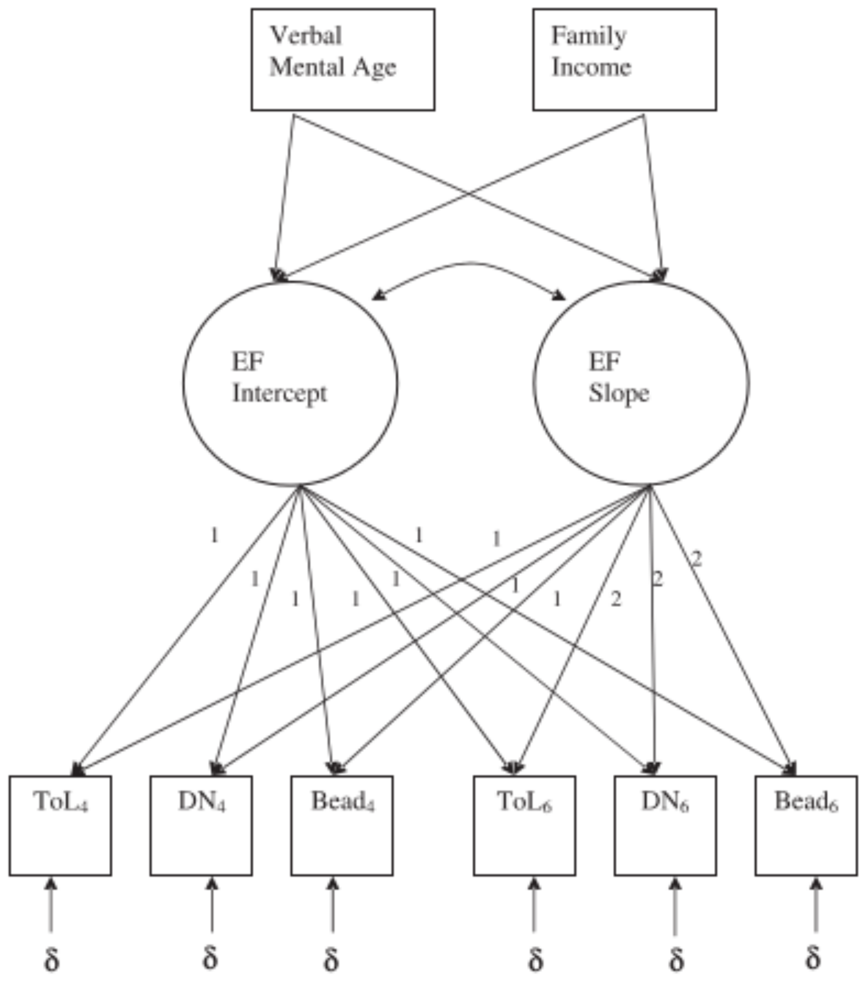


Figure 1.2: An example of a MIMIC model found using factor analysis, from Hughes et al, 2010. The model depicts two latents (related to executive function). The outputs are three different test scales acquired when subjects were ages 4 and 6.

types of factor analysis based on prior specification is odd; it suggests that if a researcher were to test enough models using confirmatory analysis, the only difference in justification between the confirmatory and the exploratory version would be that the researcher wrote a list of models earlier. Yet practitioners are using completely different model selection criteria, even though there is little substantive difference between the two "kinds" of analysis. If we accept the distinction between types of factor analysis, however, then the factor analysis being discussed in this thesis is exploratory, not confirmatory.

The history of factor analysis (and its use to infer a causal structure) began when Charles Spearman observed that a collection of variables, specifically children's grades in different subjects, had a correlation matrix which followed a pattern of constraints (known as tetrad constraints). Using this pattern, he claimed that there exists a common latent "factor," which he referred to as G, representing a person's intelligence. Spearman's pattern failed to hold in general, which led to modifications by his students. Spearman's method was intractable at the time (Glymour et al., 1987). Thurstone later modified the method so that it was both computationally tractable and capable of handling more than one factor (Thurstone, 1934).

Unfortunately, factor analysis is an unreliable tool for causal inference, in that the method cannot reliably cluster variables around latents. This is due to the tendency of factor analysis to report a different structure when F is right multiplied by an orthogonal matrix transpose(m), and w is left multiplied by the same orthogonal matrix (m). This distressing result is known as the rotation problem (Shalizi, 2012). Simulations have been run which illustrate just how unreliable factor analysis is (see chapter 6).

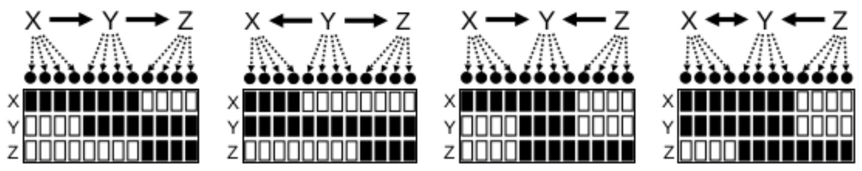


Figure 1.3: Several examples of NEM models, from Tresch and Markowetz (2008). Note that the black dots are observed effects. X, Y, and Z are observed inputs. The black rectangles in each 3 by 12 matrix denote input variable influence, with row corresponding to input variable and column denoting output variable. Also note that the underlying latent structure is not specified.

1.2.2 Nested Effect Models (NEM)

Nested effect models (e.g., Markowetz et al. (2007); Tresch and Markowetz (2008)) have been developed specifically for genomics. They rely on perturbing genes (i.e., performing an intervention on a gene), observing the resulting changes in gene expression level, comparing these changes to the unperturbed levels, and finally fitting a model on the data (generally fitting a Bayesian network via a Markov-chain Monte Carlo type algorithm). Nested effect models are used to cluster genes with phenotypes (effects) (Markowetz et al., 2007) and order gene clusters based on “subset relations between phenotypes” (ibid). This is not the same as a MIMIC model, as causal connections between the latent parts of the signalling network are not contained in the reported NEM, leaving the details of their construction unspecified.

Despite recent improvements (Tresch and Markowetz, 2008), the various methods based on nested effect models are still computationally intractable for high dimensional problems (which some parts of genomics face). Additionally, the fastest method for constructing a NEM (Tresch and Markowetz, 2008) is very sensitive to the addition or removal of an edge in the model, leading to a very unstable likelihood function.

1.2.3 SELS (Sparse Endogenous Latent Search)

Another method recently proposed for studying causally incomplete systems (i.e., where relevant variables are unobserved) is known as SELS, or Sparse Endogenous Latent Search. It begins by calculating the rank of a dataset's "sparse factorization." The extent to which the resulting matrix is rank deficient gives the number of latent variables. This information is used to orient edges in a partial ancestral graph, the details of which are unimportant for this discussion.

Unfortunately, SELS makes a number of strong assumptions, preventing its use in the MIMIC model case. Specifically, in the true graph (i.e., the actual underlying causal structure), there can be no direct causal relations between latents, the parents of latents, or the children of latents (Brodie, 2014)[pg. 22]. Additional parametric assumptions, namely that variables follow a (multivariate) Gaussian distribution and are linearly related to one another (Brodie, 2014)[pg. 22], further reduce the method's applicability.

1.2.4 Find One Factor Clusters (FOFC)

Silva et al. (2006) developed a method for finding subsets of measured variables that share a single common unmeasured cause, and for estimating the causal relations among latent causes of different clusters. The procedure has been improved by Kummerfeld et al. (2014).This procedure can be used to find the latent structure of MIMIC models, with several limitations, including:

1. Relations between measured and latent variables must be linear.

2. Which inputs affect which latents is not usually identified.

3. Some measured variables will be eliminated.

4. Retained measured variables cannot influence other measured variables.

5. Each latent must have at least three output variables as children, neither of which is the child of another latent variable.

The method has the advantage, however, that the true structure need not be singly connected.

Chapter 2

Causal Graphs

2.1 Background

In a causal graph, a directed edge is interpreted causally. The origin node of the edge is a cause of the terminating node (also known as a child of the origin node). The origin node is also referred to as the ancestor of the terminating node.

There are three kinds of basic structures used in causal graphs: Chains, forks, and colliders. Chains (Figure 2.1) consist of two or more nodes, with each node (except the last) possessing a child. Forks (Figure 2.3) consist of at least three nodes, with a common node causing all of the other nodes. Finally, a collider (Figure 2.2) consists of at least two nodes with a common child.

An undirected path connecting two variables (A and B) is a a series of (one or more) transitions between nodes, beginning with A and ending with B. A directed path is similar, except it only allows transitions to travel in the direction of the arrows.

Definition 2.1.1. (Causal Markov Condition) A node in a directed acyclic

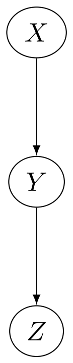


Figure 2.1: A chain.

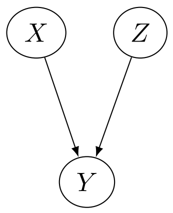


Figure 2.2: A collider.

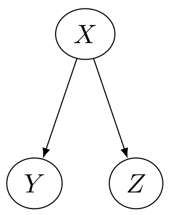


Figure 2.3: A fork.

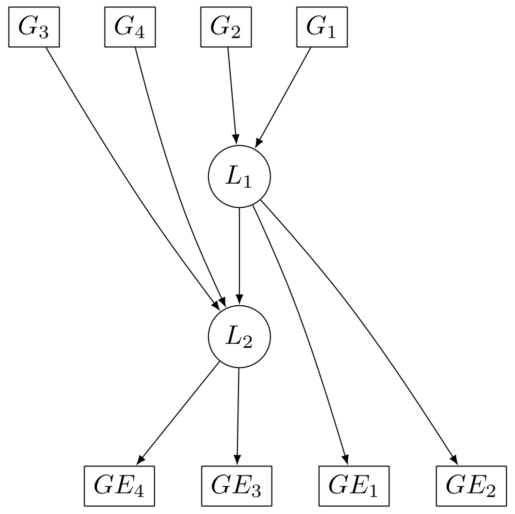


Figure 2.4: An example of a MIMIC model.

causal graph is independent of its non-descendants, given its parents.

Definition 2.1.2. (d-separation) Two nodes (X and Y) in a directed acyclic causal graph are d-separated (and therefore independent) by a set S, where X and Y are not elements of S, if and only if all paths connecting X and Y are blocked by S. If two nodes are not d-separated, then they are said to be d-connected (and are therefore dependent).

A path is blocked if the path contains an intermediate node which is neither a collider nor a descendant of a collider on the path, and said node has been conditioned on. A path is also blocked if there is a collider on the path (which hasn't been conditioned on) and no descendant of the collider has been conditioned on.

Finally, nodes displayed in a box are observed. In contrast, nodes in an oval or circle are unobserved, and are often referred to as "latents" or "factors".

To make both MIMC models and causal graphing language more concrete for the reader, here is a hypothetical example of a MIMIC model (Figure 2.4).

In the model, L_2 and L_2 are latents, as their nodes are represented by circles. G_1, G_2, G_3, G_4, GE_1, GE_2, GE_3, and GE_4 are all observed, as their nodes are represented by boxes.

As L_1 is a collider, and it blocks the only path between G_1 and G_2, G_1 is independent of G_2. As L_1 is a fork for GE_1 and GE_2, GE_1 and GE_2 are dependent. Similar relations hold for G_3, G_4, GE_3, and GE_4. The most interesting relation, however, is that while G_1 and G_2 are dependent with GE_1, GE_2, GE_3, and GE_4, G_3 and G_4 are only dependent with GE_3 and GE_4. This is due to L_2 acting as a collider, d-separating G_3 and G_4 from GE_1 and GE_2. It is also interesting to note that the independence of G_3 (or G_4) and GE_1 (or GE_2) disappears when GE_3 (or GE_4) is conditioned on, as doing so deactivates the collider at L_2, leaving G_3 and GE_1 d-connected.

Two other pieces of background information also need to be covered, as they are used in the algorithm discussed in the next chapter.

2.1.1 Sober's Criterion

Sober's criterion (Sober, 1998) is an empirical method for choosing between models which only cover cause and effect vs. ones which include latent variables. Figure 2.5 depicts two competing models which are representative of this problem. The criterion asks a simple question: Are the effects independent of one another when their causes are conditioned on? If the answer is yes, then the causes are not related to the effects via a latent variable. Otherwise, there is a latent present.

2.1.2 The PC Algorithm

The PC algorithm is a method for finding (given some assumptions) the set of causal graphs consistent with the data. It takes a dataset as input, and returns a

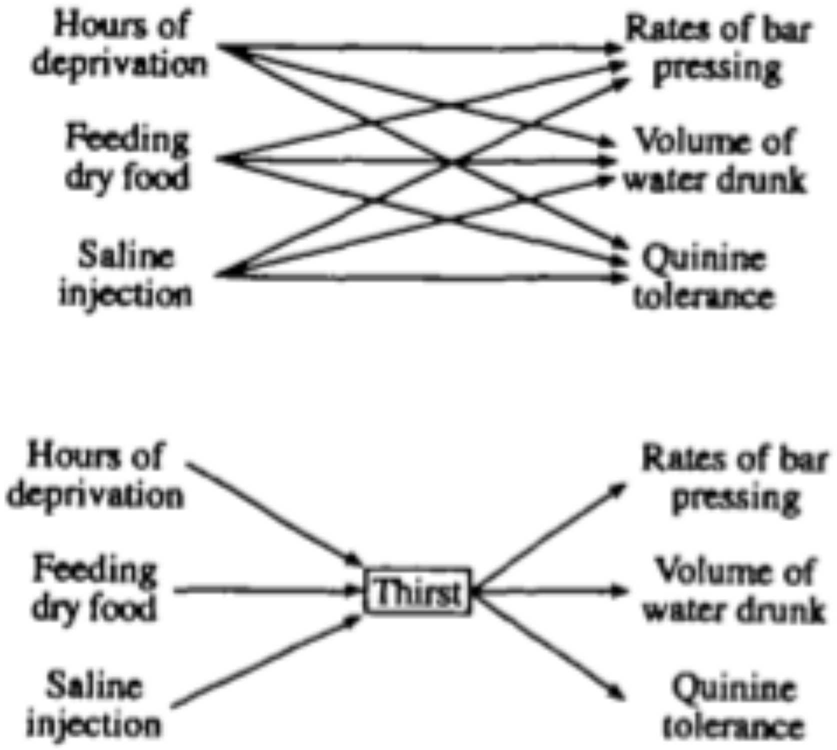


Figure 2.5: From Sober (1998). It depicts a "black box" model choice problem: Should the intermediate variable "thirst" be included or excluded from the model? Note that the bottom model is an example of a MIMIC model.

pattern consistent with the observed independence relations (i.e, an equivalence class of models). The pattern may or may not depict a unique causal graph, as a pattern can include undirected edges. The presence of an undirected edge in the pattern means that it (the edge's) direction is indeterminate, and one direction or the other (but not both) may be the truth. 1 is its pseudocode.

```
Data: Takes a dataset as input
Result: Returns a pattern.
A.) Form the complete undirected graph $C$ on the vertex set $V$.
B.)
$n = 0$.
repeat
    repeat
        select an ordered pair of variables $X$ and $Y$ that are adjacent in $C$
        such that $Adjacencies(C, X) \setminus Y$ has cardinality greater than or
        equal to n, and a subset $S$ of $Adjacencies(C, X) \setminus Y$ of cardinality
        $n$, and if $X$ and $Y$ are d-separated given $S$ delete edge $X - Y$
        from $C$ and record $S$ in $Sepset(X, Y)$ and $Sepset(Y, X)$;
    until All ordered pairs of adjacent variables $X$ and $Y$ such that
    $Adjacencies(C, X) \setminus Y$ has cardinality greater than or equal to $n$ and
    all subsets $S$ of $Adjacencies(C, X) \setminus Y$ of cardinality $n$ have been
    tested for d-separation;
    $n = n + 1$;
until For each ordered pair of adjacent vertices $X$, $Y$,
$Adjacencies(C, X) \setminus Y$ is of cardinality less than $n$;
C.) For each triple of vertices $X$, $Y$, $Z$ such that the pair $X$, $Y$ and the
pair $Y$, $Z$ are each adjacent in $C$ but the pair $X$, $Z$ are not adjacent in $C$,
orient $X - Y - Z$ as $X \rightarrow Y \leftarrow Z$ if and only if $Y$ is not in $Sepset(X, Z)$
D.) repeat
    If $A \rightarrow B$, $B$ and $C$ are adjacent, $A$ and $C$ are not adjacent, and there
    is no arrowhead at $B$, then orient $B - C$ as $B \rightarrow C$. If there is a
    directed path from $A$ to $B$, and an edge between $A$ and $B$, then
    orient $A - B$ as $A \rightarrow B$.
until no more edges can be oriented;
```

Algorithm 1: The pseudocode for the PC algorithm. (Spirtes et al., 2000)

The DM algorithm uses the PC algorithm for two tasks: classifying inputs and outputs using the edge direction found by PC, and determining which inputs are related to which outputs. In neither case is the output of PC used for its original purpose.

Having covered the basics of causal graphs, an example of a MIMIC model, and two other necessary pieces of background information, we can now move on to the algorithm itself.

Chapter 3

The DM (or detect.MIMIC) Algorithm

3.1 Definitions

Before delving into the algorithm, a number of definitions need to be made clear.
Inputs: Measured variables with only directed edges to (a) latent(s).
Outputs: Measured variables with a directed edge from a latent(s)
Singly connected: A graph is singly connected if there is at most a single undirected path connecting any pair of nodes.
Indegree: A variable's indegree is the number of directed edges pointing towards the variable,
Outdegree: A variable's outdegree is the number of directed edges pointing away from the variable.
Total Degree: The sum of a variable's indegree and outdegree.
Adjacency degree n: a function which returns the number of edges adjacent to node n.

Latent descendant: Given a latent (L1), A latent descendant is a latent variable caused by another latent variable.

3.2 Assumptions

Additionally, the algorithm currently requires a number of different assumptions.

Assumptions: The following properties are assumed to be true in the data generating process:

- A1: Markov Assumption: Every variable is independent of its non-descendants given the variable's parents.
- A2: Faithfulness: A graph and a probability distribution are faithful to one another if all the (un)conditional independence relations in the probability distribution are entailed by the graph and the Markov assumption.
- A3: The true graph is acyclic.
- A4: The true graph is singly connected.
- A5: Every latent has at least two inputs and two outputs.
- A6: No input has a path to an output except through a latent.
- A7: Inputs are probabilistically independent of one another.

  Note: Generalizations of the algorithm are possible without this assumption (A7), but the information recovered may be reduced.
- A8: Every measured variable is an input, an output, or a descendant of (an) output(s).

With those assumptions made, we can now move onto the algorithm itself. It consists of seven steps, and is followed by a worked example.

3.3 Pseudocode

Algorithm : **DM**(**Data**)

```
Data: Takes a Dataset as Input
Result: Returns a MIMIC model
$PC$ := A function returning the pattern produced by the PC algorithm.
inputs := $NULL$ The set of inputs.
outputs := $NULL$ The set of outputs.
$\mathbf{X}$ := Data
$pc.pattern := PC(\mathbf{X}, depth = 0)$
$\mathbf{N} := Nodes(pcpattern)$
for each $n$ in $\mathbf{N}$ do
    if $adjacency(n) \neq 0$ then
        if $adjacency(n) = outdegree(n)$ then
            add $n$ to inputs
        end
        else
            add $n$ to outputs
        end
    end
end
Input.Parents($n$) := $PAR(n, pc.pattern) \cap$ inputs
```

Algorithm 2: Step 1 of the DM algorithm

```
$\mathbf{Latents} := NULL$
$\mathbf{Latents}(L) :=< \mathbf{IN}(L), \mathbf{OUT}(L), \mathbf{LC}(L) >$
for all $L$ do
    $\mathbf{Latents}(L) :=< NULL, NULL, NULL >$
end
$\mathbf{Input}.\mathbf{Parents}$ : The set of cluster assignments. Each member of
Latents (i.e., a specific latent) contains < IN = set of inputs for the
latent, OUT = set of outputs, and LC = set of latent children (i.e., a
latent descendant). > for all $x$ in outputs do
    if there exists a $y$ in latents such that $\mathbf{Input}.\mathbf{Parents}(x) = \mathbf{IN}(y)$
    then
        $\mathbf{OUT}(y) := \mathbf{OUT}(y) \cup x$
    end
    else
        Create a new member, $z$, of Latents, with
        $\mathbf{Latents}(z) :=< \mathbf{IN}(z) := \mathbf{Input}.\mathbf{Parents}(x), \mathbf{OUT}(x) \cup x,$
        $NULL >$
    end
end
```

Algorithm 3: Step 2 of the DM algorithm

```
for each $x$, $y$ in Latents do
    if $\mathbf{IN}(x)$ is a proper subset of $\mathbf{IN}(y)$, and $\mathbf{IN}(x)$ is the largest such
    subset then
        $\mathbf{LC}(x) := \mathbf{LC}(x) \cup y$;
        for all $z$ in Latents do
            $\mathbf{IN}(z) := \mathbf{IN}(z) \setminus \mathbf{IN}(x)$
        end
    end
end
```

Algorithm 4: Step 3 of the DM algorithm

```
for each $x$, $y$ in Latents do
    if $\mathbf{LC}(x) = y$ and $\mathbf{OUT}(x) \perp\!\!\!\perp \mathbf{OUT}(y) \vee (\mathbf{IN}(x)$ and $\mathbf{IN}(y))$ then
        $\mathbf{LC}(x) := NULL$
        Let $z$ be the smallest subset of $\mathbf{IN}(x) \cup \mathbf{IN}(y)$ such that
        $\mathbf{OUT}(x) \perp\!\!\!\perp \mathbf{OUT}(y) \vee (z)$
        $\mathbf{IN}(x) := \mathbf{IN}(x) \cup z$
        $\mathbf{IN}(y) := \mathbf{IN}(y) \cup z$
    end
end
```

Algorithm 5: Step 4 of the DM algorithm

```
Step 5. $pc.pattern.infinite := PC(\mathbf{X}, depth = infinite)$
Step 6. Examine the graphs produced in steps 4 and 5 (name these $G4$
and $G5$, respectively).
for each output variable $O_i$ in $G4$ such that there is no direct edge
between $O_i$ and any input variables in $G5$ do
    remove the edge between $O_i$ and its latent.
    Add any adjacencies (from G5) between $O_i$ and the outputs
    connected to $O_i$'s former latent
end
Step 7. Return the graph from the end of step 6.
```

Algorithm 6: Steps 5, 6, and 7 of the DM algorithm

3.4 Worked Example

Having described the algorithm (pages 28, 29, 29, 29, and 30), we can now move on to a worked step-by-step example of the algorithm being used. Say we have the following as the true graph (Figure 3.1):

Step 1: First, we run the PC algorithm with depth set equal to 0, giving Figure 3.2.

Now we examine each variable, and if it has an an indegree of 0, we call it an input. Otherwise, it is an output. In Figure 3.2, 1, 2, 3, and 4 are inputs, while 5, 6, 7, 8, and 9 are all outputs.

Note; If the input variables are known, as is often the case, Step 1 can be skipped (though input/output dependences still need to be found).

Step 2: Next, we look to see which output variables have a common set of directly connected inputs. In this case, we have two sets: $OUT(< 5, 6, 7 >)$ is connected to $IN(< 1, 2 >)$, while $OUT(< 8, 9 >)$ is connected to $IN(< 1, 2, 3, 4 >)$. For each of these input/output pairs, we posit a latent variable between the input and output .

Note: On the assumption that the system is linear, the number of latent variables can be inferred from the rank of the correlation matrix of the output variables – provided there are no causal connections among the measured

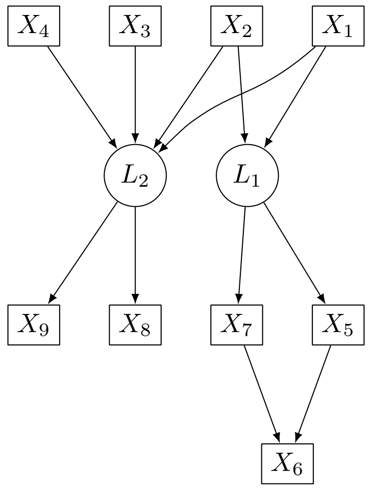


Figure 3.1: True graph

variables.

Step 3: We now check to determine which (if any) of the input sets are a subset of another input set. In this example, $IN(<1,2>)$ is a subset of $IN(<1,2,3,4>)$. We draw a path from the latent beneath $IN(<1,2>)$ to the latent beneath $IN(<1,2,3,4>)$. We also remove 1 and 2 from $IN(<1,2,3,4>)$, giving $IN(<3,4>)$. Finally, we draw a path from each input to the latent beneath the input, as well as draw a path to each output from the latent above the output.

This gives us the following graph (Figure 3.4):

Step 4: We now apply the next step in the algorithm (i.e., Sober (1998)). In this step, we check to see whether conditioning on some set of inputs (belonging to two latents connected by a directed edge) produces independence between the outputs of the two connected latents. If so, then we remove the path connecting the two latents, and draw paths from the conditioning set of inputs to each latent. In the example, this means that we remove the path between latents 1 and 2, and we connect X_1 and X_2 to both latents. Doing so yields the following

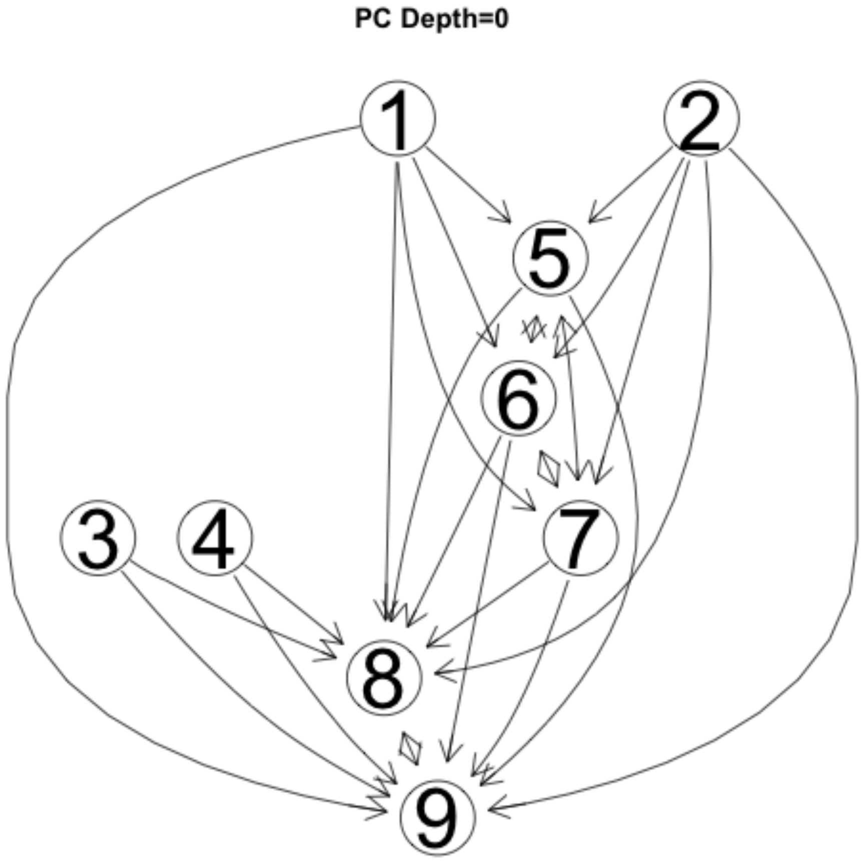


Figure 3.2: The pattern returned by the PC algorithm, with depth set to 0.

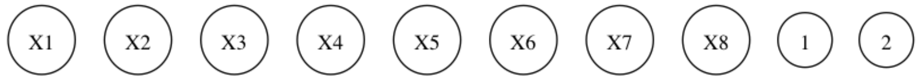


Figure 3.3: List of all known variables.

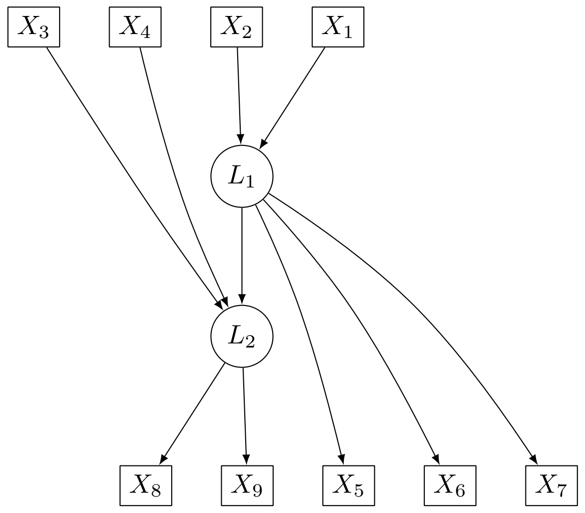


Figure 3.4: The graph prior to applying Sober's criterion.

graph (Figure 3.5):

Step 5: Finally, we rerun the PC algorithm (but with a depth greater than zero). We continue running the PC algorithm (increasing the depth by one each time) until none of the reported graph adjacencies change.

Step 6: Using this graph (Figure 3.6), we look to see if any of the output variables have no direct edges connecting them to an output variable. If so, we then disconnect that (former) output variable from its latent, and connect the variable to the outputs (still connected to the latent) using the adjacencies reported in the $PC(depth = 0)$ graph. In the example, output 6 has ceased to have any direct edges connecting it to an input variable. Therefore, we disconnect X_6 from latent 1, and draw edges to X_6 from X_5 and X_7.

Step 7: We now end the algorithm, and return the graph depicted in Figure 3.7.

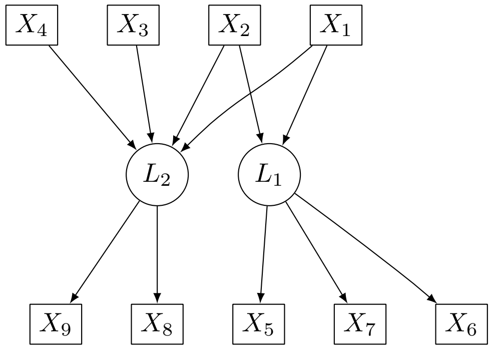


Figure 3.5: The graph after applying Sober's criterion.

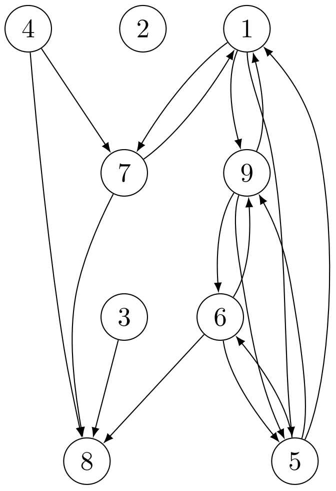


Figure 3.6: The pattern reported by the PC algorithm, with depth greater than 0.

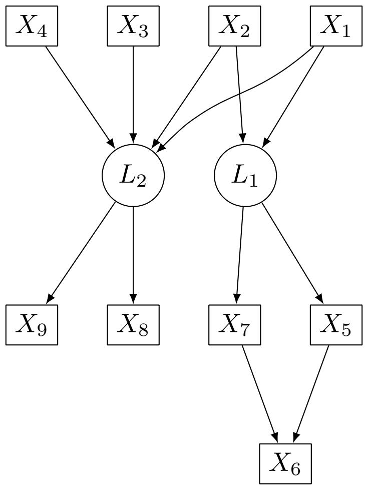


Figure 3.7: Final graph.

3.5 Proof of Correctness

Proof of correctness for step 1 (classifying input and output variables)

Due to assumptions A1, A2, and A3, the PC algorithm will produce a pattern consistent with the unconditional independence relations true of the measured variables in the true graph. Using this pattern, every input variable from the generating graph will only have adjacencies connecting it to output variables in the generating graph (as assumption A7 forbids adjacencies between input variables).

For every pair of variables that are inputs in the true graph, there will be no adjacency between the two variables in the PC pattern (by A7).

For every variable that is an input in the true graph and every output that is a descendant of that input, there will be an adjacency in the pattern returned by PC (by A6).

All of these adjacencies in the pc.pattern will ultimately be a directed edge from an input to an output variable, as the only paths from inputs to outputs in the PC graph output will be through unshielded colliders (due to assumptions A5 and A6). Therefore, every input will have a total degree of no more than 0. Finally, due to assumption A8, every output variable must have an indegree greater than 0. So step 1 correctly classifies the input and output variables.

Proof of correctness for step 2

As every edge connecting an input to an output in pc.pattern must be the result of a path through a latent in the true graph (due to A6), and every output variable is a descendant of a latent (due to A8), there must be at least one latent (assuming the PC graph is not empty).

If there are sets of outputs whose members only have edges (in the pc.pattern) to some subset of the inputs, then there must be more than one latent (due to A6), and each of these sets of outputs must have its own latent as the only path from an input to an output is through a latent (again due to A6). This yields the correct number of latents.

Proof of correctness for step 3

If the input set of a latent (a) is a subset of the input set of another latent (b), and a is the largest such subset, then it must be the case that a is a latent cause of b (or latents a and b share some inputs). Otherwise, the inputs of a would have to have a path to the outputs of b via a non-latent (forbidden by A6), or via some latent between a and b (which is forbidden by the "largest subset" condition).

Proof of correctness for step 4

If step 3 reports an edge between two latents, then either that edge exists in the true graph, or the latents share some input variables (A4 forbids both being true simultaneously). Therefore, if there isn't an edge connecting the two latents in the true graph, then $OUT(x) \perp\!\!\!\perp OUT(y) \vee (IN(x)IN(y))$, as there would be no open path connecting $OUT(L1)$ and $OUT(L2)$. If there is an edge between L1 and L2 in the true graph, then $OUT(x) \not\perp\!\!\!\perp OUT(y) \vee (IN(x)IN(y))$.

Proof of correctness for step 5

PC can be used due to A1, A2, and A3.

Proof of correctness for step 6

If an output variable has no paths to an input variable (in the pc.infinite pattern), then that output variable must be a child of only other output variables, else conditioning on observed variables would be insufficient to block all paths between the output variable and the input variables.

Chapter 4

Simulations

4.1 Why Perform a Simulation?

Despite the rotation problem (described in the literature review), factor analysis remains a popular method for discovering causal structures. Analytic arguments against a method are not always sufficiently persuasive on their own. Therefore, simulations were run to support the analytic argument by providing both an illustration of the weakness of factor analysis and a source of comparison for the DM algorithm.

When searching for a MIMIC model structure, there are four main goals: finding the correct number of latents, classifying variables into inputs and outputs, clustering inputs and outputs around their respective latent(s), and determining the latent-to-latent structure. As factor analysis is incapable of performing the second and fourth goals, and measures of accuracy are only meaningful when being used as a source of comparison, the simulations only examine the first and second.

4.2 How the Simulations were Performed

Multivariate Gaussian data were generated from seven different causal graphs (see Figures 4.1 and 4.2 for a list of graphs). Each input variable followed a standard normal distribution. As the latent variables were generated by a linear combination of their inputs (and connected latents), they are simply a weighted sum (where the edge weights are all set to 1) of standard normal variables. Noise variables also followed a standard normal distribution. The outputs followed the same distribution, as they too were a weighted sum of standard normal variables. Factor analysis and the DM algorithm were both run on the same datasets, with sample sizes of 250, 500, 1000, and 10,000 observations. For each causal graph and sample size, the data were generated five hundred times and the algorithms run and scored on each dataset. Setting the edge weights to 1 is convenient but not optimal to demonstrate the accuracies of various algorithms, but random assignment of weights would only change the strengths of covariances, not the asymptotic constraints on the data that various models (e.g., a 1 factor model) imply.

As it is recommended to select the number of latents in factor analysis using several different methods (Hair et al., 1998), four different methods (non-graphical approximations of a scree plot) were used: optimal coordinates, acceleration factor, parallel analysis, and the Kaiser rule. Each method was allowed a vote for the recommended number of factors, and the number with the highest frequency was selected. In the event of a tie, the smallest number was selected.

The algorithms were judged based on two criteria: the number of cases where a given algorithm reported an incorrect number of latents (reported as the percentage of incorrect cases out of 500), and (assuming the algorithm reported the correct number of latents) the reliability of undirected edge discovery. Reliability was represented by the true positive[1], false positive[2], and true discovery[3] rates for reported edges. Reported graphs were judged on their ability to identify undirected edges correctly, as it is unclear how without prior knowledge measured variables are to be separated into input (exogenous) variables and output (endogenous) in a reported factor model.

[1]The true positive rate is the number of correctly found edges divided by number of true edges in the actual graph.

[2]The false positive rate is the number of incorrectly found edges divided by the number of true gaps in the actual graph.

[3]The true discovery rate is the number of correctly found edges divided by number of found edges.

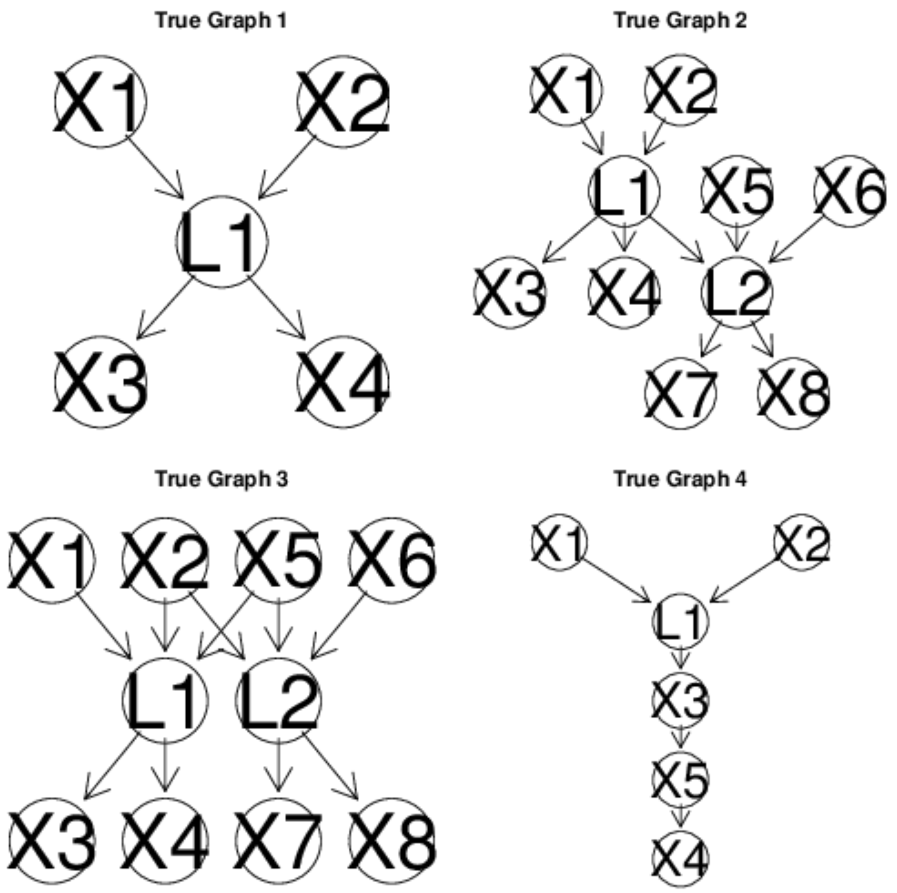


Figure 4.1: The first four graphs tested. Note that L1 and L2 are latent variables.

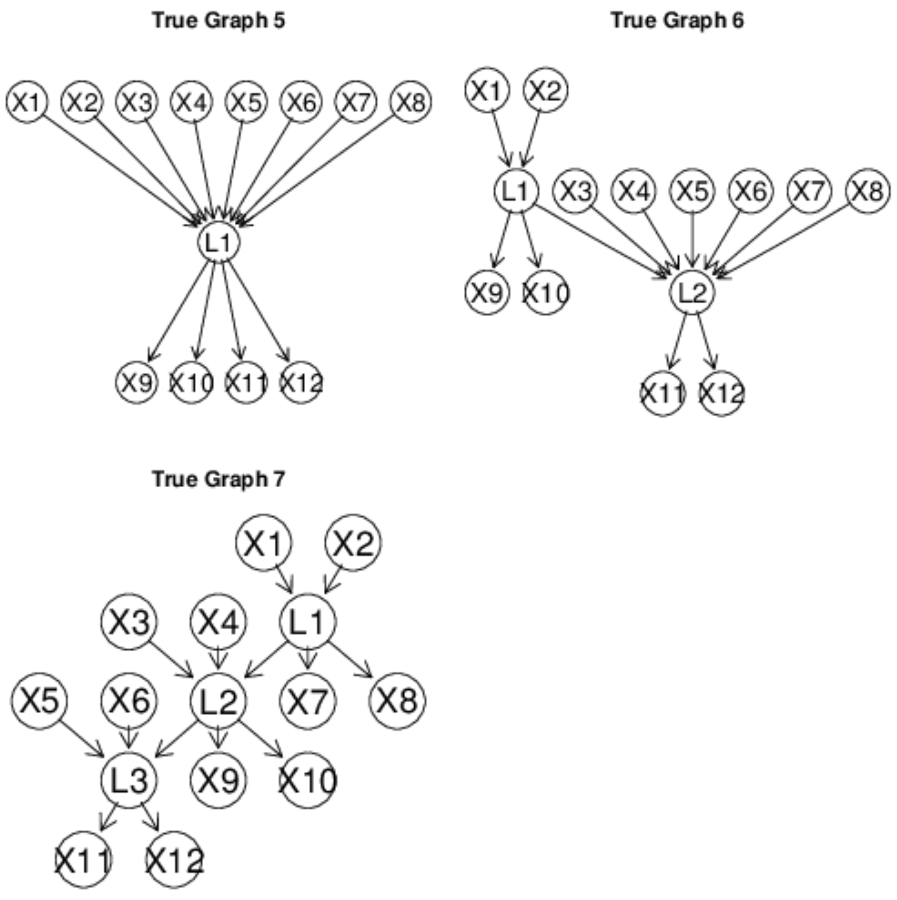


Figure 4.2: The second three graphs tested. Note that L1, L2, and L3 are latent variables.

4.3 Analysis of Results

In general, the DM algorithm outperformed factor analysis, with a lower false positive rate (graphs 2, 3, 4, and 6), a higher true discovery rate (graphs 2, 3, 4, 6, and 7), and a higher true positive rate (graphs 2, 3, 4, 6, and 7). The algorithm also proved to be correct about the number of latents more often, with the exception of graphs 3 and 5 where it performed marginally worse, and graph 6, where there were relatively many variables and few latents (as a single incorrect independence test can result in the creation of a new latent, this increased the odds of getting the number of latents wrong). Performance on graph 1 was similar for both methods. Finally, unlike factor analysis, the performance of the DM algorithm generally improved as sample size increased. Bar plots of accuracies are given in Appendix A.

Factor analysis is unreliable at best. In some cases, its performance decreases as sample size increases! For graphs 2 and 7, the number of correct latent cases became smaller as sample size increased (in some cases, factor analytical methods did not get the right answer once). In most cases, factor analysis was outperformed by the DM algorithm, except for graphs 1 and 5, where it marginally performed better. However, given the general unreliability of factor analysis for correctly determining even a small number of latents, such a performance cannot be reasonably extended to general cases (such as those encountered during a data analysis).

Chapter 5

Data Analysis

5.1 Goals of the Analysis

The two primary goals of analysis were:

1. Cluster the genes (inputs) and expressions (outputs) around their respective latents.

2. Find latent-to-latent connections.

As a secondary goal, any unusual patterns, such as an uncommon or atypical number of input variables tied to a single latent in the resulting MIMIC graph, are noted for future study.

5.2 Algorithm Assumptions and the Dataset

In the dataset, many of the algorithm's assumptions are known to be satisfied. Inputs are independent of one another (as genes do not "cause" one another), no input has a path to an output except via a latent (it is impossible for a gene to directly affect mRNA), and every measured variable is either an input or an output.

As variables are already classified as inputs or outputs, we can relax the assumption that every latent have 2 inputs and 2 outputs. Now they need only have 1 of each. This also means that we do not need to run the PC algorithm, but can instead simply record which inputs and outputs are dependent. Since it is impossible for mRNA (output) variables to directly cause other mRNA variables, we can also omit the other run of PC used in the last step of the DM algorithm.

The true graph may prove to not be singly connected, however this only means that some information will be lost (e.g., an input shared by 2 connected latents may be reported as being directly connected to only 1 latent). In which case, the algorithm is reporting one of several graphs consistent with the observed independence relations.

The most dubious assumption made is that of an acyclic graph. Cyclic relations between gene expressions and latent variables have been observed in other genomic datasets. In the event of this assumption being violated, latents can be incorrectly merged, and in some cases the reported latent-to-latent structure can be incorrect. At present there is no known solution to this problem, and any results reported by the algorithm should be interpreted with this caveat in mind.

5.3 The Initial Data

The original data were gathered using microarray analyses and massively parallel sequencing (Network et al., 2011). Every observation was a patient with ovarian cancer. Each patient had two classes of variables recorded. First, every gene was categorized based on whether it was mutated[1] or had multiple copies (i.e., more than a single possibly partial copy of a chromosome) in some fashion. If so, that gene was coded as "1". Otherwise, it was coded as "0".

Second, gene expression levels, in the form of mRNA, were measured as continuous variables. This continuous data was then split into three-level ordinal variables. The cutoffs[2] were chosen based on 25% and 75% of the range of values, with below 25% coded as "-1" (for low gene expression level), above "75%" as "1" (for high), and all others cases as "0" (for normal).

At the beginning of analysis, the dataset consisted of 562 patients, all with ovarian cancer. There were 17,610 genes (inputs), as well as 12,042 gene expression variables (outputs).

The mean number of mutations was 205 and the median was 124.5. The largest number of mutations any subject had was 2,968. There were no subjects with zero abnormalities. Figure 5.1 depicts the frequency distribution of abnormalities per subject.

The average gene was mutated in 6.544 subjects, though the median number of subjects was 4. The most frequently mutated gene was mutated in 382 subjects. Figure 5.2 depicts their distribution.

[1]Meaning a change in at least a base or an allele due to: non-synonymous substitution, frameshift, deletion, or truncation.
[2]The cutoffs were chosen by the creator of the dataset.

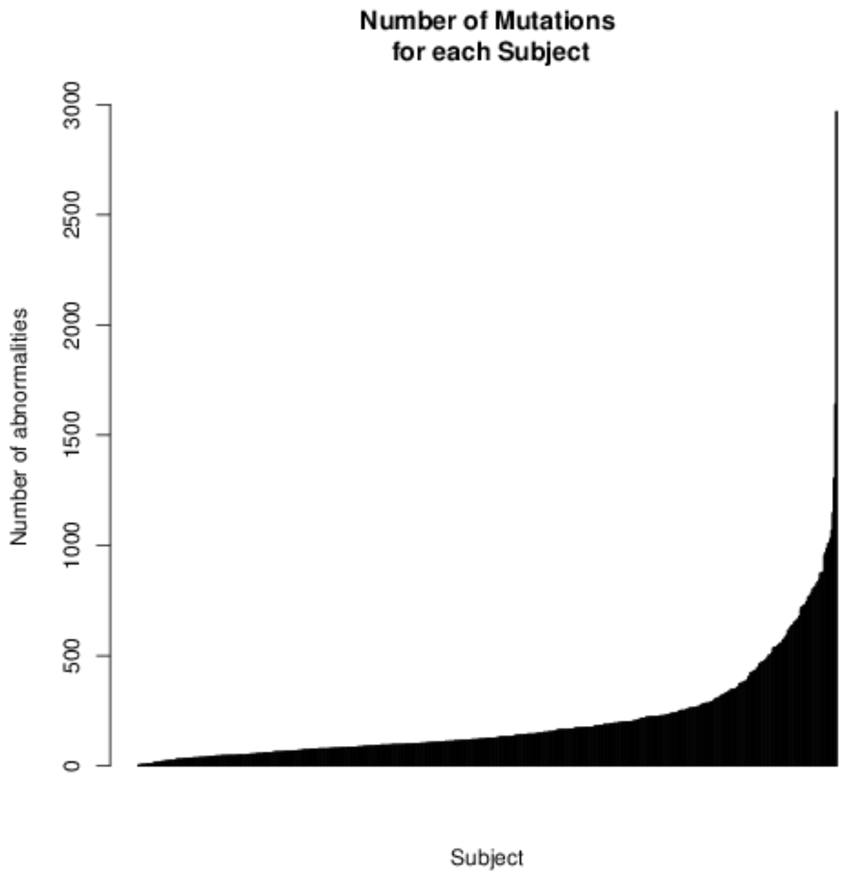


Figure 5.1: Number of abnormalities for each subject.

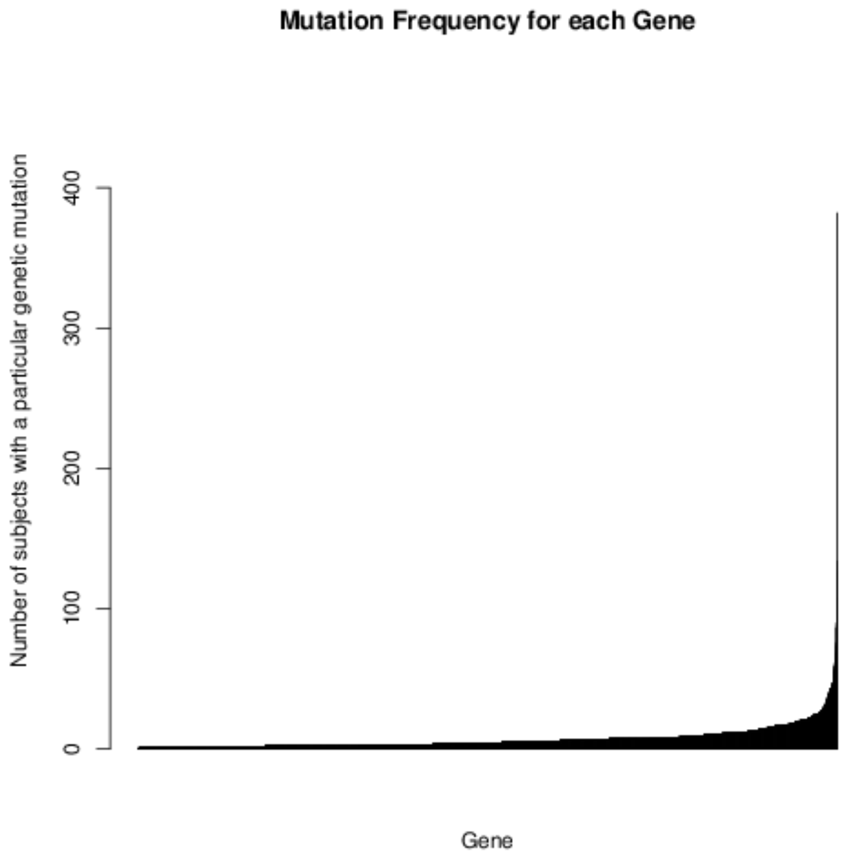


Figure 5.2: Number of subjects with a given abnormality.

5.4 Dimension Reduction

Due to computational constraints in both memory and time, the algorithm cannot (as implemented) be run on the complete dataset. Therefore, cross-validated[3] lasso regression[4] (Hastie et al., 2009) was used in order to reduce the number of variables in the dataset. This regression was used in two ways. First, it was used to remove gene variables by predicting expression levels using genes, taking advantage of the sparsity forced by the lasso regression to exclude less useful gene variables. Similarly, lasso regression was used to remove expression variables by predicting genes using expression levels, again exploiting the forced sparsity property of lasso regression. Lasso regression was used (instead of simply randomly selecting some number of variables for inclusion) in order to increase the chance a selected node was related to another node.

As a constant cannot be used as the response of a regression, constant gene and expression variables were dropped.

The reason categorical expression variables were used instead of continuous versions is that the function used to perform the cross-validated lasso regression reported ambiguous errors (likely due to some variables having very few instances of one of the possible categories) when using a binomial response (i.e., when predicting whether or not a gene was mutated). This problem of ambiguous errors necessitated treating the gene variables as continuous. To avoid mixing variable types, the expression variables were also treated as continuous, despite being ordinal categoricals. Doing this had the added benefit of increasing the rate with which the lasso regressions could be performed.

Even after using lasso regression to select variables, too many variables re-

[3]Mean squared error was used as the criterion for judging models.

[4]As a number of variables had some categories which rarely occurred, the lasso regression assumed the variables were continuous. Doing otherwise would have resulted in many variables being dropped for no reason other than a lack of variability. Note that as a result, no regression coefficients produced by the lasso are interpretable.

mained. Therefore, an additional reduction was performed. For each of the 29,652 lasso regressions (17,610 with genes as the response, 12,042 with expressions as the response), the number of chosen predictors was examined. Only variables that belong to an unusually large group of predictors (of a size greater than the 99th quantile) were preserved. In other words, a variable was only included if it was part of a large collection of predictors in at least one lasso regression. Specifically, a group had to be greater than 69 predictors for gene variables and 141 predictors for expression variables in order for its members to be preserved in the final dataset. This reduced the total number of variables in the dataset to 4,369.

5.4.1 Side Effects of Dimension Reduction

As any dimension reduction procedure necessarily results in the exclusion of some variables, violations of the various assumptions made by the DM algorithm can occur no matter what reduction procedure is followed. The algorithm (when used on the cancer dataset) assumes that each latent variable has at least one gene mutation (input) and gene expression (output) variable. If the dimension reduction procedure were to drop all of a latent's inputs or outputs, then the structure reported by the algorithm can be incorrect. As a latent might only posses a single input or output, simply excluding 1 variable can lead to errors. One possible error, which may partially explain the lack of latent-to-latent edges in the graphs shown in Figures 5.4 and 5.5 (though such absences could also be the truth), occurs when a latent losses all of its outputs. Consider the following graph (Figure 5.3). If the algorithm were run on the observed variables in it, the resulting graph would have no edges between latents, even though the true graph contains such an edge (It would also report too few latents). As we do not already know the underlying causal structure in the dataset, we

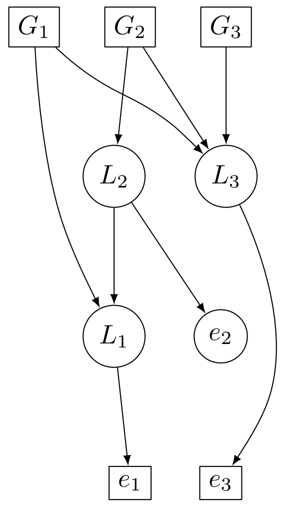


Figure 5.3: The above graph gives an example where, due to a dimension reduction procedure, the output variable e_2 has been omitted, resulting in the DM algorithm returning an incorrect graph.

cannot determine if this situation has occurred. It is thus advisable to avoid any dimension reduction when deploying the DM algorithm. Due to the limited computational resources available, this guideline had to be ignored. Caveat lector.

5.5 Graphical Results

To construct the initial adjacency matrix, Fisher's exact test (of independence) was used instead of the PC algorithm, as inputs and outputs were already known and doing so reduced the number of independence tests performed. For Sober's step, a chi-squared test was used. A p-value of .000001 was used when performing both kinds of tests.

After running the algorithm on the reduced dataset and removing all nodes with a total degree of zero, 445 variables remained. The reported graph (Fig-

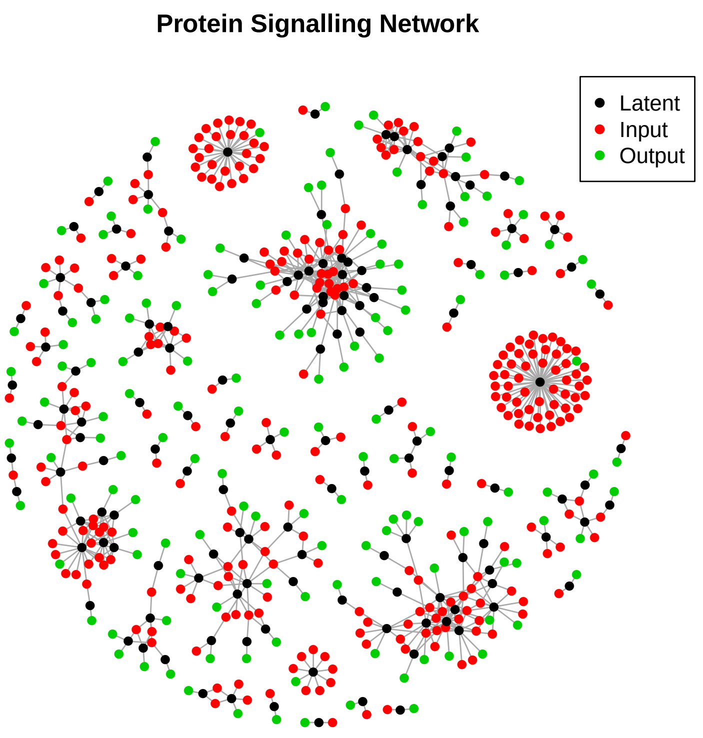


Figure 5.4: Initial reported signalling network. Note that due to the number of nodes, edges can sometimes overlap.

ure 5.4) contains 139 latents, 295 inputs, and 150 outputs.

5.5.1 Interpretation

Interestingly, there are a number of disconnected subgraphs in the reported graph (Figure 5.4). There are three subgraphs where a single output variable is related to a large cluster of input variables. Although this is a possible consequence of the dimension reduction procedure (as relevant gene expression level variables capable of “breaking up” the cluster may have been excluded), the result may point to gene expression variables usable as indicators for the presence of ovarian cancer. There are also several subgraphs with more complicated network structures. Enlarged graphs of these clusters are included in Appendix B.

One interesting thing to note (in both Figures 5.4 and 5.5) is the relative absence of latent-to-latent connections (indeed, there was only a single latent-to-latent edge. Latent 88 caused latent 137 in Figure B.22). One possible explanation (covered in more detail in the section on dimension reduction) for this phenomenon is that, due to the dimension reduction procedure, a single latent is “standing in” for a number of others. Therefore, while there may be many latent-to-latent connections, their existence is being covered up by the presence of some disconnected latents that are wrapped up in the single “representative” latent. An alternative explanation is that the p-value chosen for performing Sober’s step was too extreme.

As a nice sanity check for the reported graph, note that several gene variables (HOOK2, GIPC1, NDUFB7, and PIN1) have directed paths to their (known to be) related gene expressions (HOOK2.1, GIPC1.1, NDUFB7.1, and PIN1.1). An unsystematic search of the literature failed to discover any clear association between regulatory role and latent clustering.

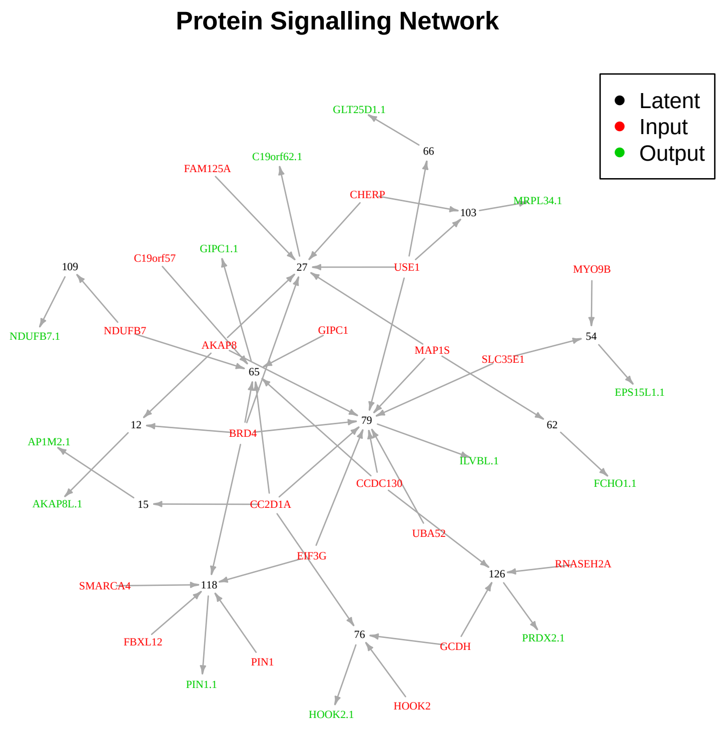


Figure 5.5: A subgraph of the larger protein signalling network. Note that the black numbers are simply labels for the latent variables. Also note that some output variables have a “.1” appended to their name in order to avoid naming conflicts with input variables. There is an enlarged version in Appendix B.

Chapter 6

Future Work

There are a number of different possible avenues for future research.

The implementation using backwards and forwards regression for variable elimination and a very low alpha value probably resulted in a structure for the genomics data that is too sparse in important respects. In a practical scientific application, the algorithm would need to be run with variations in these strategies, and/or using a false discovery rate. The algorithm could be improved if methods can be found that allow relaxing various assumptions (BPC and MIMBuild seem likely candidates), most importantly:

1. Acyclicality (A3).
2. Inputs are independent of one another (A7).
3. Singly connected graph (A4).

Some properties of the algorithm, such as its computational complexity could also be formally calculated. Another possible avenue for research is more thorough simulations, involving:

1. Randomly generated MIMIC models, with different numbers of latents, input-to-output variable ratios, and latent-to-observable variable ratios.

2. Different parameterizations (and distributions).

It is also worthwhile to determine how robust the results of the algorithm are given different dimension reduction (and other data analysis) decisions. For example, how does the reported graph change when the p-value used to perform Sober's step changes?

In the event more computational power becomes available, running the algorithm on the entire genomic dataset would be of interest, as well as testing the implications of the results. As the computational power needed to run the algorithm on a dataset of around 30,000 variables is very high, a parallelized version of the algorithm, as well as access to a supercomputer may need to be investigated. There are also a number of other datasets, including those from neuroscience, psychology, economics, and other subfields of genomics, on which the algorithm could be run.

Appendix A

Simulation Results

This appendix contains the results for the simulations discussed in chapter 4.

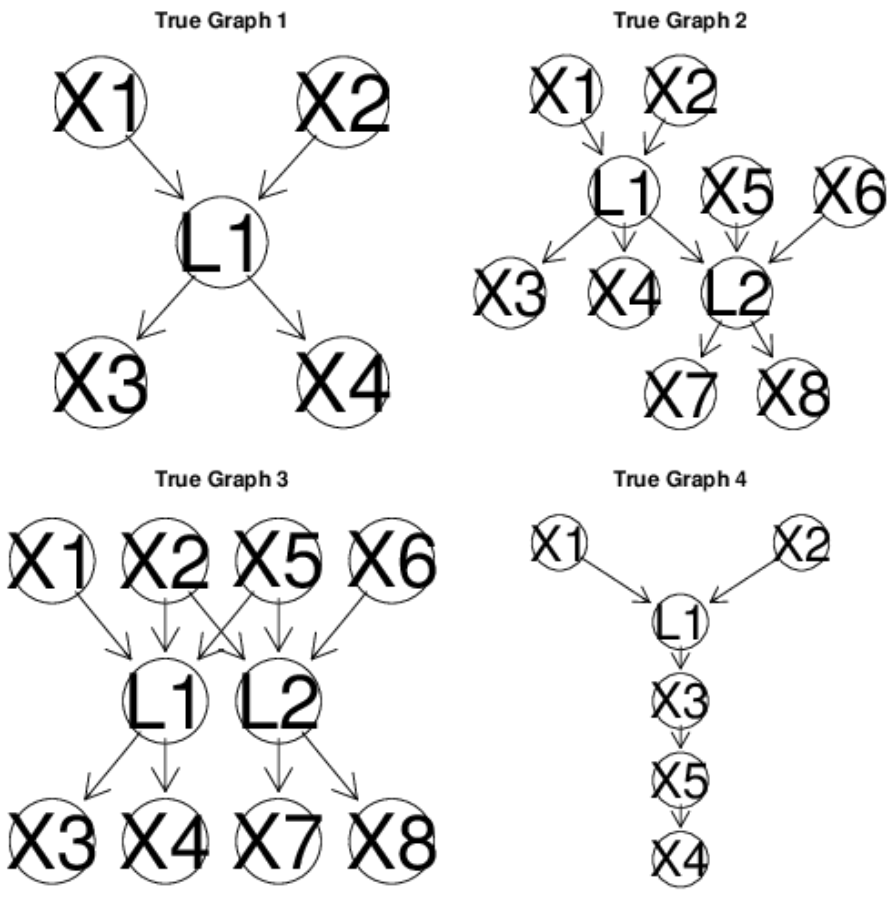


Figure A.1: The first four graphs tested. Note that L1 and L2 are latent variables.

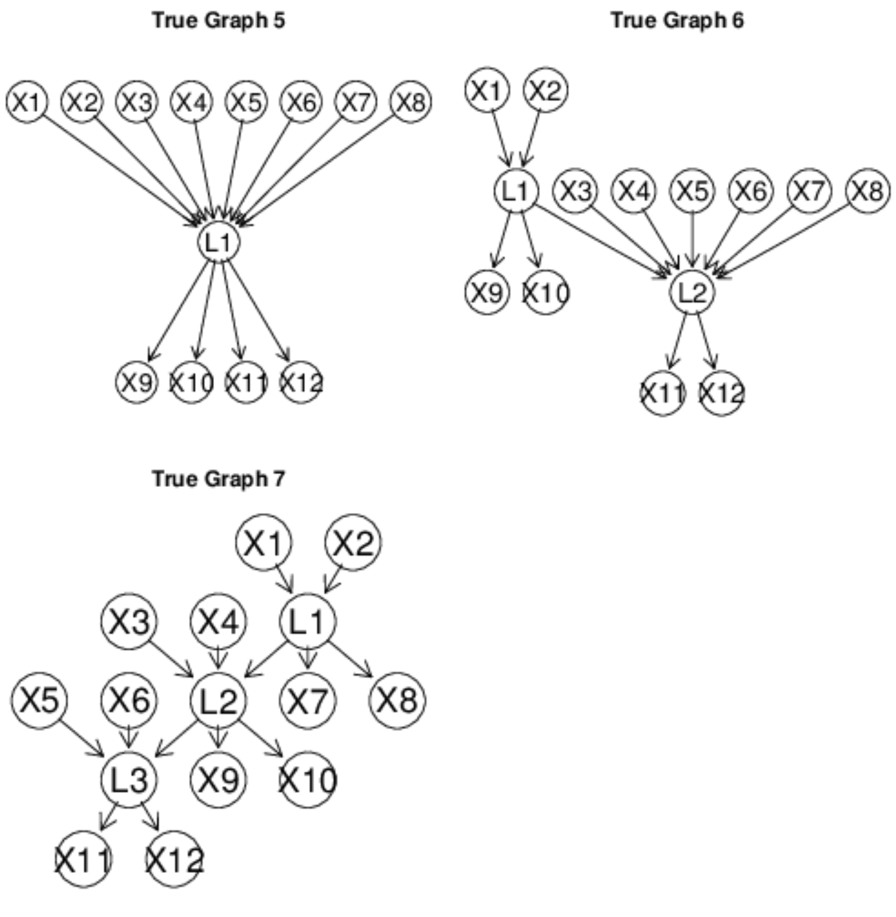
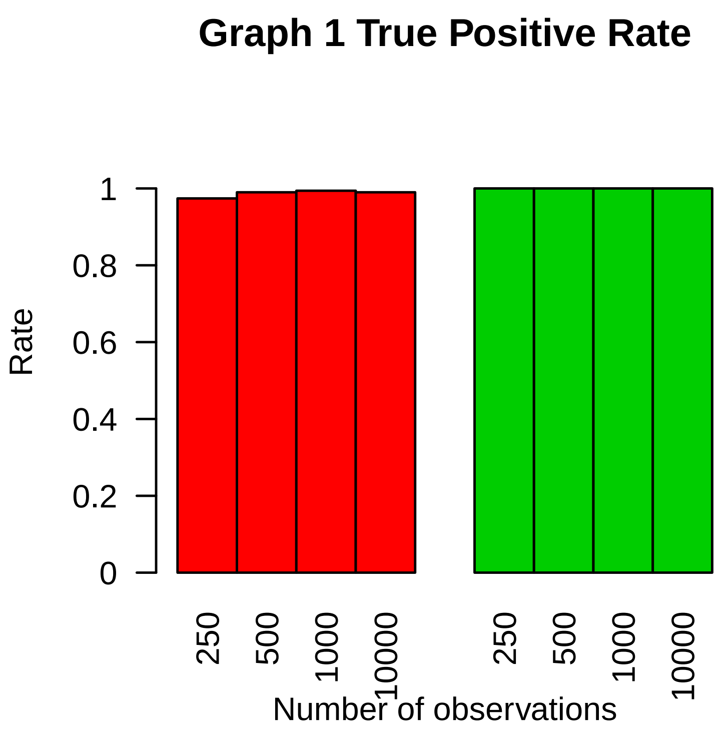
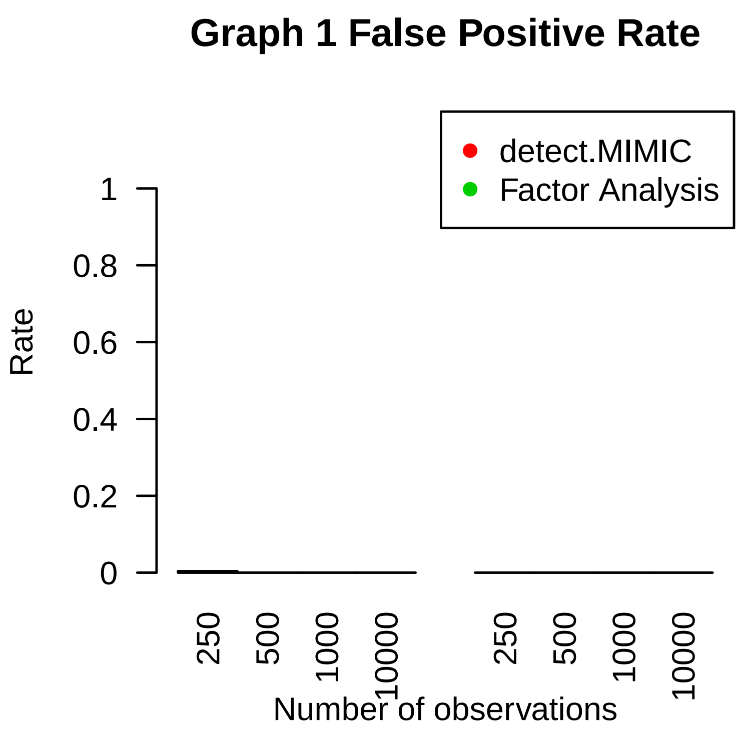
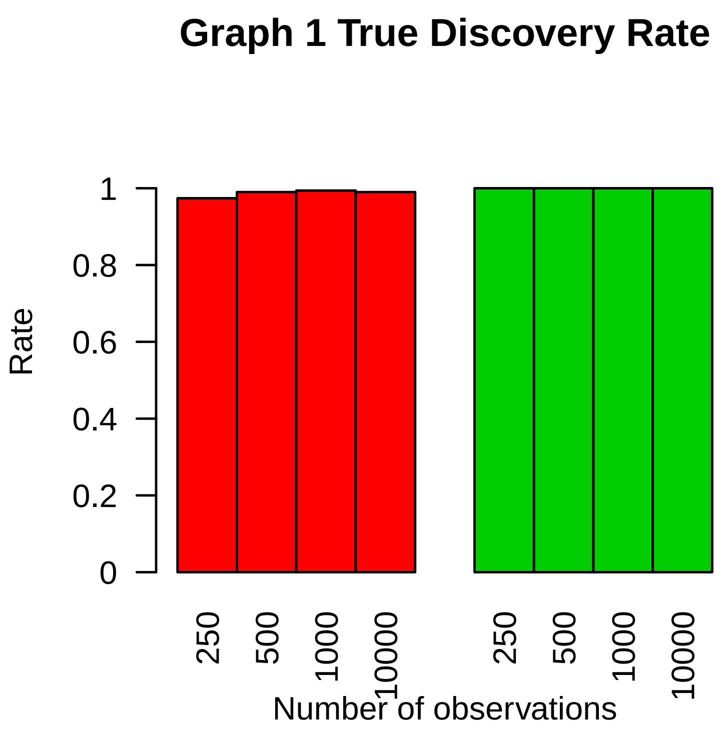
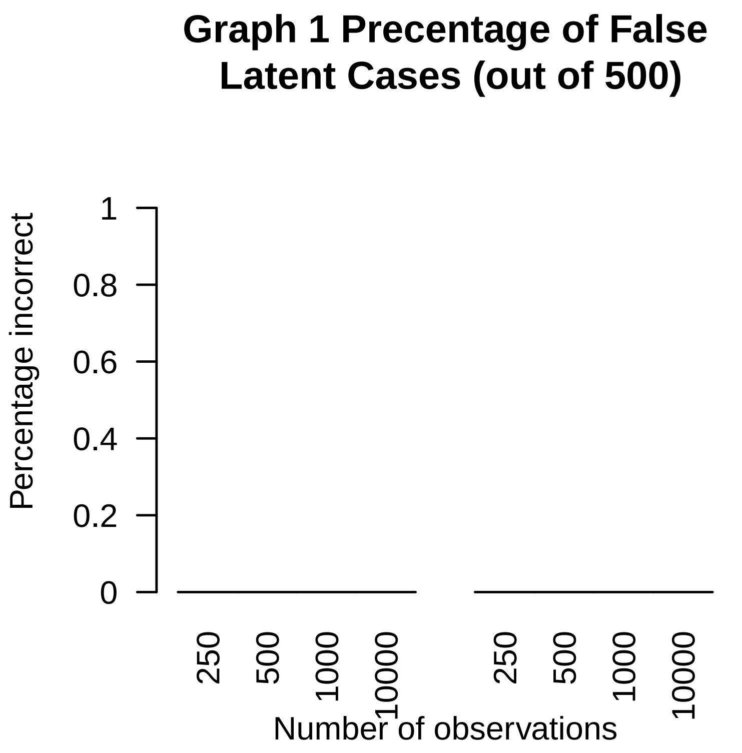


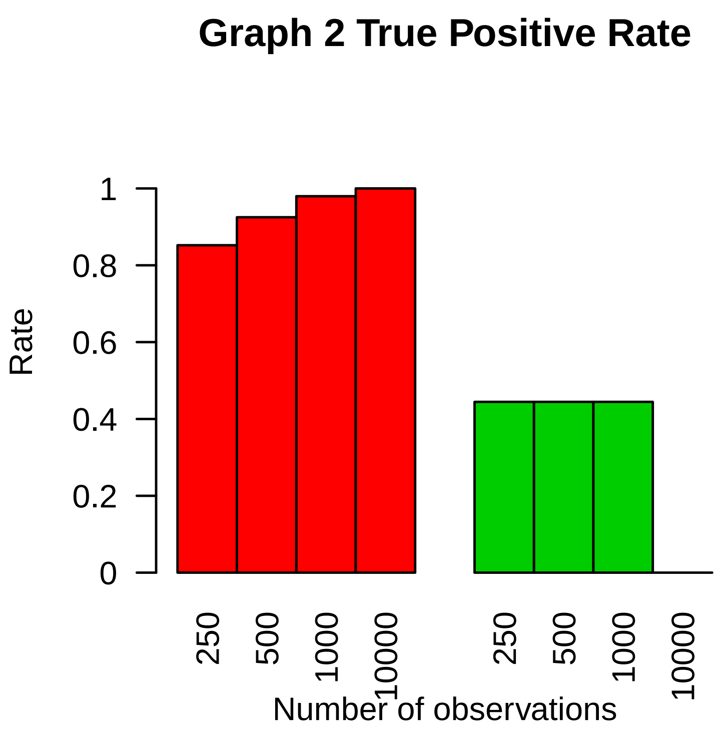
Figure A.2: The second three graphs tested. Note that L1, L2, and L3 are latent variables.

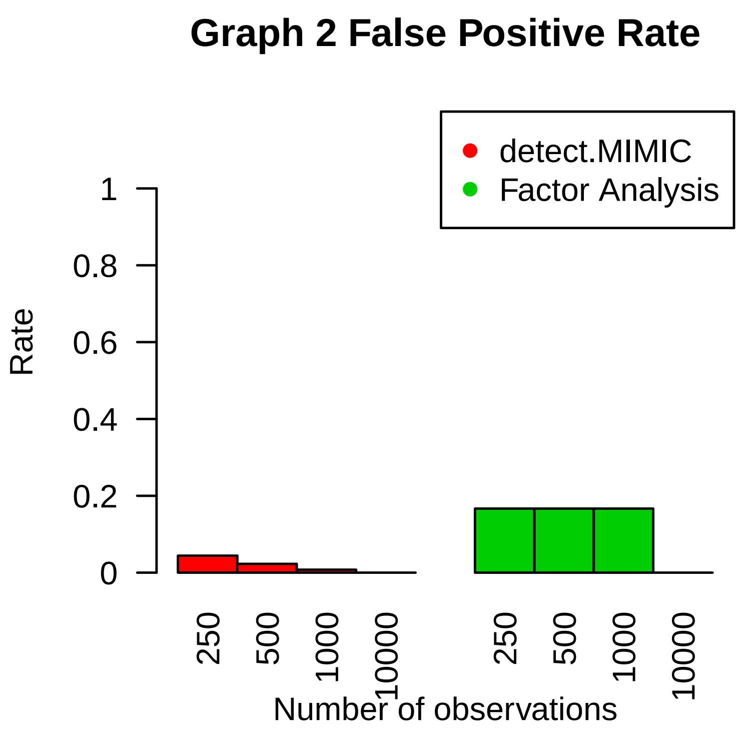

Graph 1 True Positive Rate
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

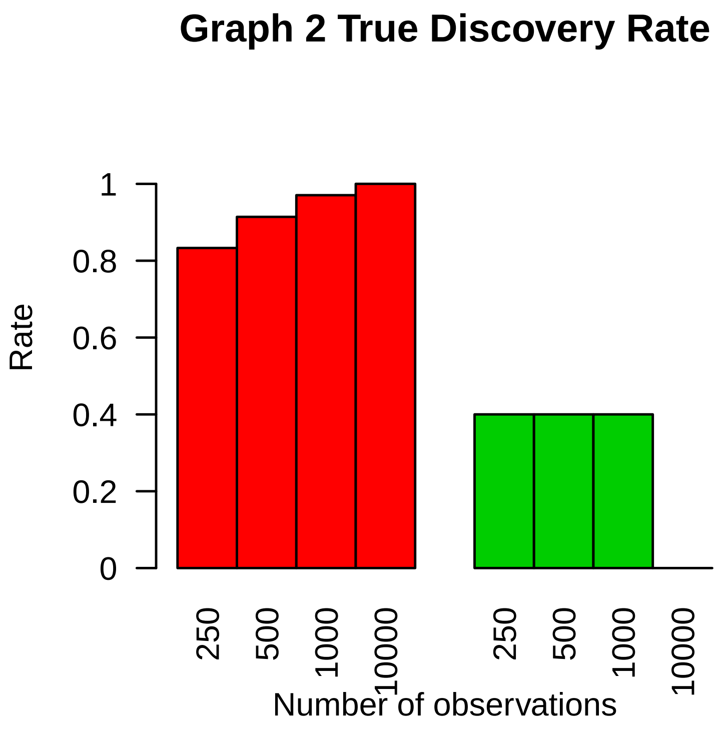

Graph 1 False Positive Rate
● detect.MIMIC
● Factor Analysis
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

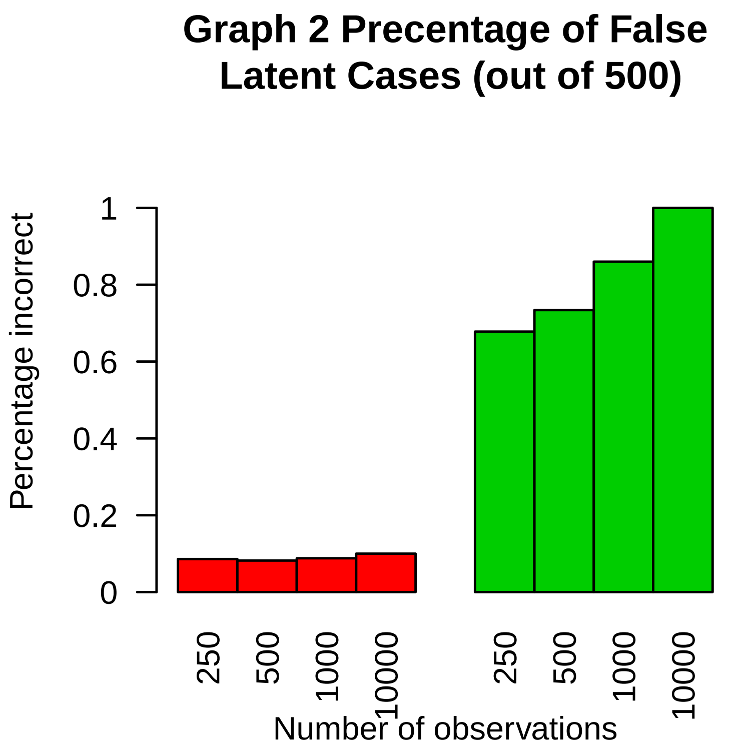

Graph 1 True Discovery Rate
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

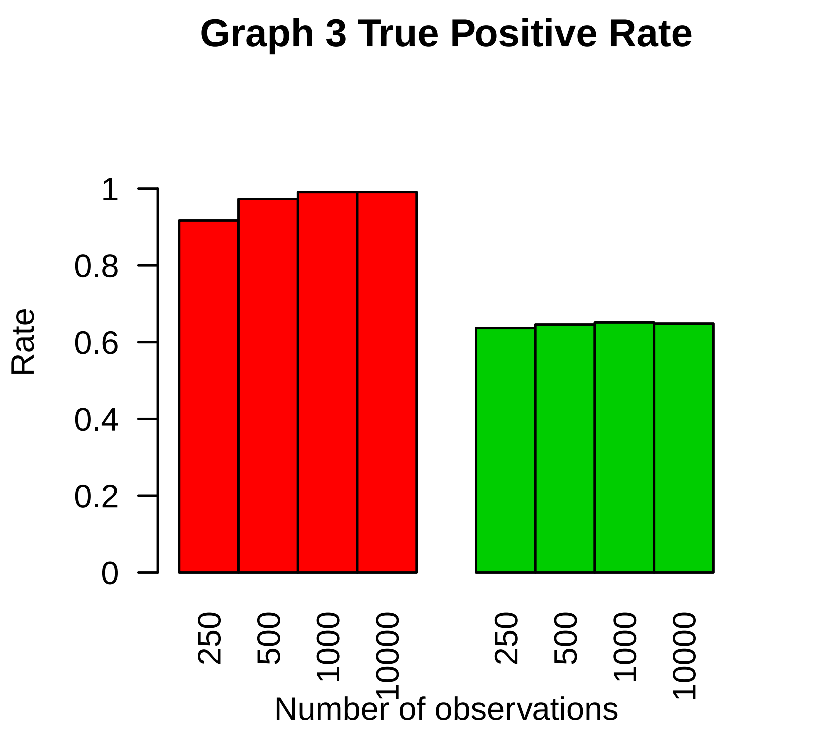

Graph 1 Precentage of False Latent Cases (out of 500)
Percentage incorrect
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

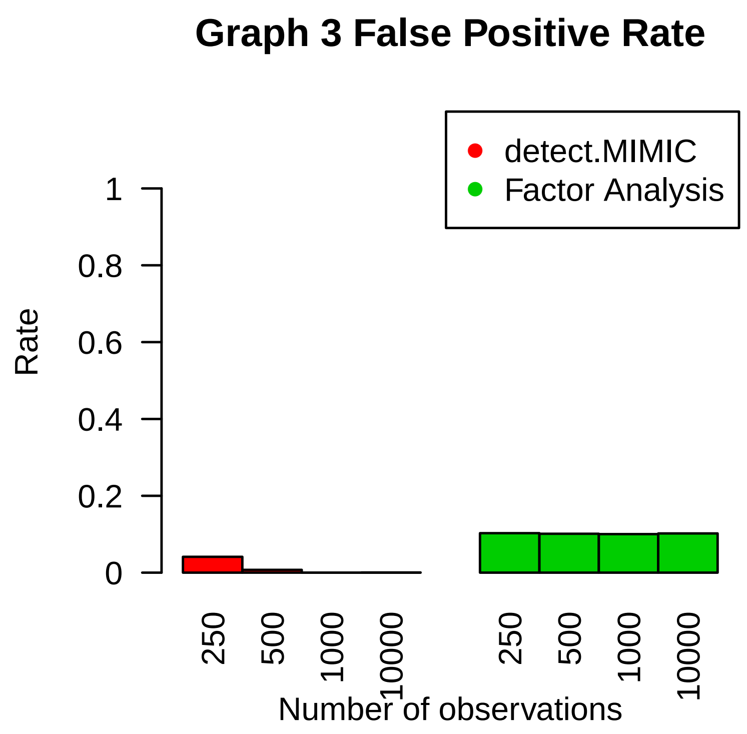

Graph 2 True Positive Rate
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

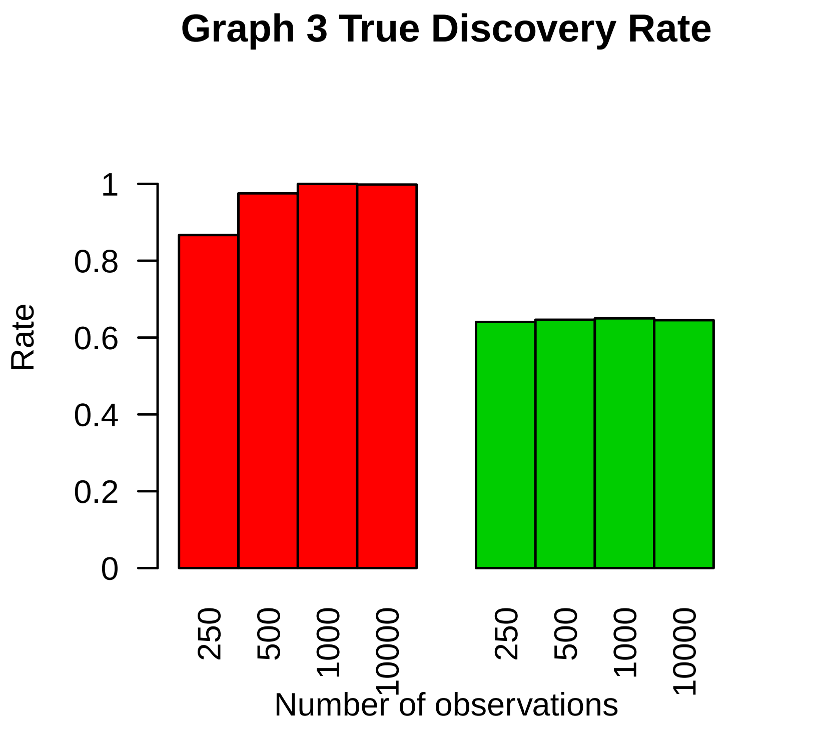

Graph 2 False Positive Rate
● detect.MIMIC
● Factor Analysis
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

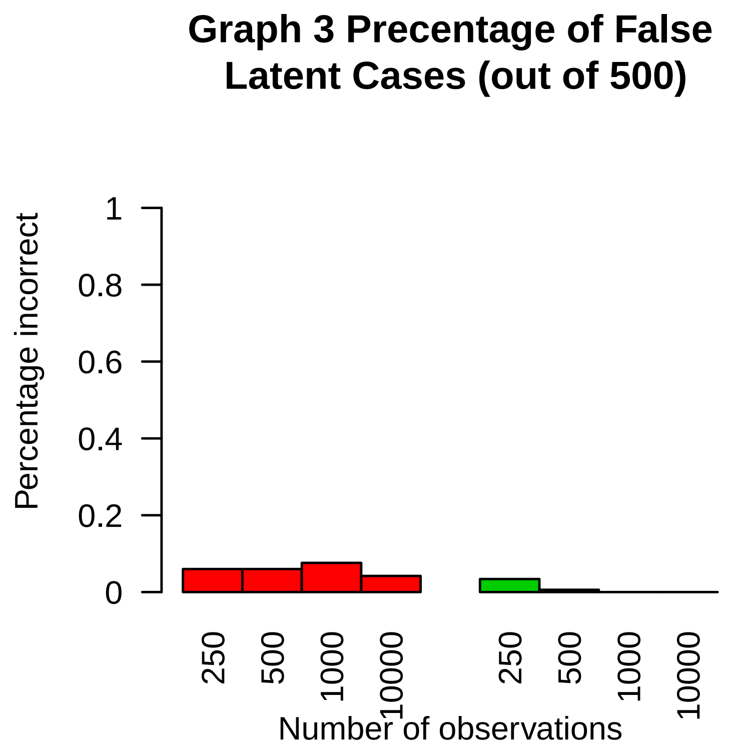

Graph 2 True Discovery Rate
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

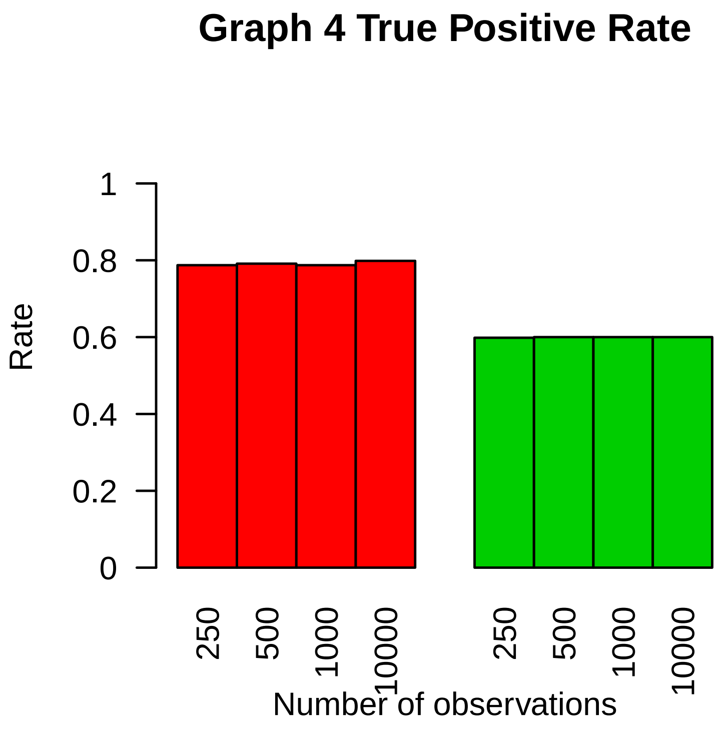

Graph 2 Precentage of False Latent Cases (out of 500)
Percentage incorrect
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

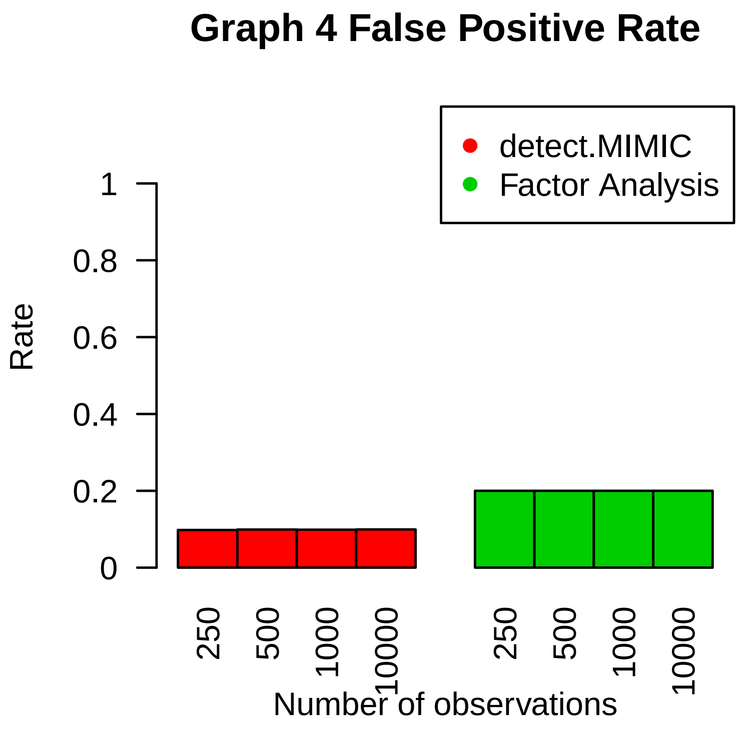

Graph 3 True Positive Rate
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

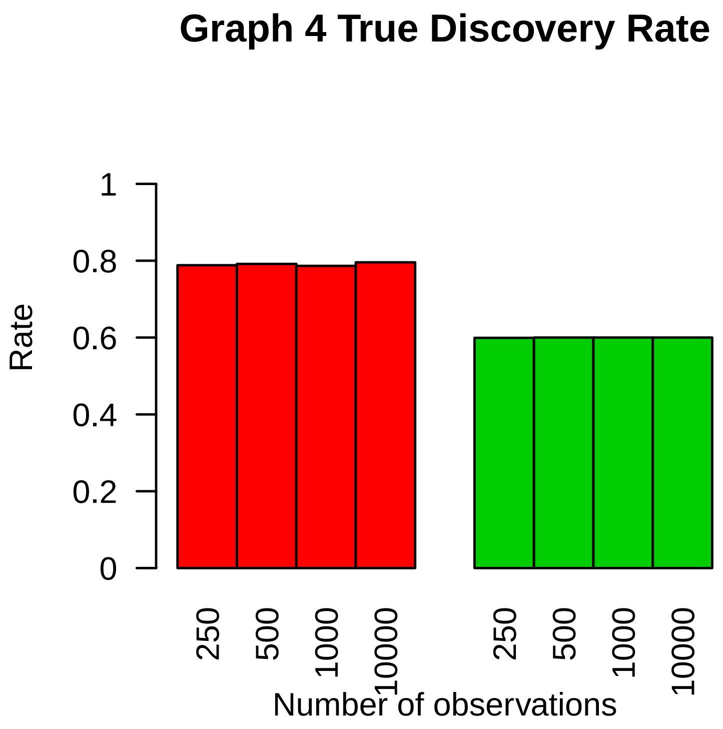

Graph 3 False Positive Rate
● detect.MIMIC
● Factor Analysis
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

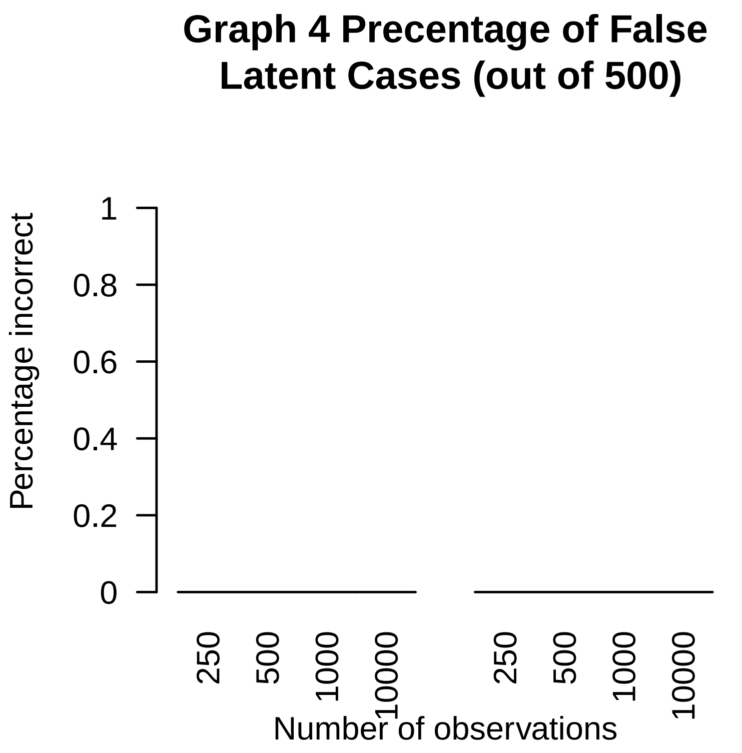

Graph 3 True Discovery Rate
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

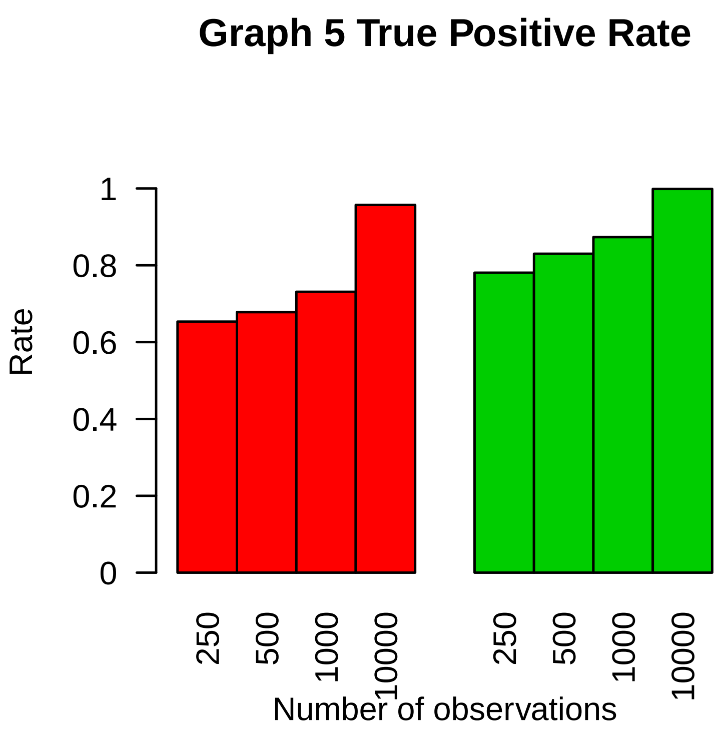

Graph 3 Precentage of False Latent Cases (out of 500)
Percentage incorrect
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

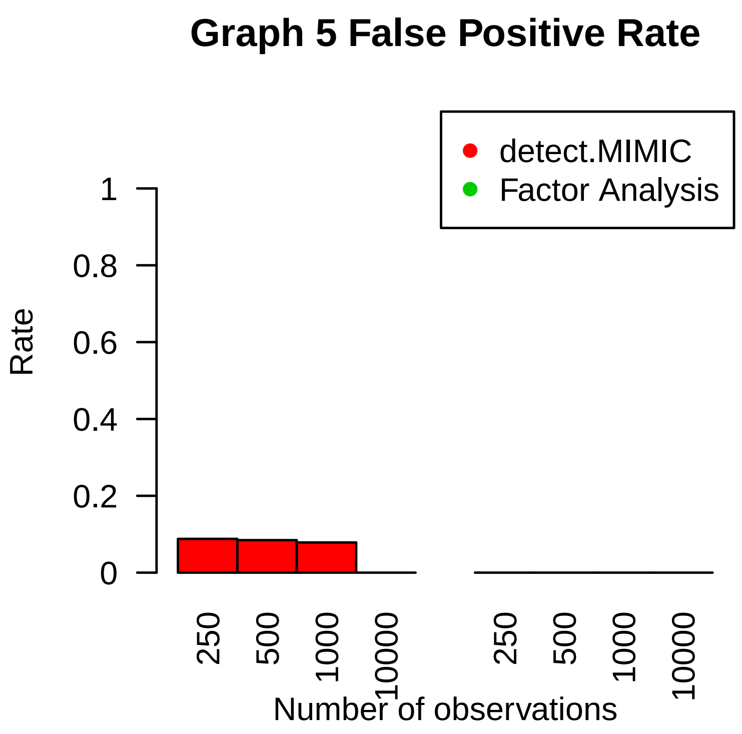

Graph 4 True Positive Rate
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

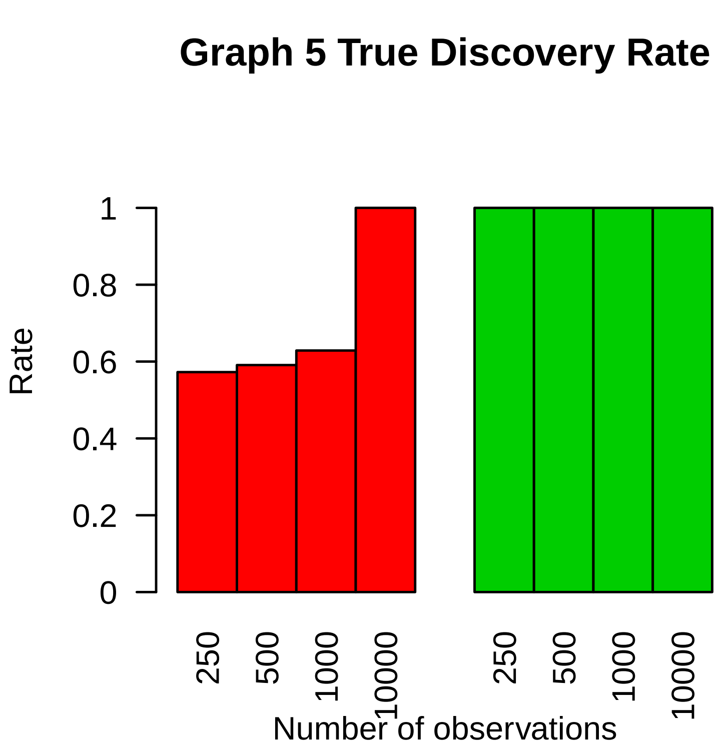

Graph 4 False Positive Rate
● detect.MIMIC
● Factor Analysis
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

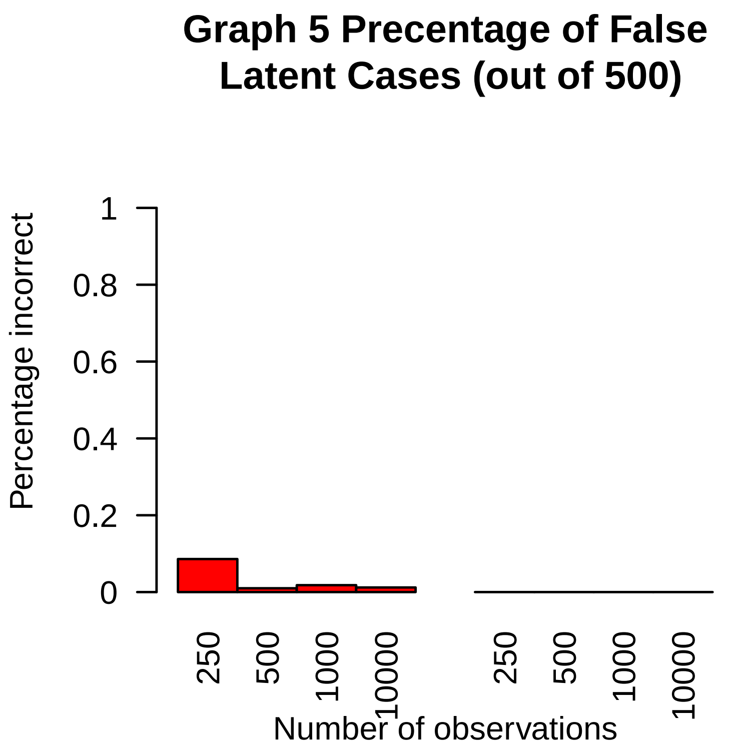

Graph 4 True Discovery Rate
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

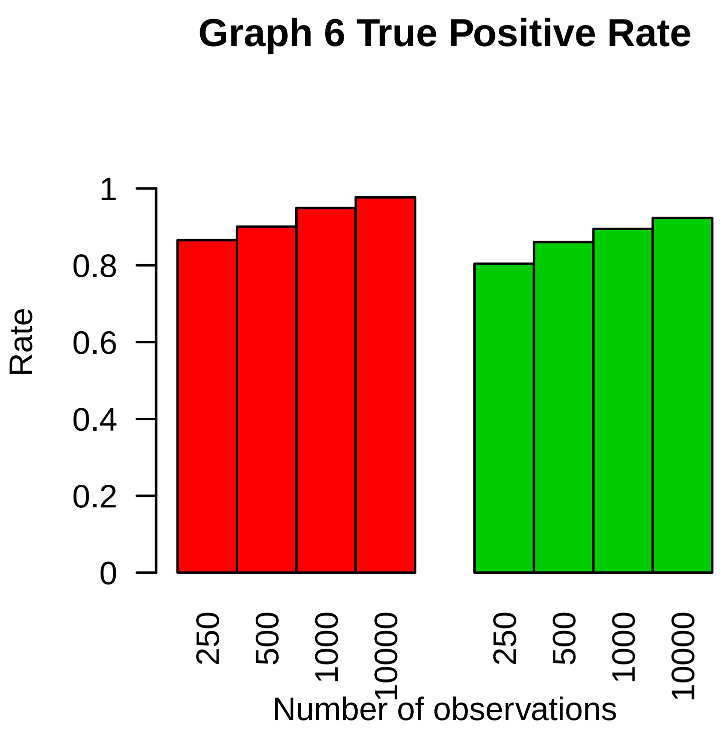

Graph 4 Precentage of False Latent Cases (out of 500)
Percentage incorrect
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

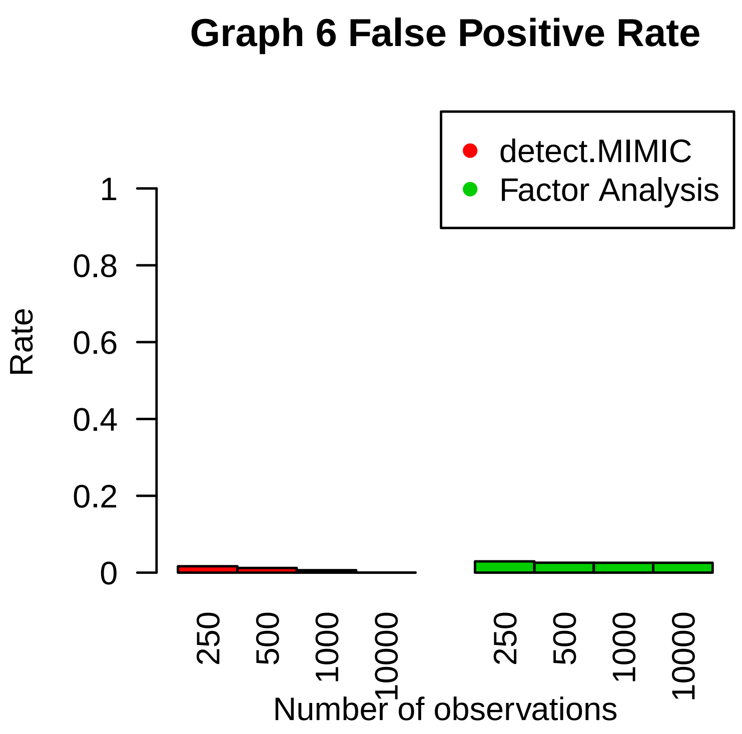

Graph 5 True Positive Rate
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

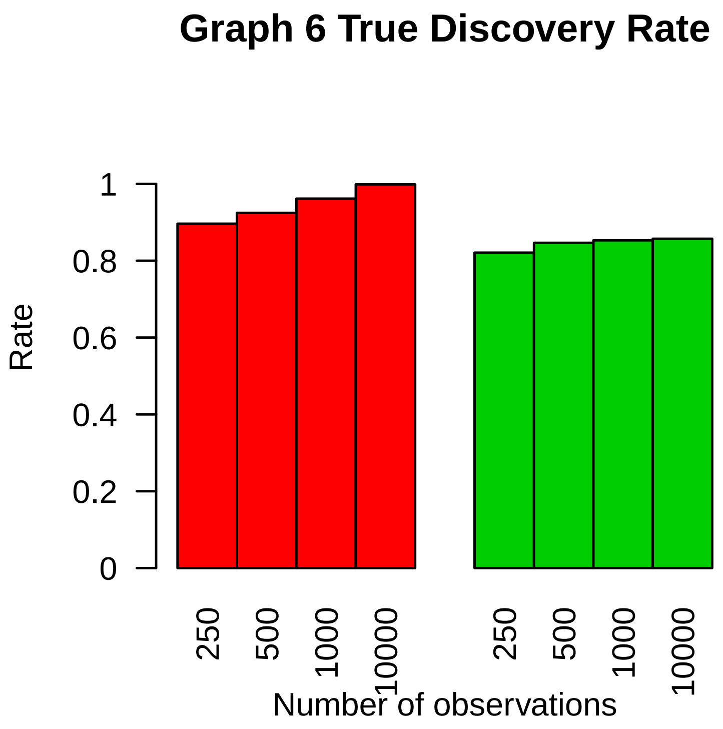

Graph 5 False Positive Rate
● detect.MIMIC
● Factor Analysis
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

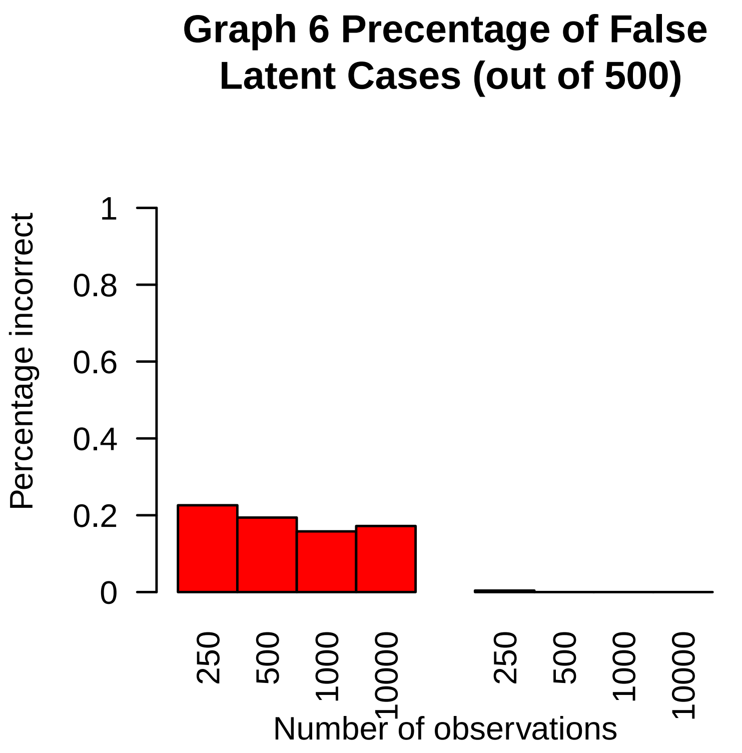

Graph 5 True Discovery Rate
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

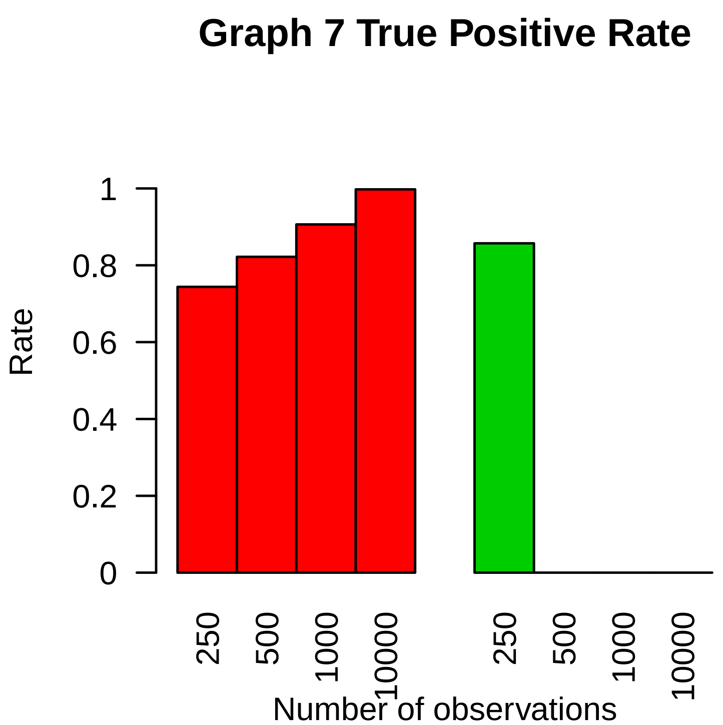

Graph 5 Precentage of False Latent Cases (out of 500)
Percentage incorrect
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

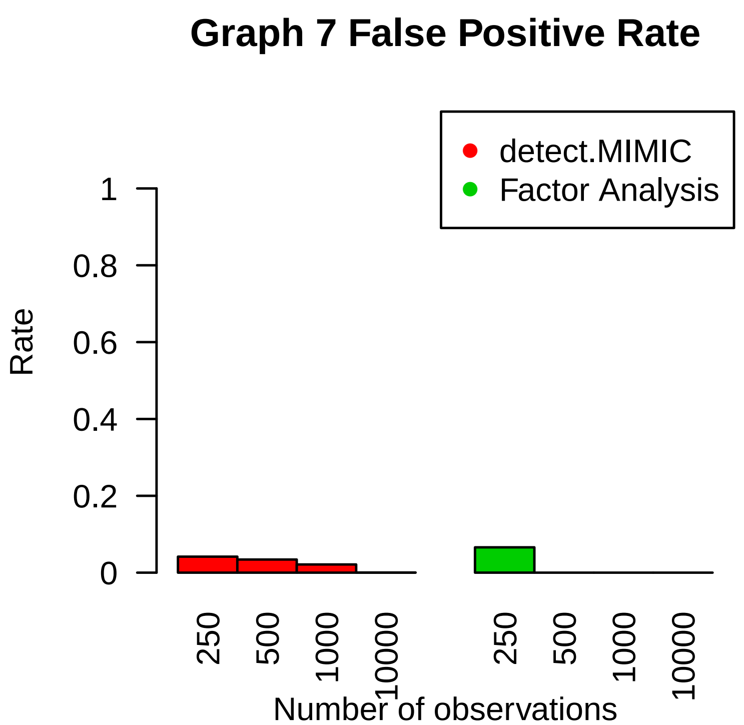

Graph 6 True Positive Rate
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

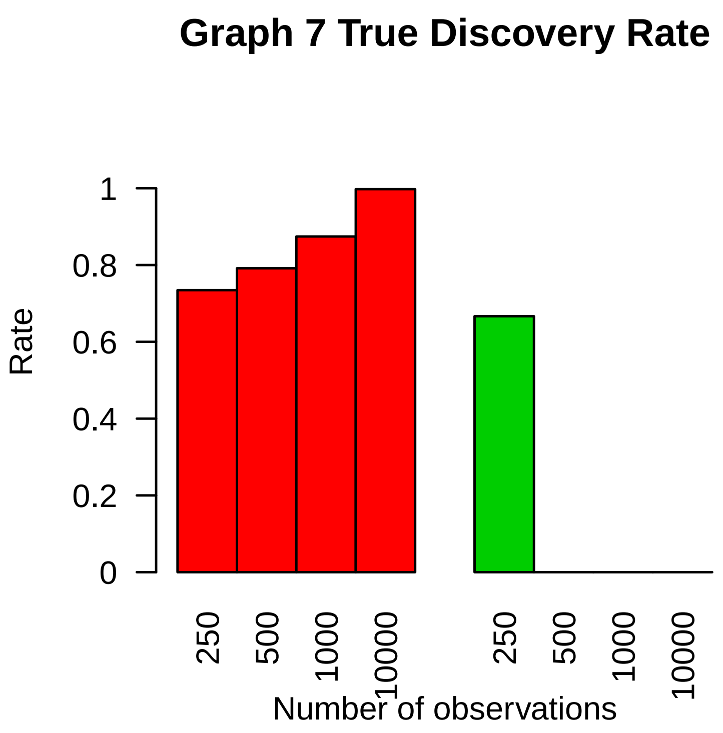

Graph 6 False Positive Rate
● detect.MIMIC
● Factor Analysis
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

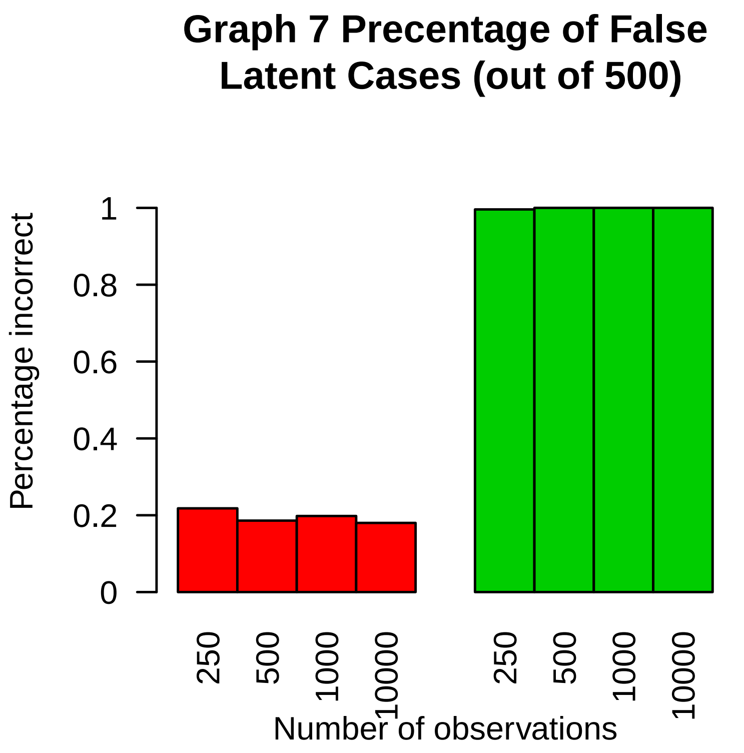

Graph 6 True Discovery Rate
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations


Graph 6 Precentage of False Latent Cases (out of 500)
Percentage incorrect
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations


Graph 7 True Positive Rate
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations


Graph 7 False Positive Rate
● detect.MIMIC
● Factor Analysis
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

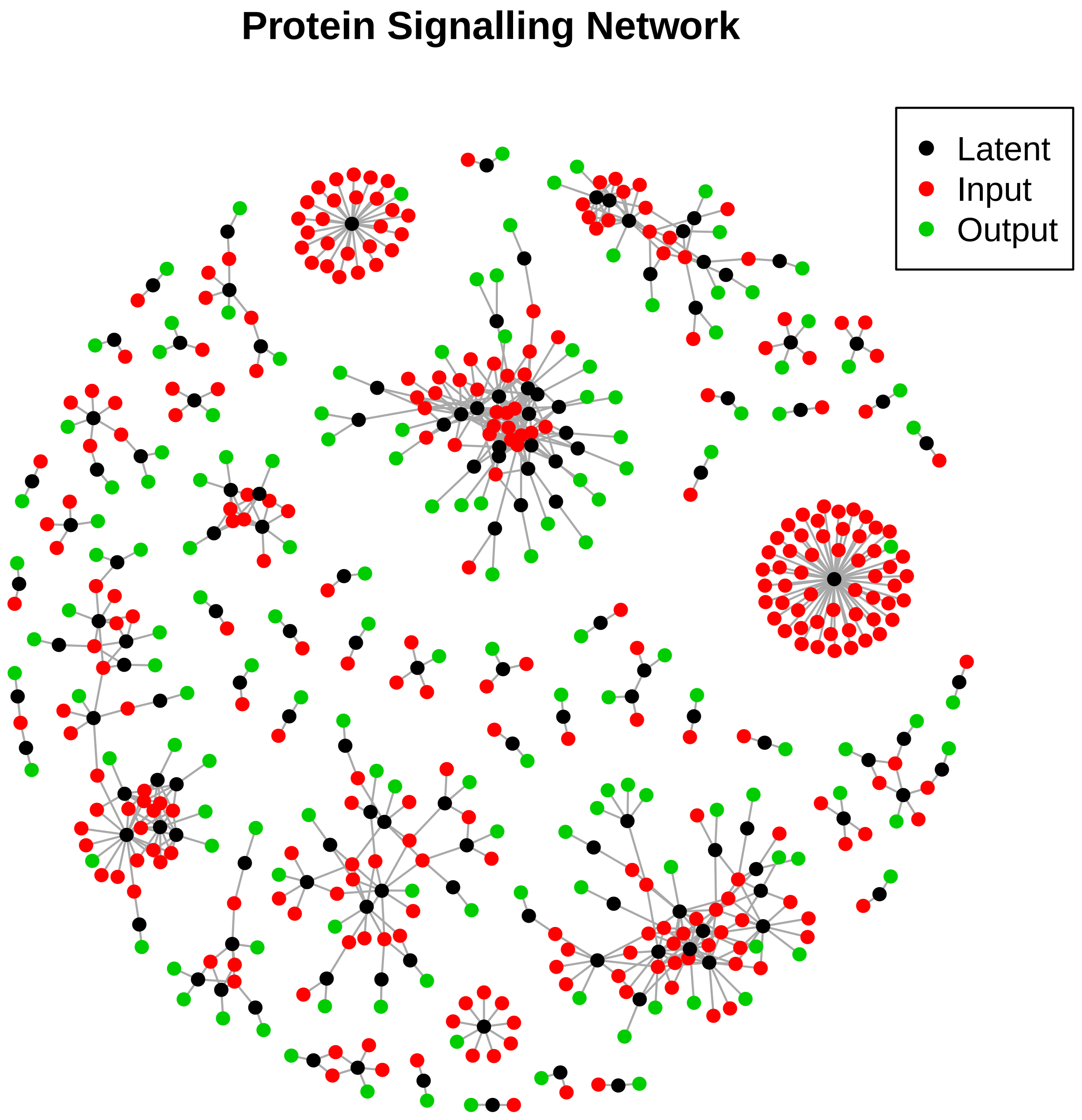

Graph 7 True Discovery Rate
Rate
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations


Graph 7 Precentage of False Latent Cases (out of 500)
Percentage incorrect
1
0.8
0.6
0.4
0.2
0
250
500
1000
10000
250
500
1000
10000
Number of observations

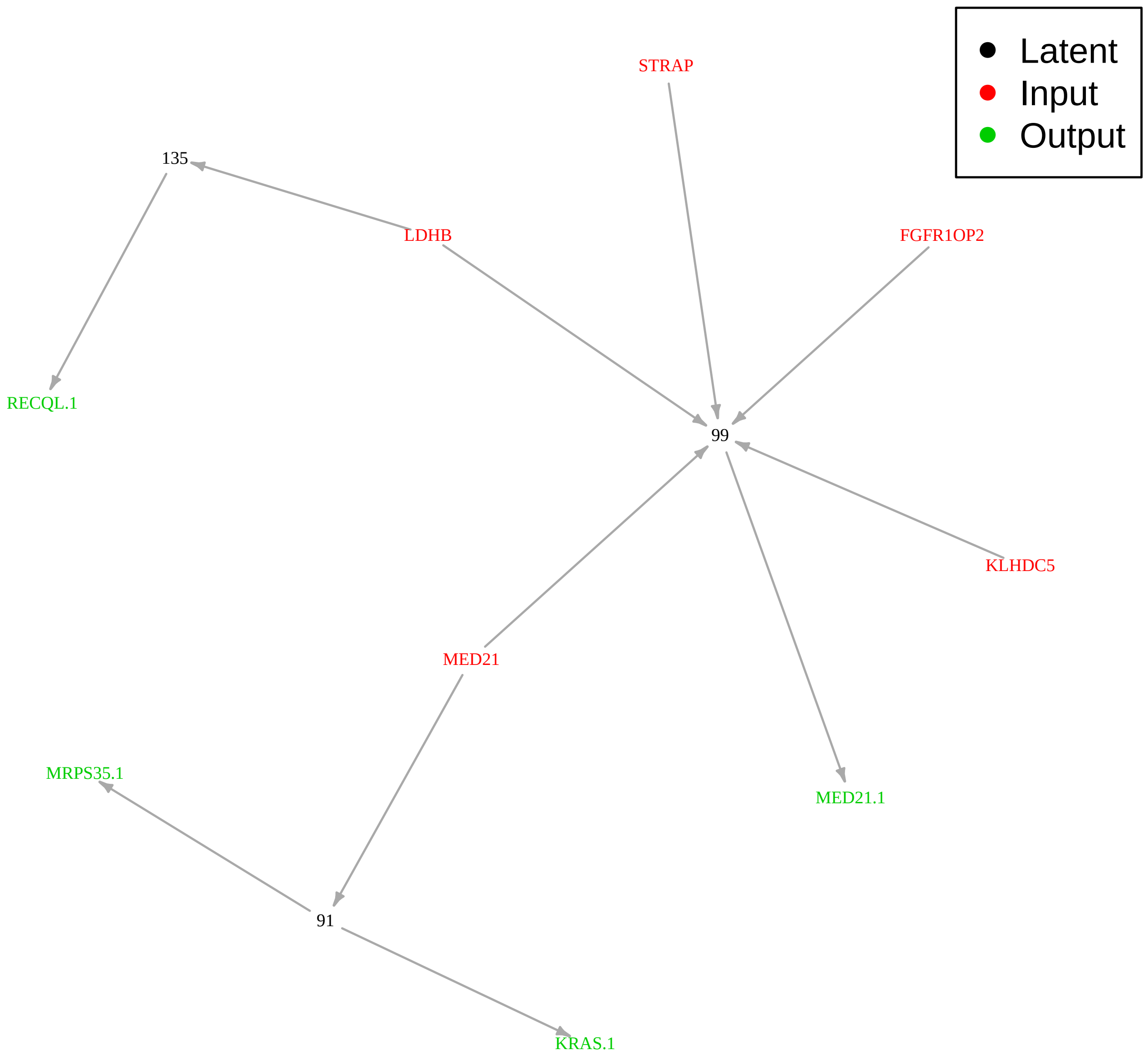
Appendix B

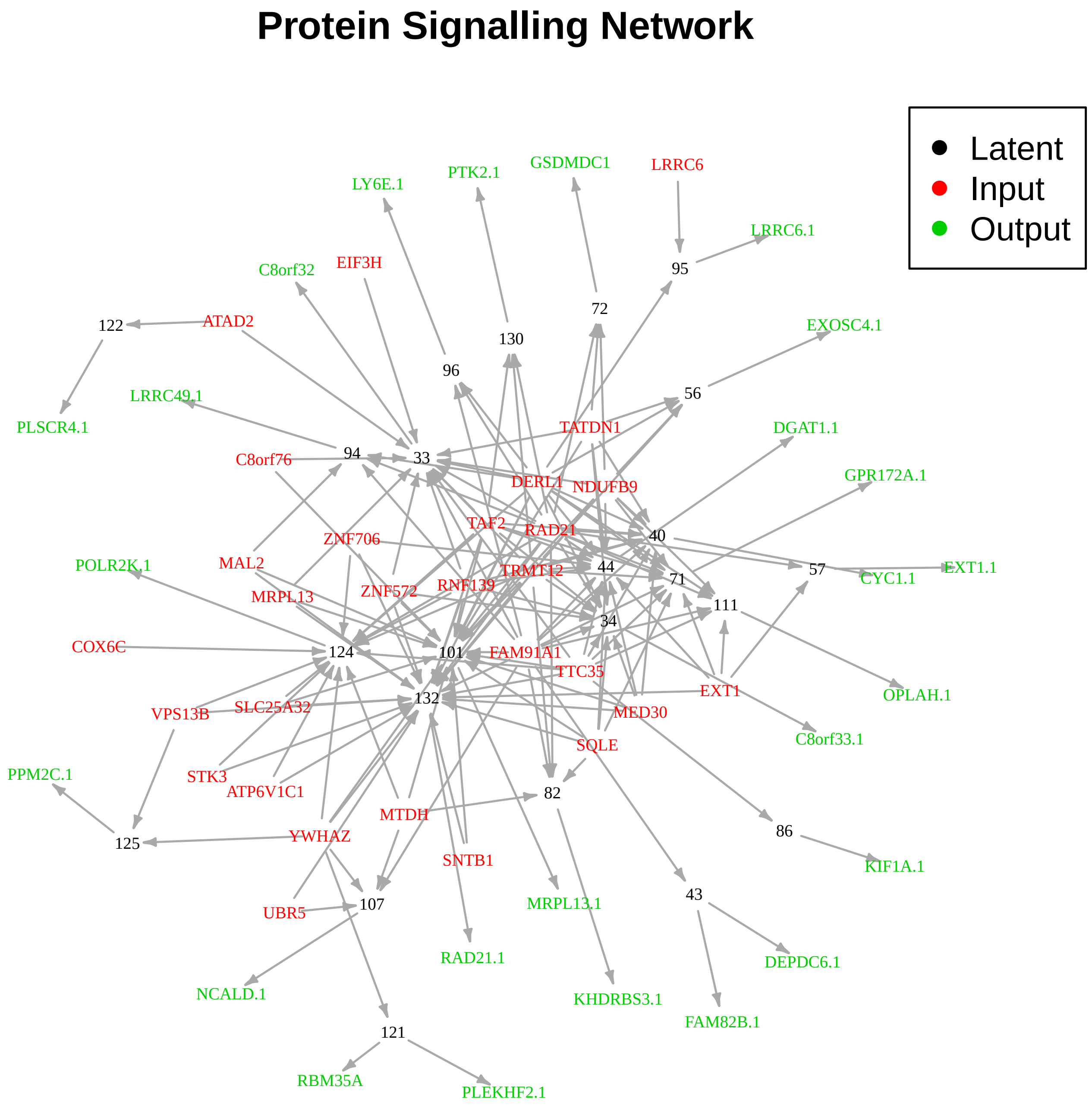
Data Analysis

This appendix contains both an enlarged version of the full protein signalling network, as well as enlarged versions of every subgraph with more than 3 nodes.

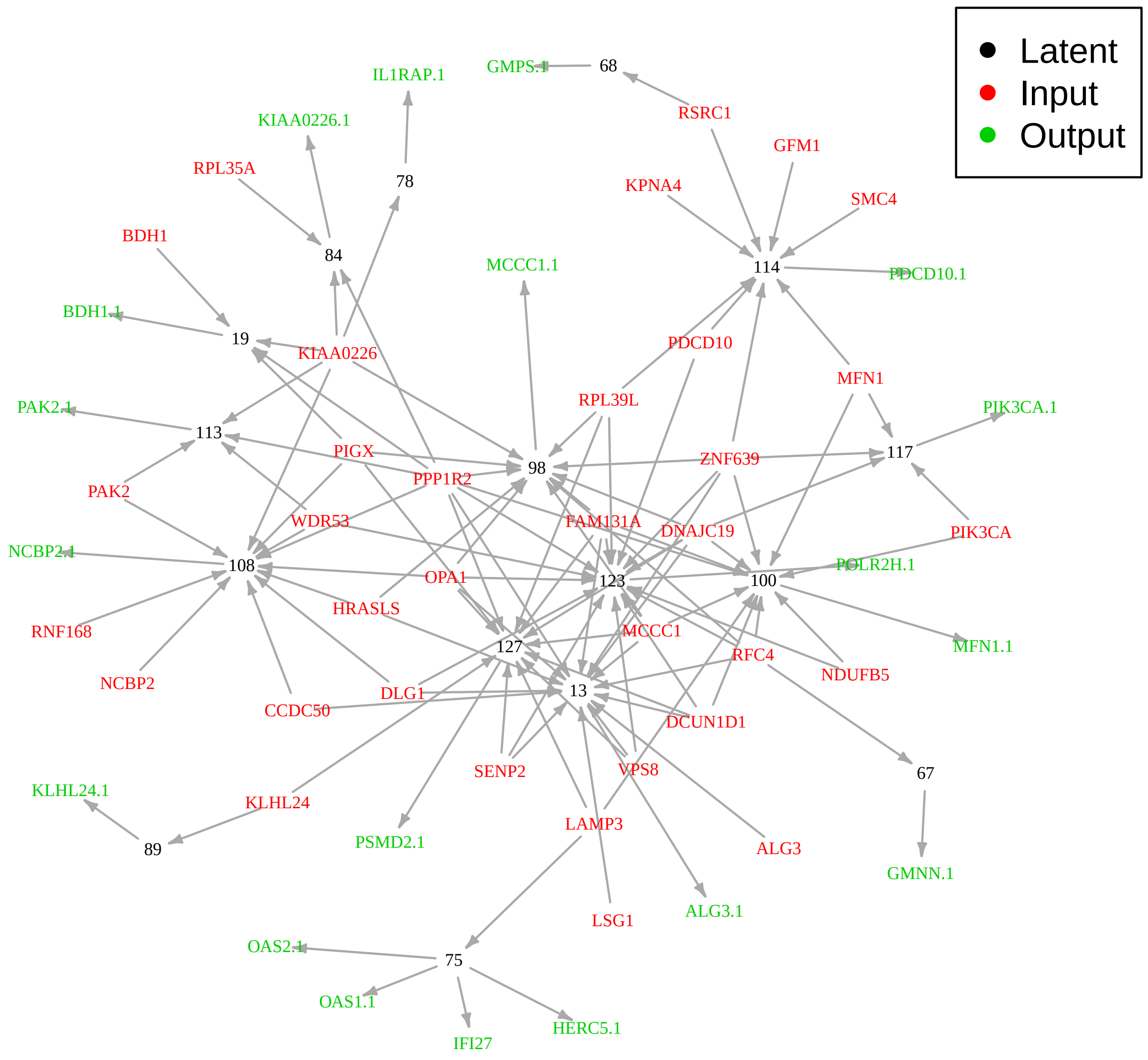

Protein Signalling Network
Latent
Input
Output

Protein Signalling Network

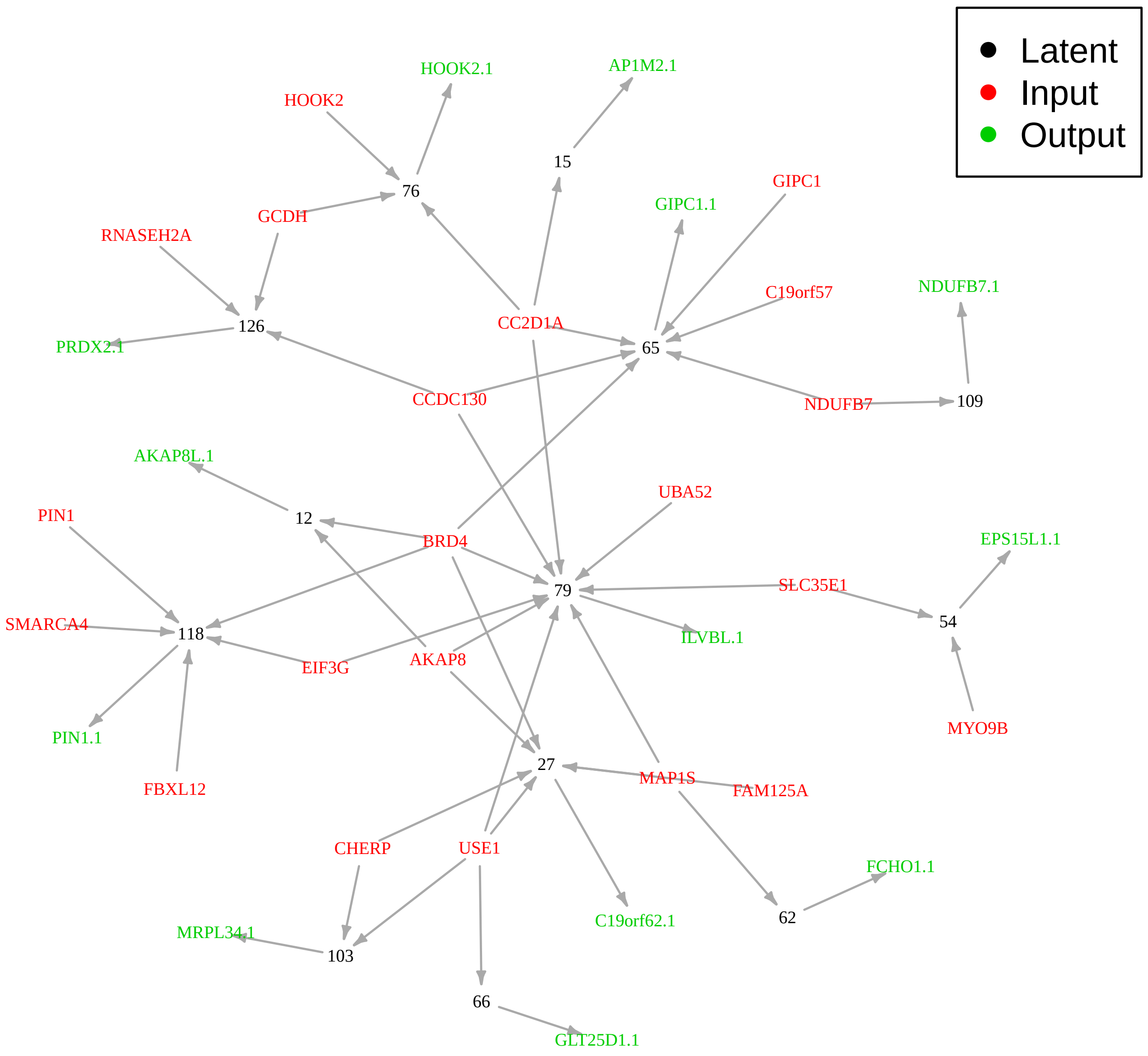



Protein Signalling Network
● Latent
● Input
● Output
LY6E.1
PTK2.1
GSDMDC1
LRRC6
LRRC6.1
95
72
130
96
C8orf32
EIF3H
122
ATAD2
EXOSC4.1
56
LRRC49.1
PLSCR4.1
TATDN1
DGAT1.1
C8orf76
94
33
DERL1
NDUFB9
GPR172A.1
TAF2
RAD21
40
ZNF706
57
EXT1.1
CYC1.1
MAL2
TRMT12
44
POLR2K.1
ZNF572
RNF139
71
MRPL13
111
34
COX6C
124
101
FAM91A1
TTC35
OPLAH.1
EXT1
SLC25A32
132
MED30
VPS13B
C8orf33.1
SQLE
PPM2C.1
STK3
ATP6V1C1
82
MTDH
86
125
YWHAZ
KIF1A.1
SNTB1
43
UBR5
107
MRPL13.1
DEPDC6.1
RAD21.1
NCALD.1
KHDRBS3.1
FAM82B.1
121
RBM35A
PLEKHF2.1

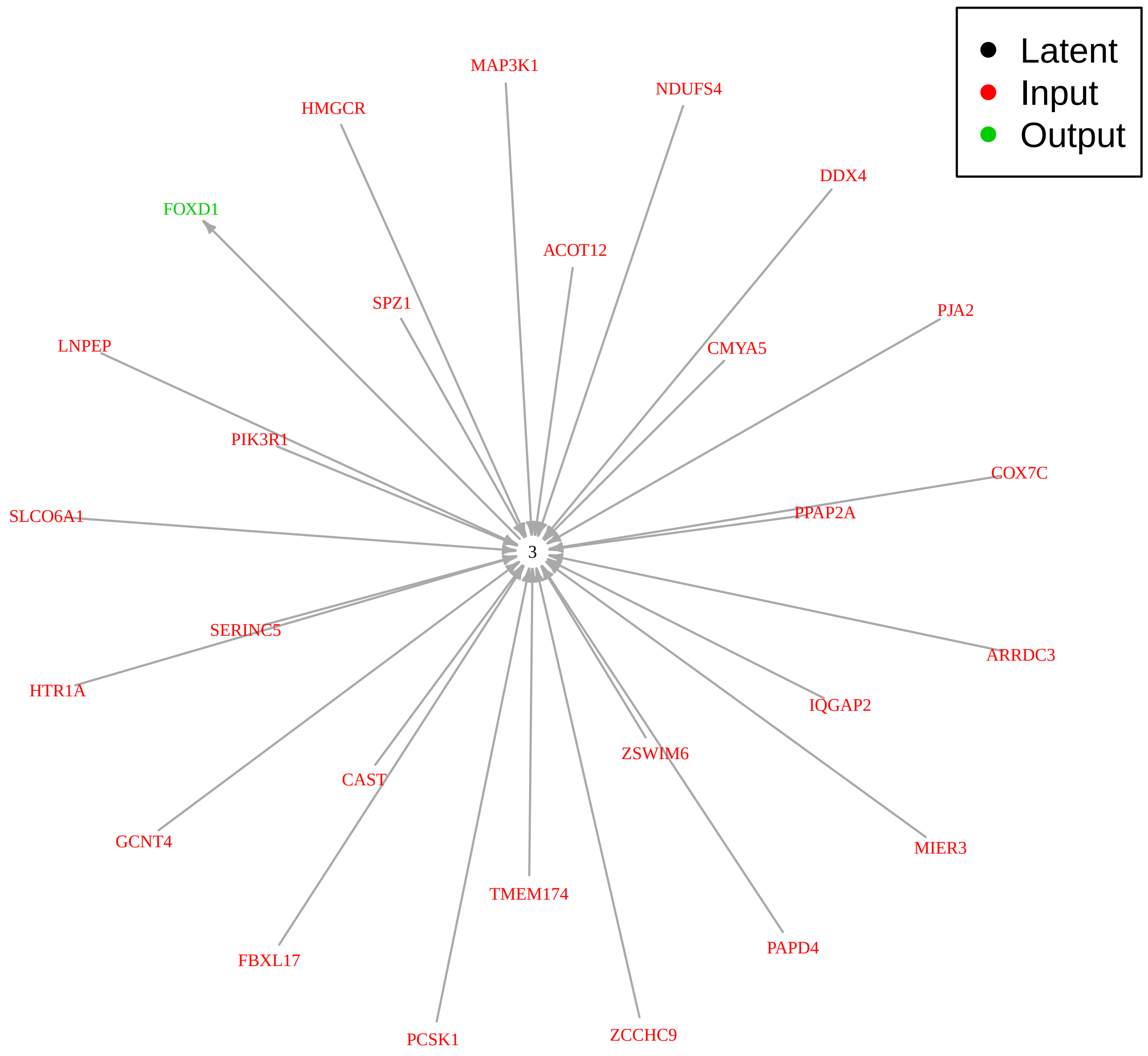
Protein Signalling Network

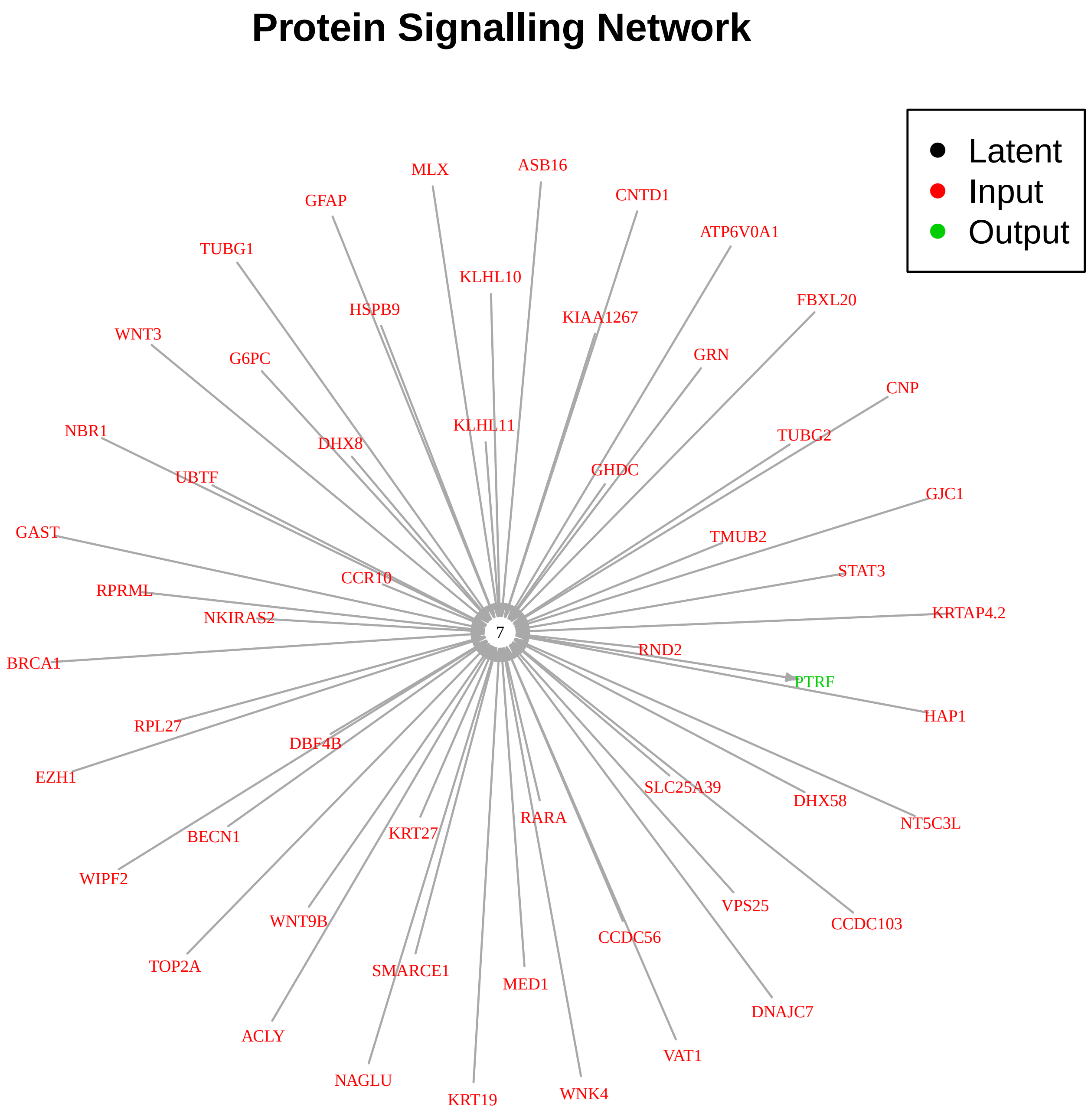


Protein Signalling Network

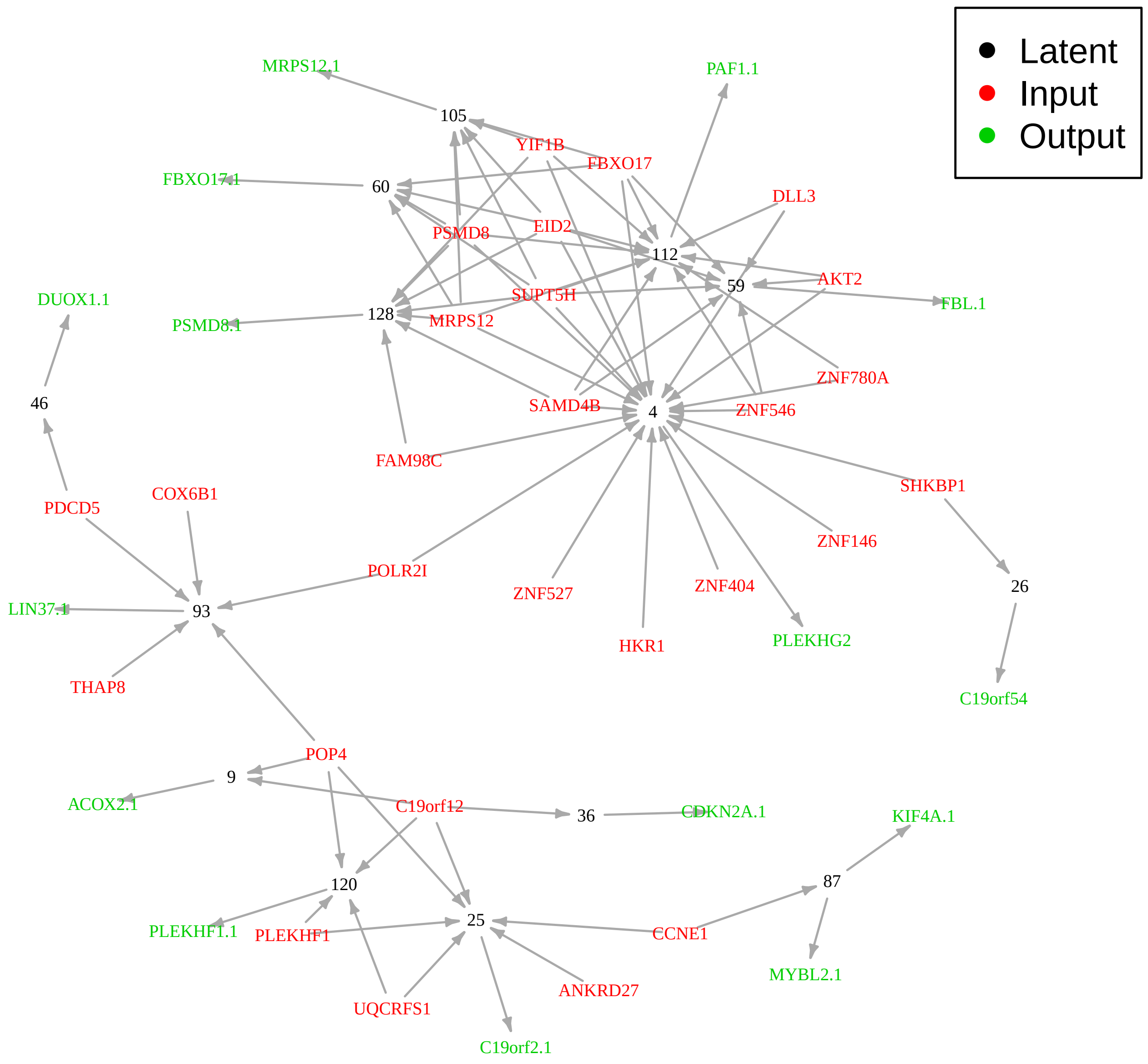


Protein Signalling Network

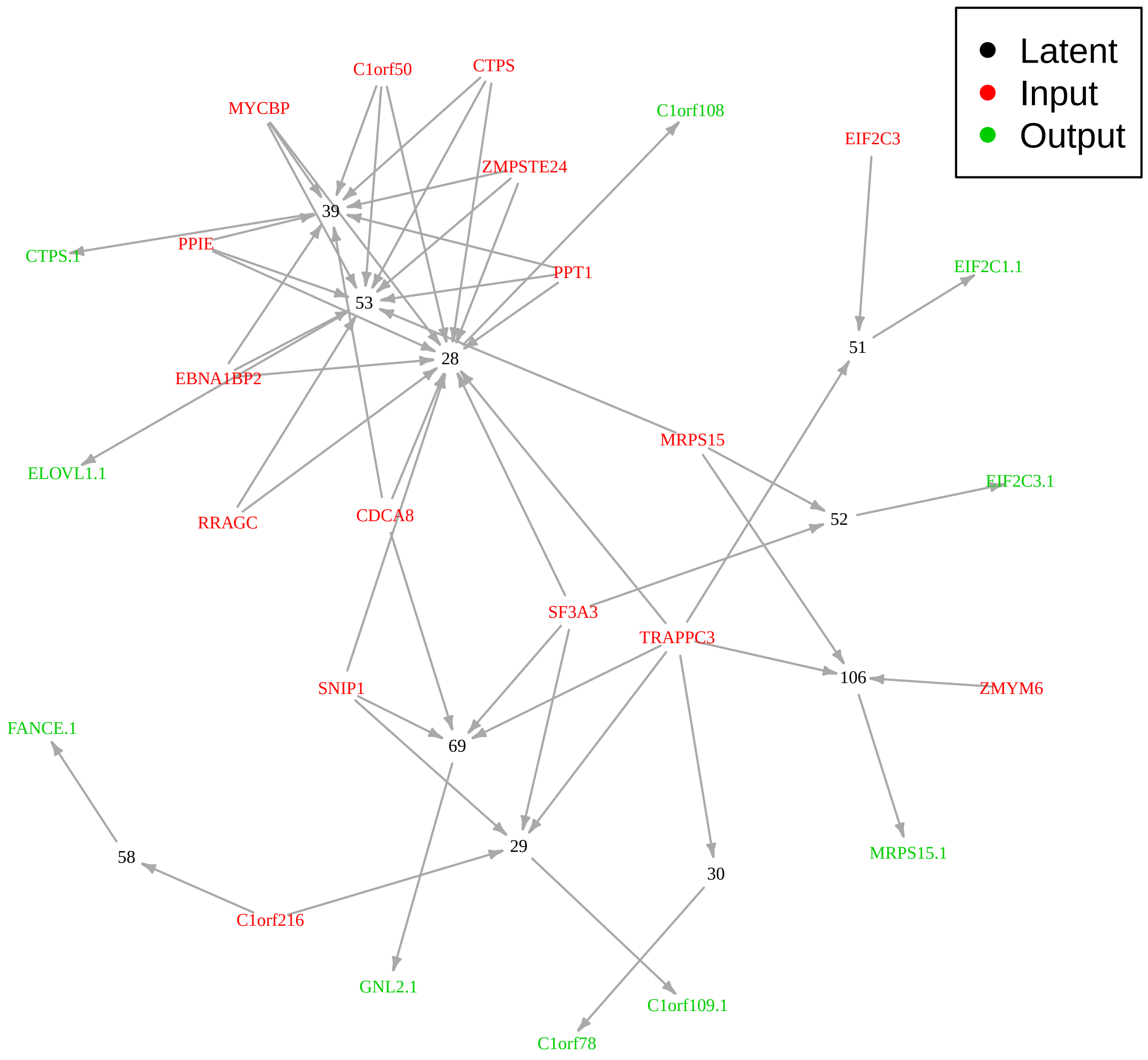


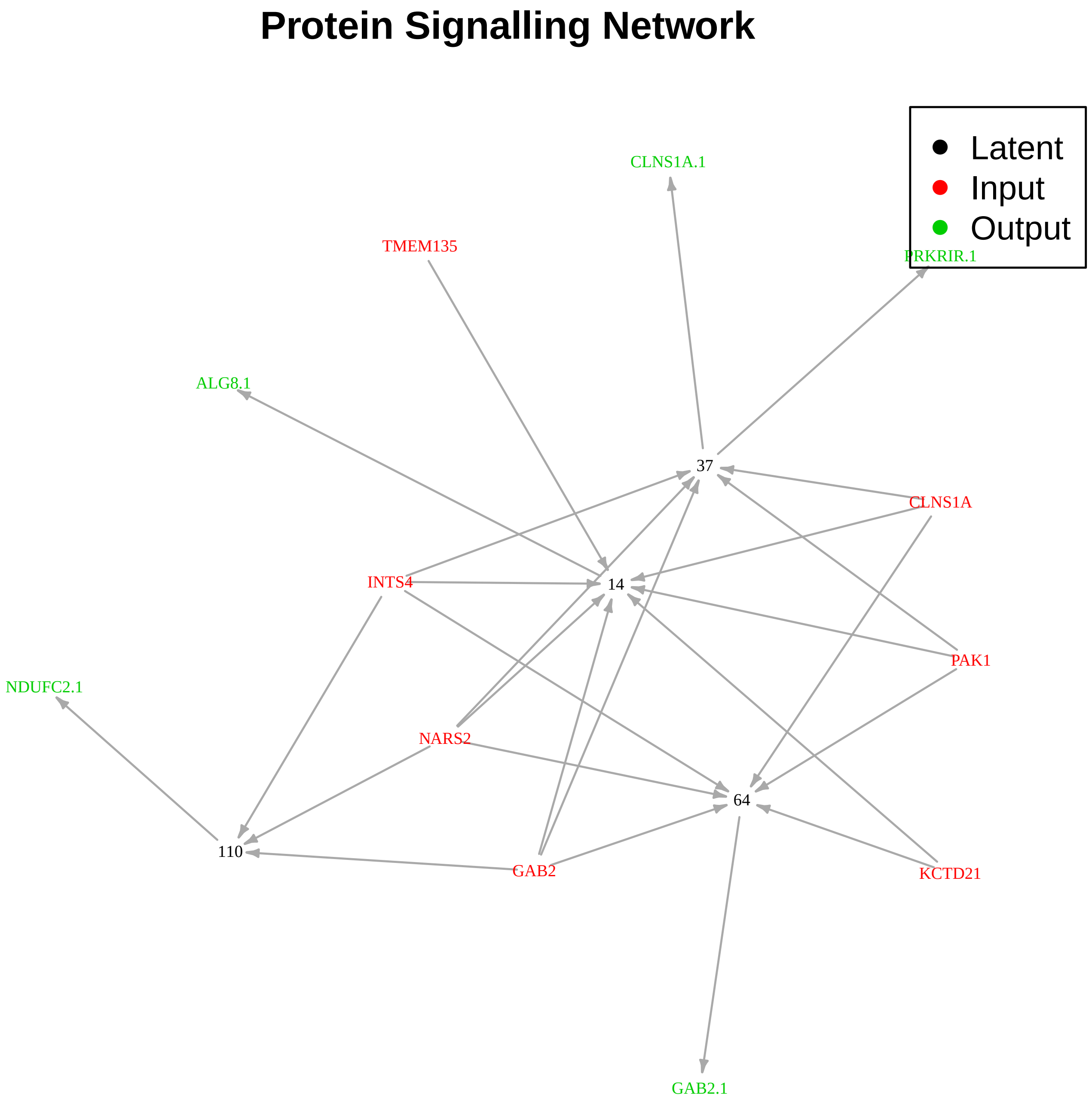

Protein Signalling Network
● Latent
● Input
● Output
MLX
ASB16
GFAP
CNTD1
ATP6V0A1
TUBG1
KLHL10
HSPB9
KIAA1267
FBXL20
WNT3
G6PC
GRN
CNP
NBR1
KLHL11
DHX8
TUBG2
UBTF
GHDC
GJC1
GAST
TMUB2
STAT3
RPRML
CCR10
NKIRAS2
KRTAP4.2
7
BRCA1
RND2
PTRF
HAP1
RPL27
DBF4B
EZH1
SLC25A39
DHX58
NT5C3L
BECN1
KRT27
RARA
WIPF2
VPS25
CCDC103
WNT9B
CCDC56
TOP2A
SMARCE1
MED1
DNAJC7
ACLY
VAT1
NAGLU
KRT19
WNK4

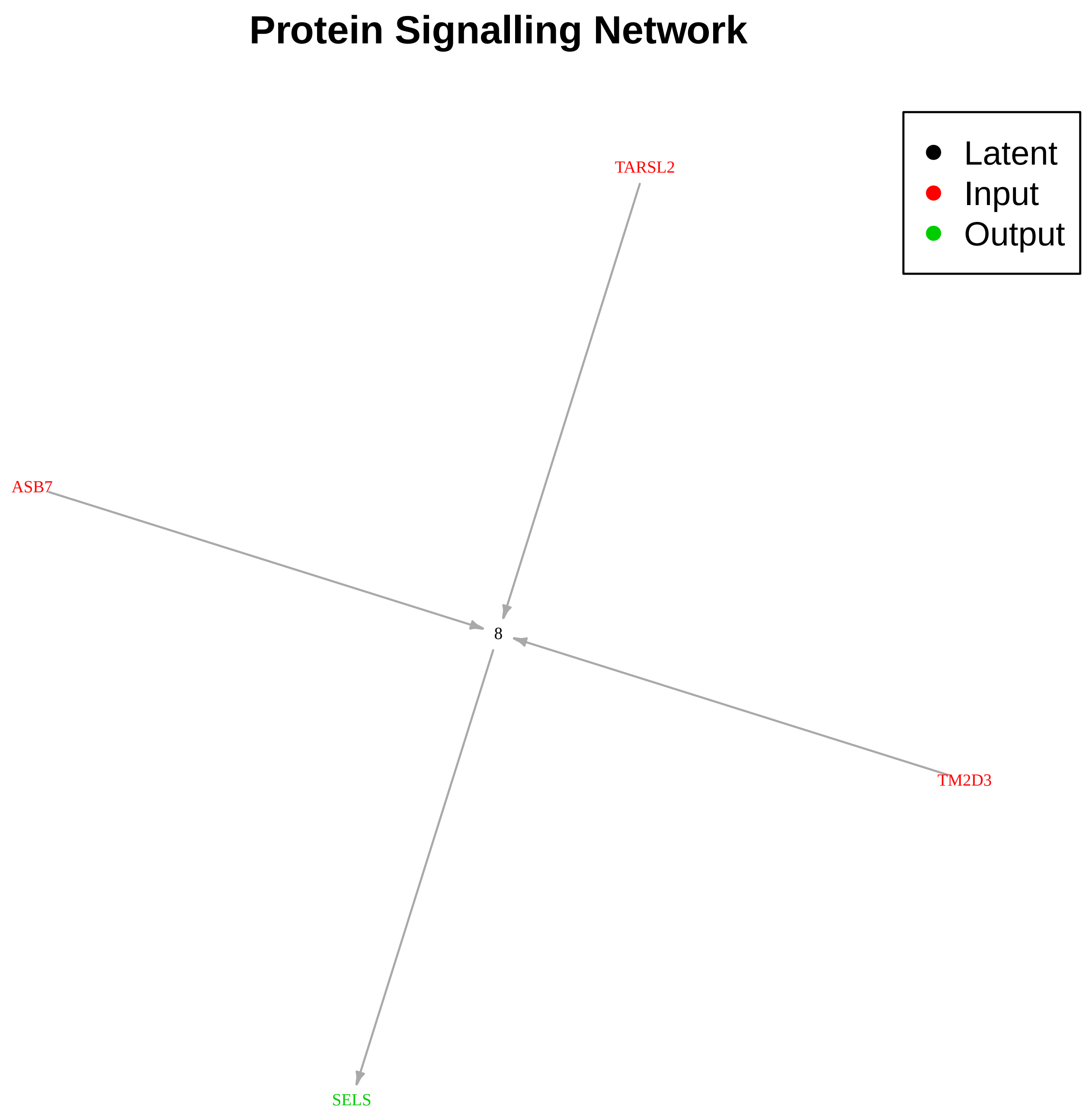
Protein Signalling Network



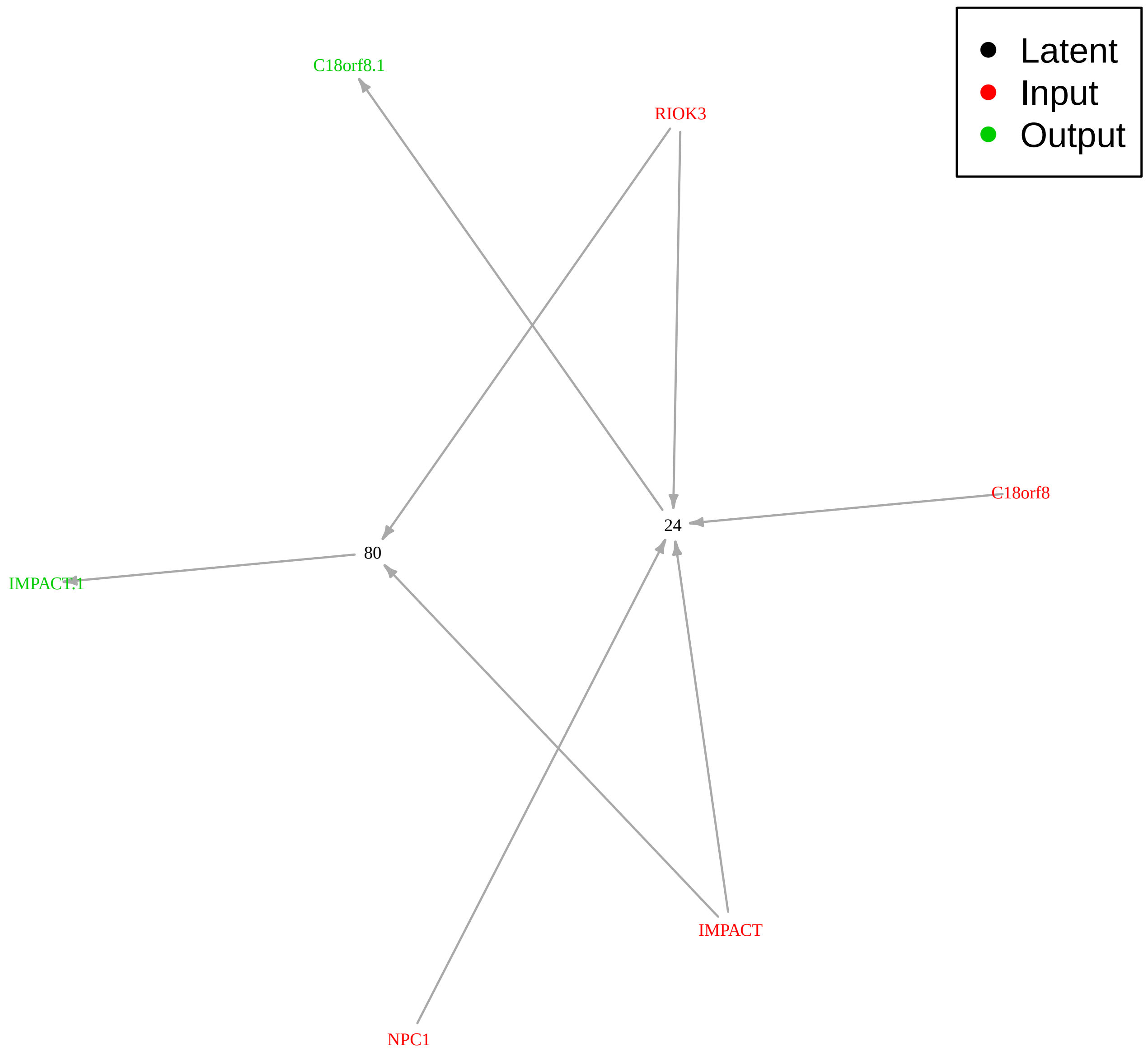
Protein Signalling Network



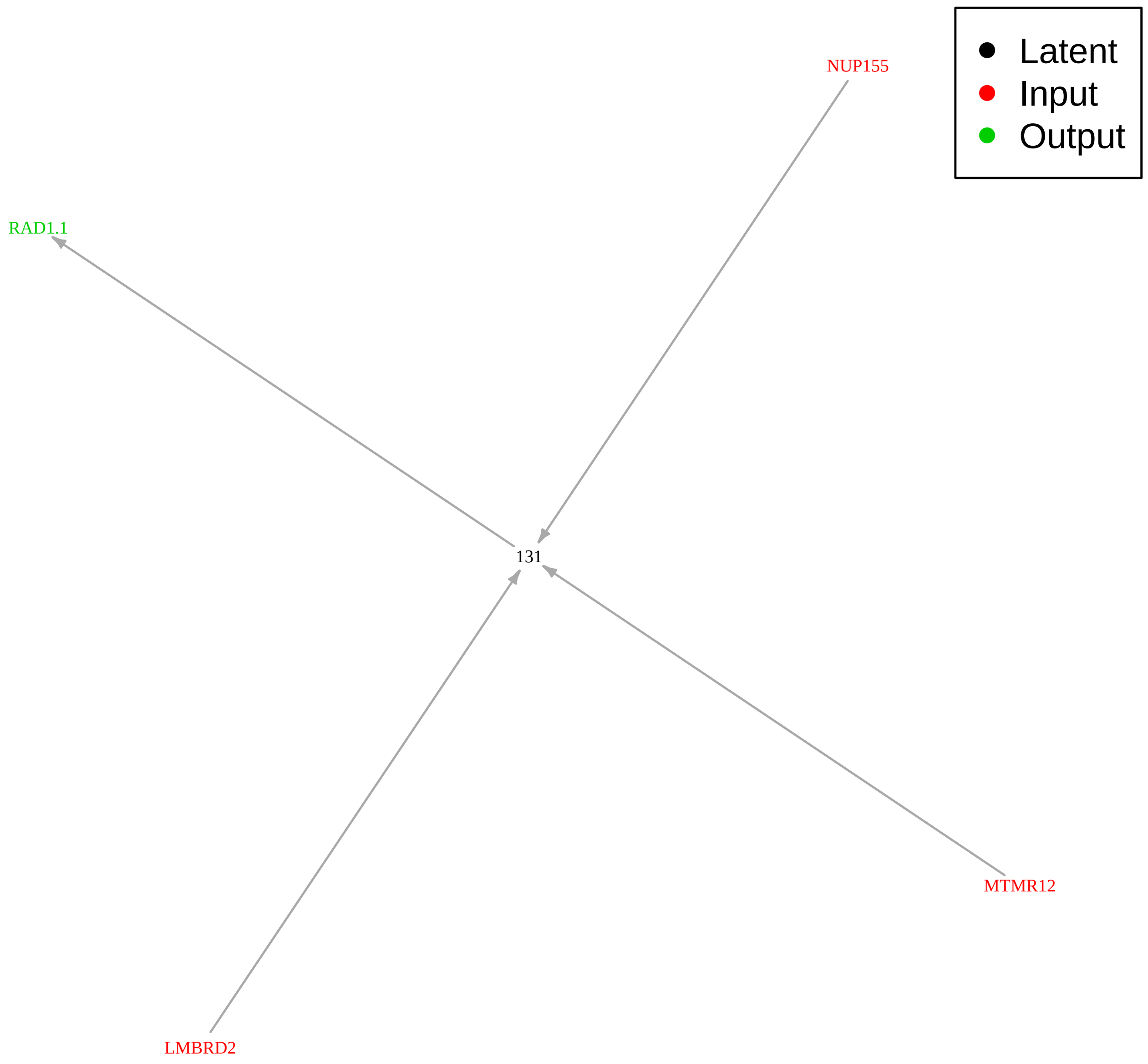

Protein Signalling Network
● Latent
● Input
● Output
CLNS1A.1
TMEM135
PRKRIR.1
ALG8.1
37
CLNS1A
INTS4
14
PAK1
NDUFC2.1
NARS2
64
110
GAB2
KCTD21
GAB2.1


Protein Signalling Network
● Latent
● Input
● Output
TARSL2
ASB7
8
TM2D3
SELS

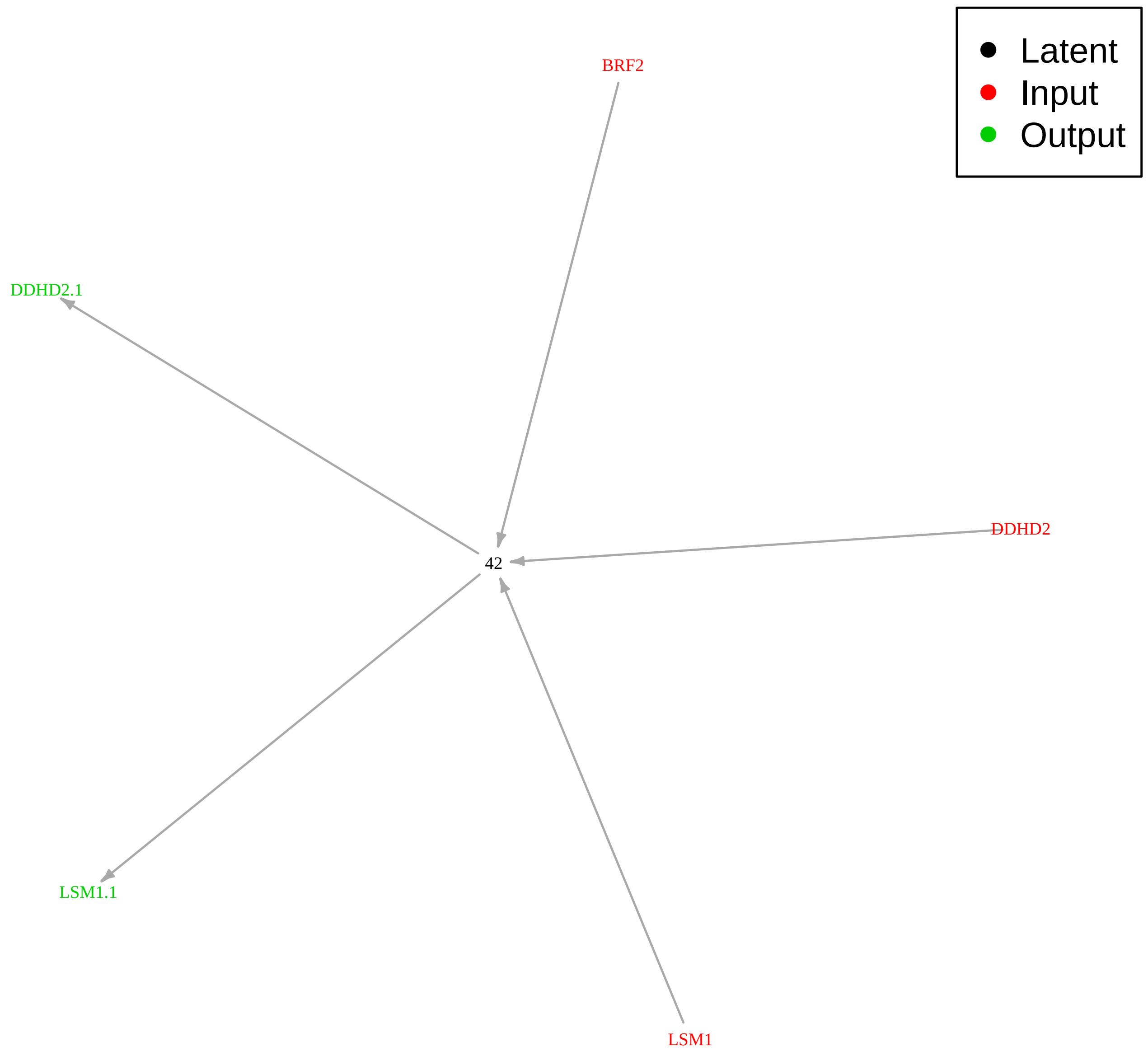
Protein Signalling Network

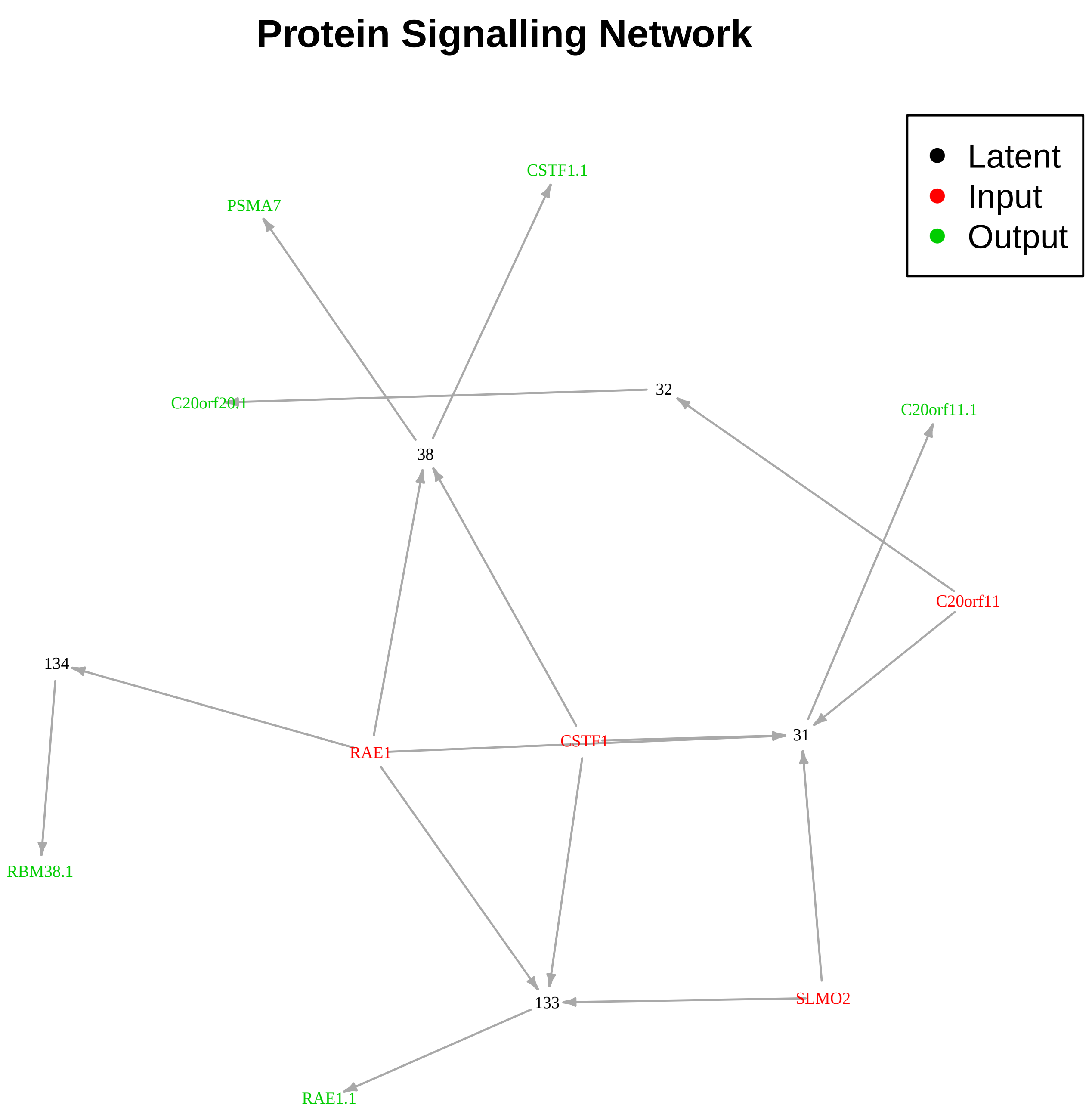


Protein Signalling Network

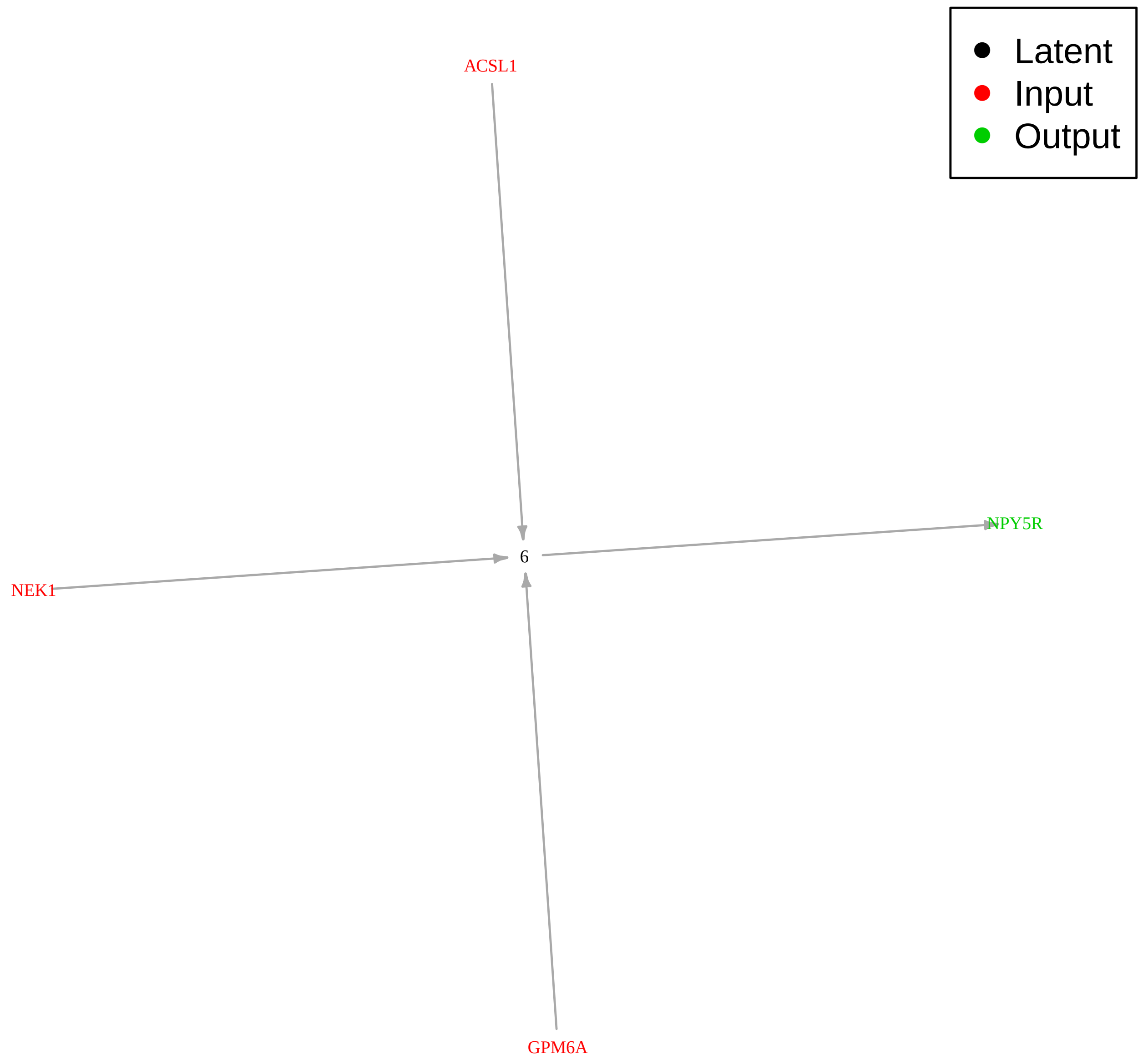


Protein Signalling Network

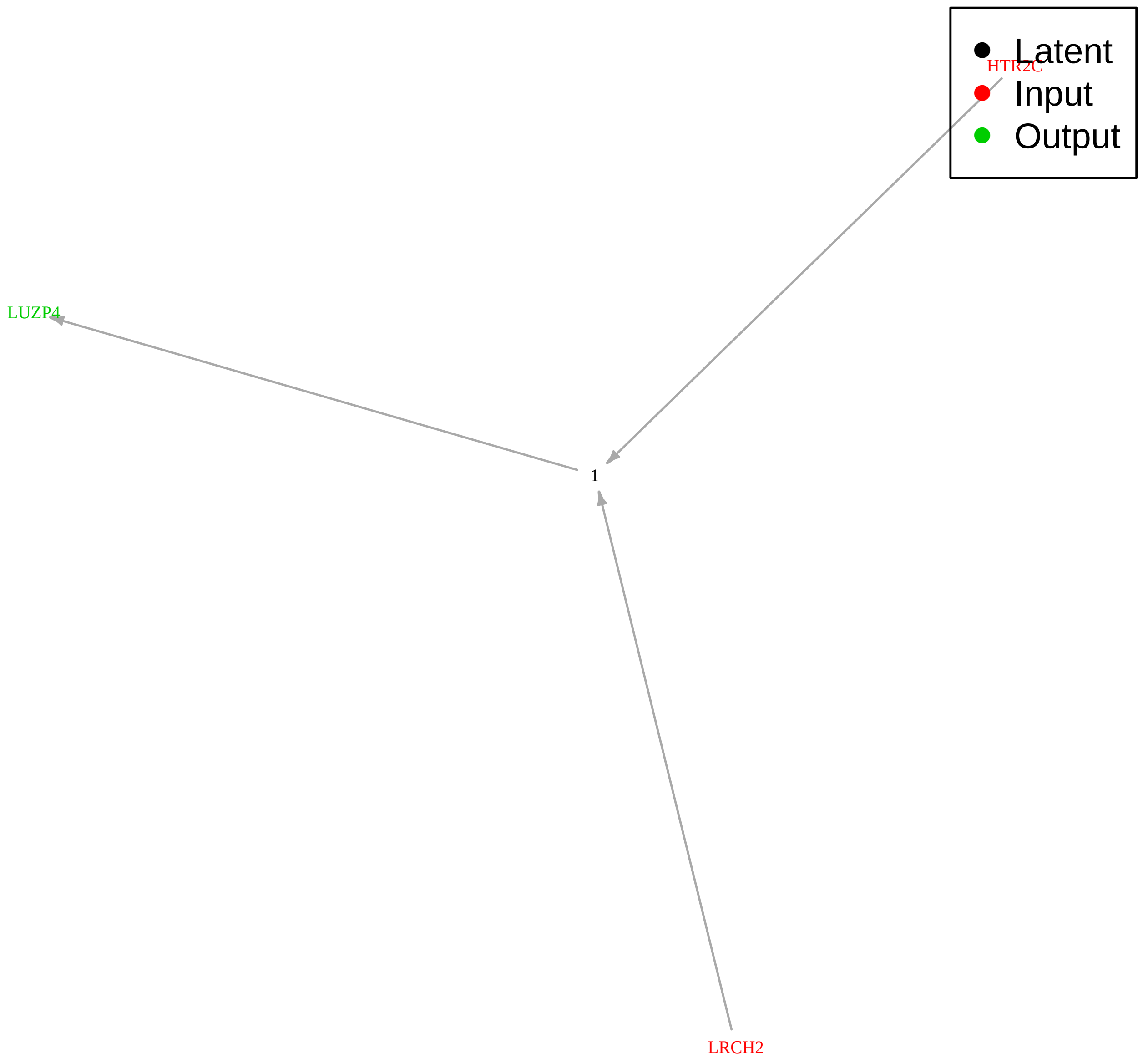


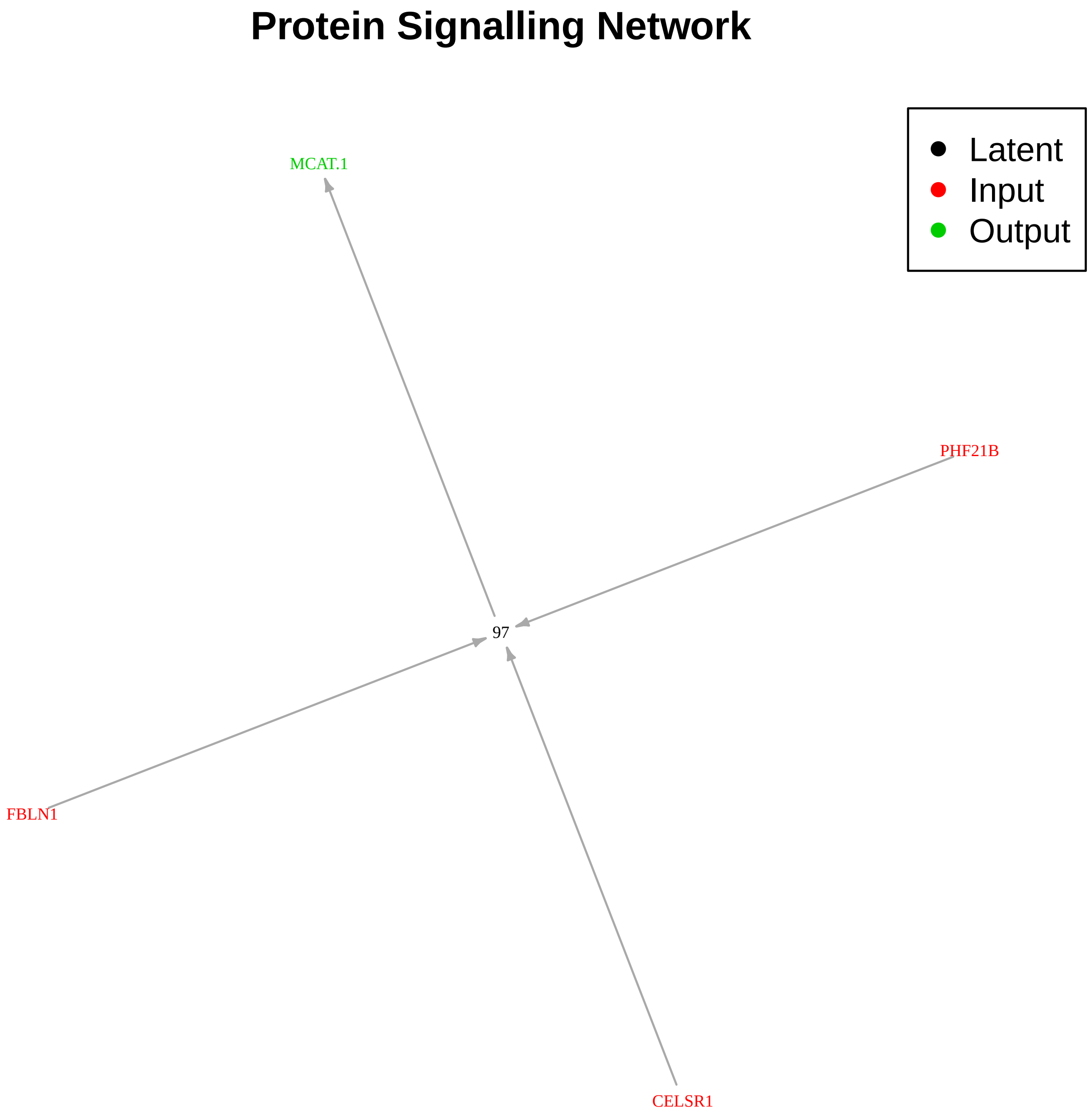

Protein Signalling Network
● Latent
● Input
● Output
CSTF1.1
PSMA7
32
C20orf20.1
C20orf11.1
38
C20orf11
134
RAE1
CSTF1
31
RBM38.1
133
SLMO2
RAE1.1

Protein Signalling Network

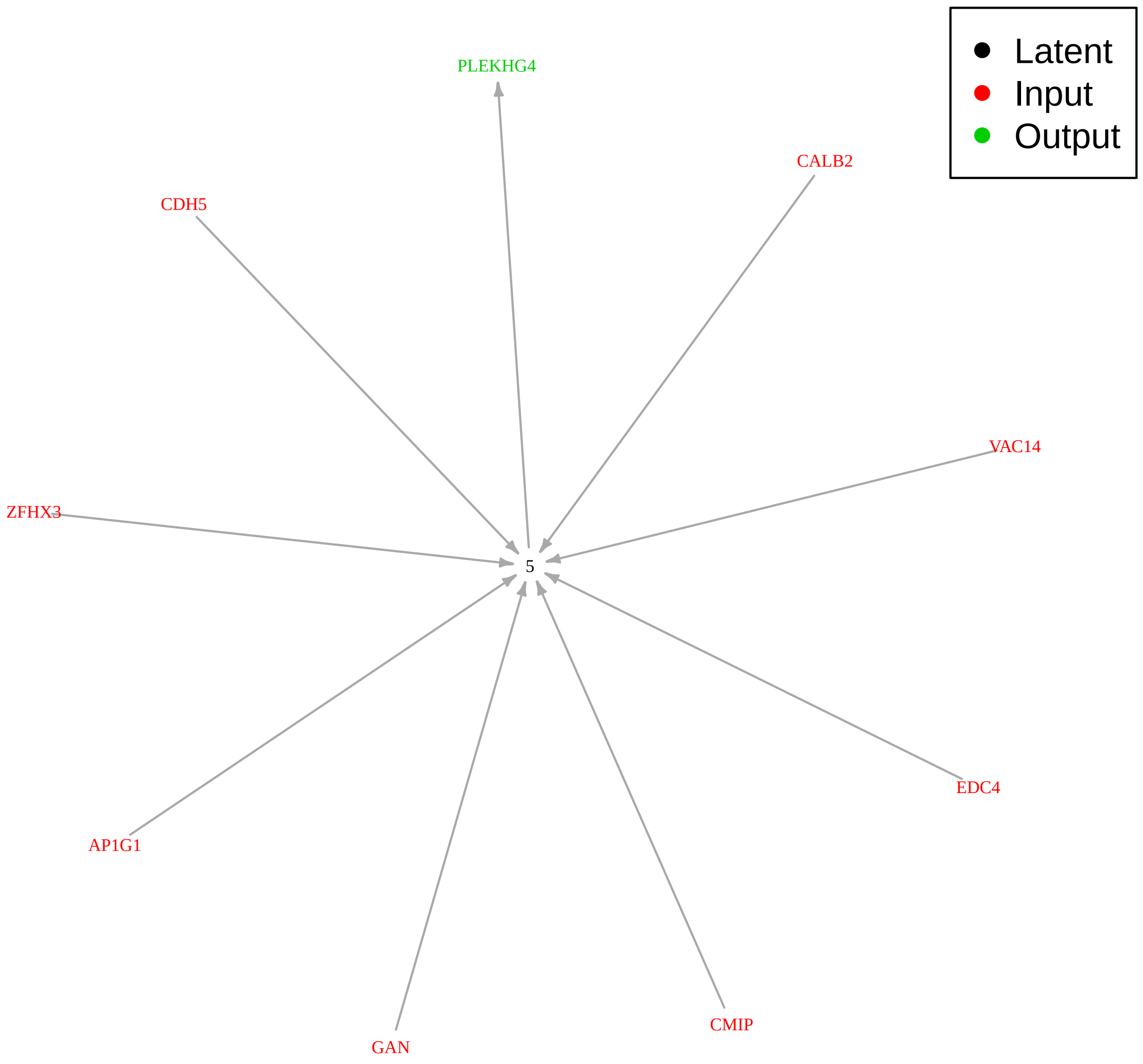


Protein Signalling Network

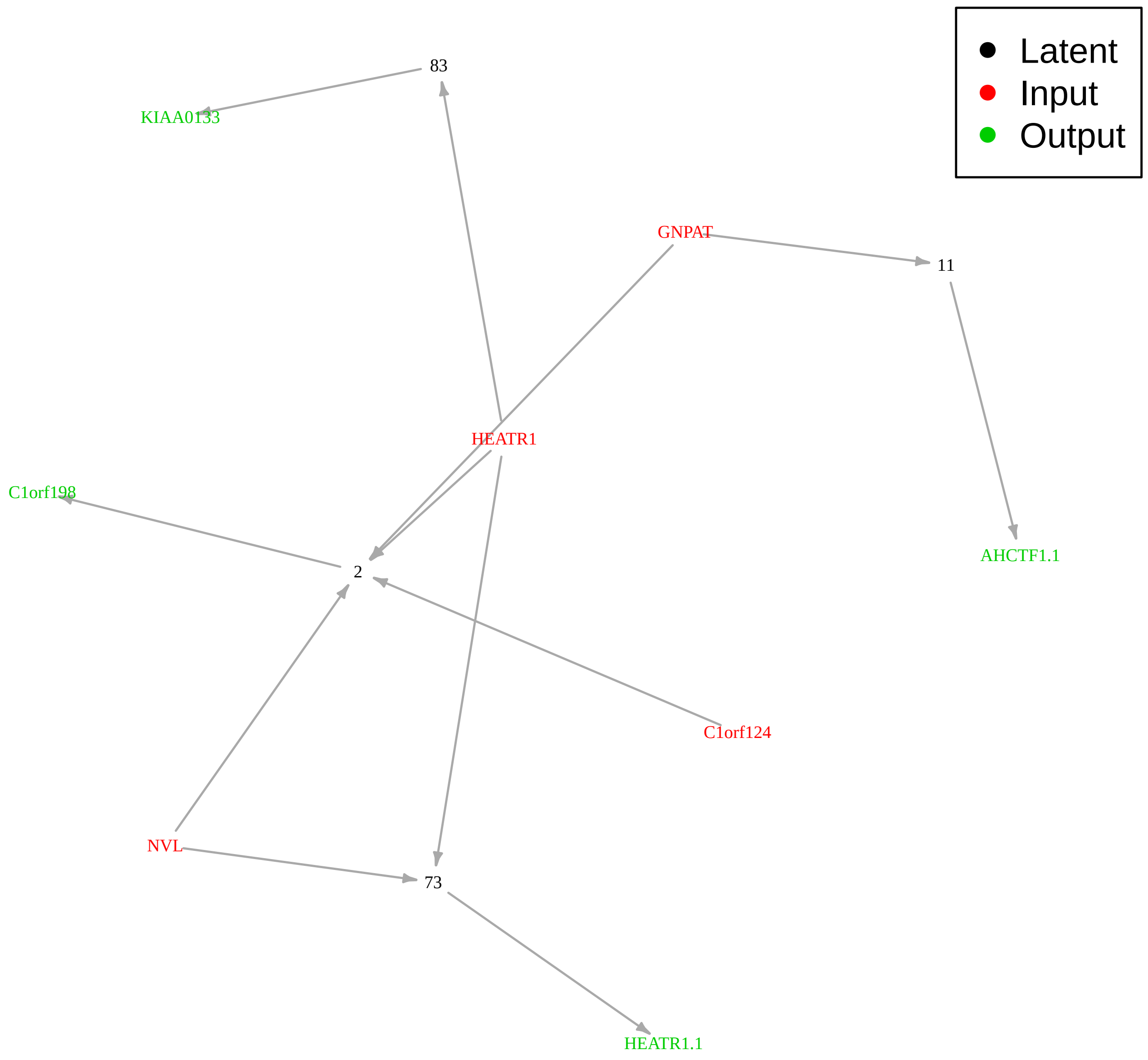



Protein Signalling Network
Latent
Input
Output
MCAT.1
PHF21B
97
FBLN1
CELSR1

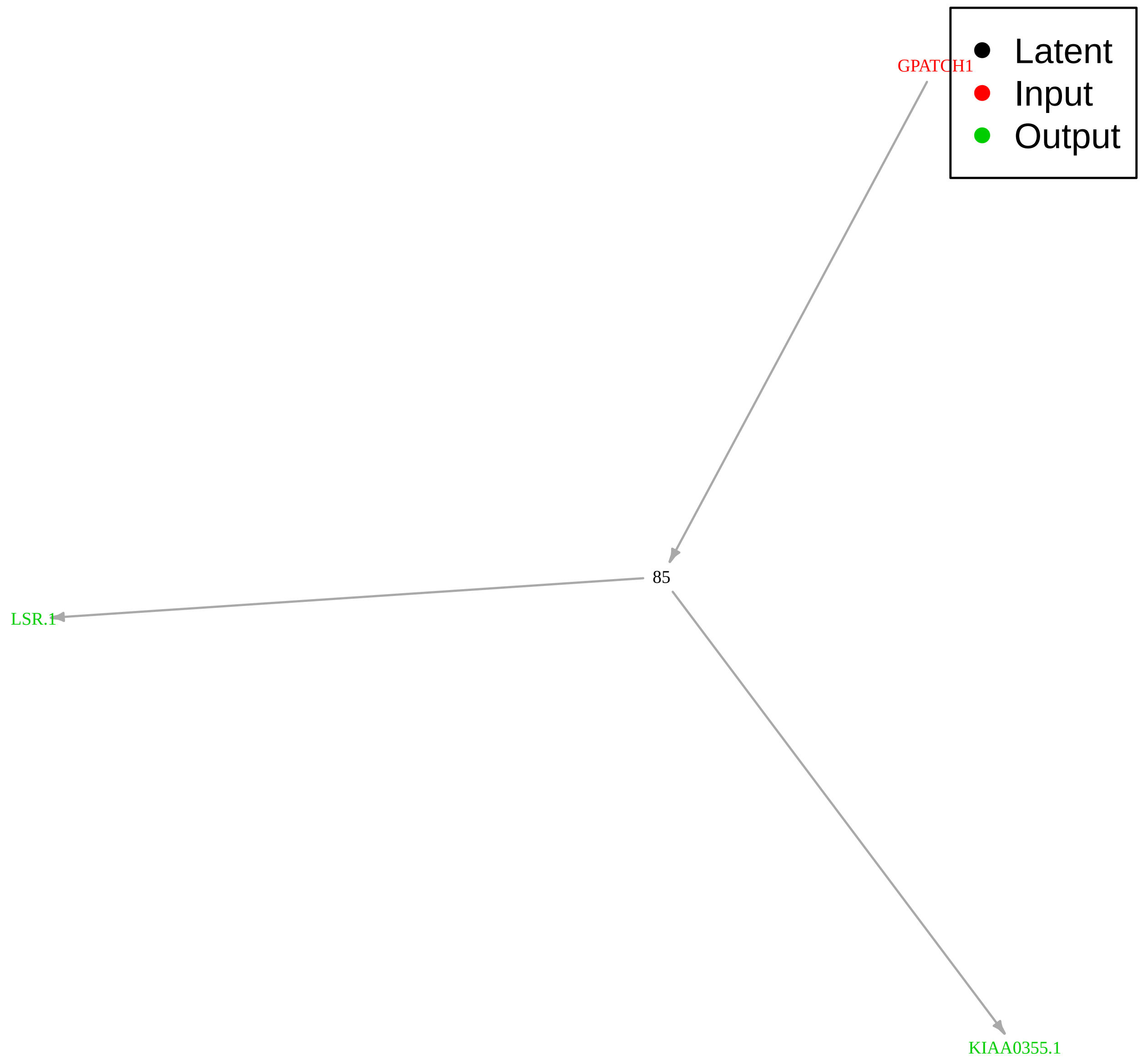
Protein Signalling Network



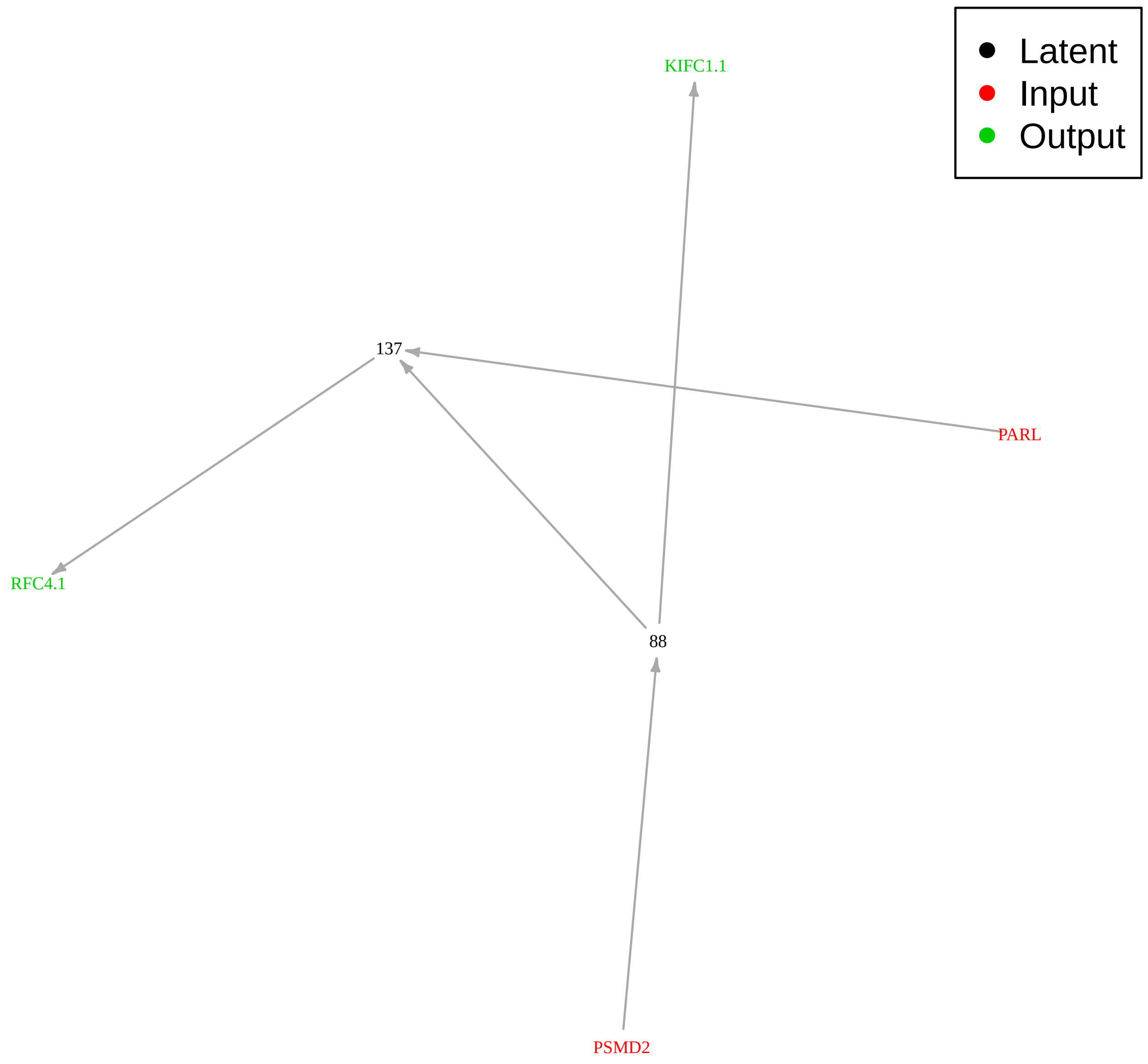
Protein Signalling Network



Protein Signalling Network



Protein Signalling Network



Appendix C

Souce Code

This appendix contains the source code used when performing both the simulations and the data analysis.

```r
require(pcalg)
require(igraph)
require(gRim)
require(plotrix)
require(nFactors)



source("construct_graph.R")
source("data_commands.R")
source("simple_simulations.R")
source("simulation_scoring.R")


# Currently, this portion is not in a function so as to avoid rerunning
# everything in the event of an error (i.e., an error will not interrupt the
# function call, leading to already completed simulations being discarded).
# Eventually, a function will be written so that this is cleaner.

# Runs the simulations.
graph1.10000 <- replicate(test.both.methods("sim.graph.1.r.txt",
 sample.size=10000), n=500)
graph2.10000 <- replicate(test.both.methods("sim.graph.2.r.txt",
 sample.size=10000), n=500)
graph3.10000 <- replicate(test.both.methods("sim.graph.3.r.txt",
 sample.size=10000), n=500)
graph4.10000 <- replicate(test.both.methods("sim.graph.4.r.txt",
 sample.size=10000), n=500)
graph5.10000 <- replicate(test.both.methods("sim.graph.5.r.txt",
 sample.size=10000), n=500)
graph6.10000 <- replicate(test.both.methods("sim.graph.6.r.txt",
 sample.size=10000), n=500)
graph7.10000 <- replicate(test.both.methods("sim.graph.7.r.txt",
 sample.size=10000), n=500)


graph1.1000 <- replicate(test.both.methods("sim.graph.1.r.txt",
 sample.size=1000), n=500)
graph2.1000 <- replicate(test.both.methods("sim.graph.2.r.txt",
 sample.size=1000), n=500)
graph3.1000 <- replicate(test.both.methods("sim.graph.3.r.txt",
 sample.size=1000), n=500)
graph4.1000 <- replicate(test.both.methods("sim.graph.4.r.txt",
 sample.size=1000), n=500)
graph5.1000 <- replicate(test.both.methods("sim.graph.5.r.txt",
 sample.size=1000), n=500)
graph6.1000 <- replicate(test.both.methods("sim.graph.6.r.txt",
 sample.size=1000), n=500)
graph7.1000 <- replicate(test.both.methods("sim.graph.7.r.txt",
 sample.size=1000), n=500)

graph1.500 <- replicate(test.both.methods("sim.graph.1.r.txt",
 sample.size=500), n=500)
graph2.500 <- replicate(test.both.methods("sim.graph.2.r.txt",
 sample.size=500), n=500)
graph3.500 <- replicate(test.both.methods("sim.graph.3.r.txt",
 sample.size=500), n=500)
graph4.500 <- replicate(test.both.methods("sim.graph.4.r.txt",
 sample.size=500), n=500)
graph5.500 <- replicate(test.both.methods("sim.graph.5.r.txt",
 sample.size=500), n=500)
graph6.500 <- replicate(test.both.methods("sim.graph.6.r.txt",
 sample.size=500), n=500)
graph7.500 <- replicate(test.both.methods("sim.graph.7.r.txt",
 sample.size=500), n=500)

graph1.250 <- replicate(test.both.methods("sim.graph.1.r.txt",
  sample.size=250), n=500)
graph2.250 <- replicate(test.both.methods("sim.graph.2.r.txt",
  sample.size=250), n=500)
graph3.250 <- replicate(test.both.methods("sim.graph.3.r.txt",
  sample.size=250), n=500)
graph4.250 <- replicate(test.both.methods("sim.graph.4.r.txt",
  sample.size=250), n=500)
graph5.250 <- replicate(test.both.methods("sim.graph.5.r.txt",
  sample.size=250), n=500)
graph6.250 <- replicate(test.both.methods("sim.graph.6.r.txt",
  sample.size=250), n=500)
graph7.250 <- replicate(test.both.methods("sim.graph.7.r.txt",
  sample.size=250), n=500)


save.image("simulation.results.latents.RData")




score.graph1.10000<- score.both(graph1.10000, graph.name="sim.graph.1.r.txt")
score.graph1.1000<- score.both(graph1.1000, graph.name="sim.graph.1.r.txt")
score.graph1.500<- score.both(graph1.500, graph.name="sim.graph.1.r.txt")
score.graph1.250<- score.both(graph1.250, graph.name="sim.graph.1.r.txt")

score.graph2.10000<- score.both(graph2.10000, graph.name="sim.graph.2.r.txt")
score.graph2.1000<- score.both(graph2.1000, graph.name="sim.graph.2.r.txt")
score.graph2.500<- score.both(graph2.500, graph.name="sim.graph.2.r.txt")
score.graph2.250<- score.both(graph2.250, graph.name="sim.graph.2.r.txt")

score.graph3.10000<- score.both(graph3.10000, graph.name="sim.graph.3.r.txt")
score.graph3.1000<- score.both(graph3.1000, graph.name="sim.graph.3.r.txt")
score.graph3.500<- score.both(graph3.500, graph.name="sim.graph.3.r.txt")
score.graph3.250<- score.both(graph3.250, graph.name="sim.graph.3.r.txt")

score.graph4.10000<- score.both(graph4.10000, graph.name="sim.graph.4.r.txt")
score.graph4.1000<- score.both(graph4.1000, graph.name="sim.graph.4.r.txt")
score.graph4.500<- score.both(graph4.500, graph.name="sim.graph.4.r.txt")
score.graph4.250<- score.both(graph4.250, graph.name="sim.graph.4.r.txt")

score.graph5.10000<- score.both(graph5.10000, graph.name="sim.graph.5.r.txt")
score.graph5.1000<- score.both(graph5.1000, graph.name="sim.graph.5.r.txt")
score.graph5.500<- score.both(graph5.500, graph.name="sim.graph.5.r.txt")
score.graph5.250<- score.both(graph5.250, graph.name="sim.graph.5.r.txt")

score.graph6.10000<- score.both(graph6.10000, graph.name="sim.graph.6.r.txt")
score.graph6.1000<- score.both(graph6.1000, graph.name="sim.graph.6.r.txt")
score.graph6.500<- score.both(graph6.500, graph.name="sim.graph.6.r.txt")
score.graph6.250<- score.both(graph6.250, graph.name="sim.graph.6.r.txt")

score.graph7.10000<- score.both(graph7.10000, graph.name="sim.graph.7.r.txt")
score.graph7.1000<- score.both(graph7.1000, graph.name="sim.graph.7.r.txt")
score.graph7.500<- score.both(graph7.500, graph.name="sim.graph.7.r.txt")
score.graph7.250<- score.both(graph7.250, graph.name="sim.graph.7.r.txt")




# TODO: Rewrite this portion. It is ugly, and should be in a function.
number.graphs<-7
graph.groups <- c()
for(k in 1:number.graphs){
    graph.groups[[k]] <- list(
     ls()[grep(pattern=paste("score.graph",
     k, ".", sep=""), ls(), fixed=T)])
}

```

```r

save.image("scored.simulations.latents.RData")



pdf("sim.res_final.pdf")
plot.rates(convert.graph.groups(graph.groups))
dev.off()
```

```

# converts the Tetrad .r.txt representation to a graphNEL object.
read.dag <- function(file){
    orig.mat <- read.table(file=file)

    orig.mat[orig.mat==1] <-0
    orig.mat[orig.mat==-1] <-1


    final.graph <- igraph.to.graphNEL(graph.adjacency(as.matrix(orig.mat)))

    return(final.graph)
}


# Takes in a graphNEL object, and generates normally distributed data from it.
generate.data.from.dag <- function(graph, n=100, errDist="normal"){

    top.sort <- topological.sort(igraph.from.graphNEL(graph))
    var.names <- nodes(graph)[top.sort]

    graph<-igraph.to.graphNEL(graph.adjacency(as(graph, "matrix")[var.names,
     var.names]))

    generated.data<-data.frame(rmvDAG(dag=graph, n=n, errDist=errDist))
    names(generated.data) <- var.names

    return(generated.data)
}

```

```r

# Runs both FA test, as well as detect.MIMIC
test.both.methods <- function(file="sim.graph.1.r.txt", sample.size=1000,
 alpha=.01, pval=.05, cut.off=.3, scree=FALSE){

    dataset <- generate.data.set(file=file, sample.size=sample.size)[[1]]

    clean.dataset <- dataset[,-grep(pattern="[L:digit:]",
     names(dataset))]

    # print(head(clean.dataset))

    fa.results <- test.fa(clean.dataset,
         n.latents=ncol(dataset)-ncol(clean.dataset), cut.off=cut.off,
         scree==scree)

    detect.mimic.results <- find.mimic(clean.dataset, alpha=alpha,
             pval=pval)


    return(list(fa.results=fa.results,
         detect.mimic.results=detect.mimic.results))

}

# Runs runs factor analysis on datset. If scree=TRUE, then attempts to
# discover n.latents. Uses varimax rotation.
test.fa <- function(dataset, n.latents=1, cut.off=.3, scree=TRUE){

    var.names <- names(dataset)

    if(scree){
        # As most papers have suggested using several different tests for
        # n.factors, have chosen to choose most frequent number reported. In
        # the event of a tie, the smaller number of latents is chosen, as
        # most papers suggest parsimoney.
        n.latents <-
         as.numeric(names(which.max(table(unlist(c(nScree(x=dataset,
         model="factors")$Components))))))
        fa.model <- factanal(dataset, factors=n.latents)
    }
    else{
        fa.model <- factanal(dataset, factors=n.latents)
    }


    return(fa.model)
}



generate.data.set <- function(file="sim.graph.1.r.txt", sample.size=10000){
    graph.data <- list(generate.data.from.dag(read.dag(file),
     n=sample.size))

    return(graph.data)
}


test.graphs <- function(n.graphs=1:7, seed.set=NULL, sample.size=1000, plot.graphs=TRUE){

    if(!is.null(seed.set)){set.seed(seed.set)}

    found.graphs <- list()

    files.to.load<-paste("sim.graph.",n.graphs,".r.txt", sep="")
    for(i in files.to.load){

        current.graphs <- test.graph(i, sample.size=sample.size)

        found.graphs<-list(found.graphs, current.graphs)

        if(plot.graphs){plot.test(current.graphs)}
    }
    return(found.graphs)
}

# Reads in graph file, and generates normally distributed data from it.
test.graph <- function(file, sample.size=1000){
    true.graph<-read.dag(file=file)

    generated.data <- generate.data.from.dag(graph=true.graph, n=sample.size)

    generated.data.no.latents <- generated.data[,-grep(pattern="[L:digit:]",
     names(generated.data))]

    result<-find.mimic(generated.data.no.latents)
    return(list(result=result, true.graph=true.graph))
}

# Plots the various stages of the algorithm used in finding the graph.
plot.test<-function(graph.list){

    results<-graph.list$result
    true.graph<-graph.list$true.graph

    # No results, so just plots true graph
    if(is.null(results)){plot(true.graph); return()}
    else if(class(results)!="list"){plot(results)}
    else{
        print(results)

        par(mfrow=c(2,3))
        if(!is.null(true.graph)){plot(true.graph, main="True Graph")}

        if(!is.null(results$pc.depth.0)){plot(results$pc.depth.0,
             main="PC Depth=0")}

        if(!is.null(results$pre.sober.model)){
                plot(results$pre.sober.model, main="Pre-Sober")}

        if(!is.null(results$mimic.model.graph)){
            plot(results$mimic.model.graph, main="Post-Sober")}

        if(!is.null(results$last.pc)){plot(results$last.pc,
            main="PC Depth>0")}

        if(!is.null(results$final.model)){plot(results$final.model,
            main="Final Graph")}
    }

}

```

```
# Returns average fa.score and mimic.score. Note that scores are
# as follows:

#           tpr
#           True Positive Rate: Number of correctly found edges (in estimated
# graph) divided by number of true edges (in true graph)
#
#           fpr
#           False Positive Rate: Number of incorrectly found edges divided by
#  number of true gaps (in true graph)
#
#           tdr
#           True Discovery Rate: Number of correctly found edges divided by
#  number of found edges (both in estimated graph)
score.both <- function(graph.sim, graph.name, cut.off=.3){

    true.graph <- as(read.dag(graph.name), "matrix")

    var.names <- names(data.frame(true.graph))

    latent.positions <- grep(pattern="[L:digit:]", var.names)

    non.latent.positions <- (1:length(var.names))[-latent.positions]

    sorted.truth <- (1:length(var.names))[c(non.latent.positions,
         latent.positions)]

    true.graph <- true.graph[sorted.truth,sorted.truth]

    # Scores FA model.
    fa.score <- lapply(graph.sim[1,],
         function(fa.model){score.fa(fa.model=fa.model, cut.off=cut.off,
             true.graph=true.graph)})

    # Scores MIMC model.
    mimic.score <- lapply(graph.sim[2,],
         function(mimic.model){
            # Handles case when only PC output was returned.
            if(length(mimic.model)>1){
                result<-score.mimic(mimic.model=mimic.model$final.model,
             true.graph=true.graph)

                return(result)
                }
            return(c(0,0,0))
            })

    n.with.incorrect.latents.mimic <- 0
    for(i in mimic.score){
        if(is.null(i)){n.with.incorrect.latents.mimic <-
             n.with.incorrect.latents.mimic+1}
    }

    n.with.incorrect.latents.fa <- 0
    for(i in fa.score){
        if(is.null(i)){n.with.incorrect.latents.fa <-
             n.with.incorrect.latents.fa+1}
    }

    fa.score <- (do.call(rbind, fa.score))
    mimic.score <- (do.call(rbind, mimic.score))

    if(is.null(fa.score)){fa.score <- c(0,0,0)}
    else{fa.score <- colMeans(fa.score)}
    mimic.score <- colMeans(mimic.score)

    return(list(fa.score=fa.score, mimic.score=mimic.score,
         latent.incorrect.mimic = n.with.incorrect.latents.mimic,
        latent.incorrect.fa = n.with.incorrect.latents.fa))
}

score.mimic <- function(mimic.model, true.graph){

    adj.matrix.true <- data.frame(true.graph)

    adj.matrix.mimic <- data.frame(as(mimic.model, "matrix"))

    n.latents.truth <- grep(pattern="[L:digit:]",
     names(adj.matrix.true))

    n.latents.mimic <- grep(pattern="[L:digit:]",
     names(adj.matrix.mimic))

    true.graph<-igraph.to.graphNEL(graph.adjacency(true.graph))


    # If the mimic.model found has a different numnber of latents than the
    # true graph, then cannot calculate tpr, fpr, tdr. Have therfore treated
    # those as NULL objects. (i.e., as with FA, correct n.latents is assumed)
    if(length(nodes(mimic.model)) == length(nodes(true.graph))){
        return(compareGraphs(mimic.model, true.graph))
    }
    return(NULL)
}

# Scores FA model.
score.fa <- function(fa.model, cut.off=.3, true.graph){

    fa.mat <- as(fa.model$loadings, "matrix")
    n.latents <- ncol(fa.mat)
    var.names <- row.names(fa.mat)
    n.vars <- nrow(fa.mat)

    fa.model <- prune.fa.paths(fa.model, cut.off=cut.off)

    fa.model <- igraph.to.graphNEL(graph.adjacency(fa.model))
    true.graph <- igraph.to.graphNEL(graph.adjacency(true.graph))

    if(length(nodes(fa.model))==length(nodes(true.graph))){

        graph.comparison <- (compareGraphs(fa.model, true.graph))
        return(graph.comparison)
    }
    return(NULL)
}


prune.fa.paths <- function(fa.model, cut.off=.3){

    fa.loadings.matrix <- as(fa.model$loadings, "matrix")

    n.latents <- ncol(fa.loadings.matrix)
    var.names <- row.names(fa.loadings.matrix)
    n.vars <- length(var.names)

    adj.mat <- matrix(FALSE, nrow=(n.vars+n.latents),
     ncol=(n.vars+n.latents), dimnames=list("row"=c(var.names,
         1:n.latents), "col"=c(var.names, 1:n.latents)))

    for(i in 1:n.latents){

        adj.mat[abs(fa.model$loadings[,i])>cut.off,
        i+length(var.names)] <- TRUE
    }
```

```r

    adj.mat[(length(var.names)+1):(length(var.names)+n.latents),
     (length(var.names)+1):(length(var.names)+n.latents)] <- FALSE

    return(adj.mat)

}

get.latent.cluster <- function(adj.matrix, n.latents, n.vars){
    latent.vectors.col <- adj.matrix[,(n.vars+1):(n.vars+n.latents)]
    latent.vectors.row <- t(adj.matrix[(n.vars+1):(n.vars+n.latents),])

    latent.clusters <- (latent.vectors.col+latent.vectors.row)[
    -((n.vars+1):(n.vars+n.latents)),]
    return(latent.clusters)
}


convert.graph.groups <- function(graph.groups = graph.groups){
    graph.final<-list()


    for(i in 1:length(graph.groups)){
        graph.list <- unlist(graph.groups[[i]])[c(3,4,1,2)]
        n.null.mimic <- c()
        n.null.fa <- c()
        graph.fa.score <- c()
        graph.mimic.score <- c()
        var.names <- c()
        for(j in 1:length(graph.list)){

            graph.score<- get(graph.list[j])

            graph.fa.score <- rbind(graph.fa.score, unlist(graph.score[[1]]))
            graph.mimic.score <- rbind(graph.mimic.score,
                 unlist(graph.score[[2]]))

            #   TODO: need to change this bit so that it handles the two-null
            # case.
            n.null.mimic <- c(n.null.mimic, unlist(graph.score[[3]]))
            n.null.fa <- c(n.null.fa, unlist(graph.score[[4]]))
        }

        row.names(graph.fa.score) <- c("250", "500", "1000", "10000")
        row.names(graph.mimic.score) <- c("250", "500", "1000", "10000")

        graph.final[[i]] <- list(fa.scores=graph.fa.score,
             mimic.scores=graph.mimic.score, n.null.mimic=n.null.mimic,
            n.null.fa=n.null.fa)

    }
    return(graph.final)
}

plot.rates <-function(score.list){
        for(i in 1:length(score.list)){

            par(mfrow=c(2,2))

            barplot(cbind(score.list[[i]]$mimic.scores[,1],
                 score.list[[i]]$fa.scores[,1]),
                main=paste("Graph ",i, " True Positive Rate", sep=""),
                 col=c(2,2,2,2, 3,3,3,3),
                 ylab="Rate", xlab="Number of observations",
                 ylim=c(0,1.2), beside=T, names=c("250", "500", "1000",
                 "10000","250", "500", "1000", "10000"), yaxt="n", las=2)

            axis(2, at=c(0,.2,.4,.6,.8,1),labels=c(0,.2,.4,.6,.8,1),
             col.axis="black", las=2)


            #legend(x="topright", col=2:3, legend=c("detect.MIMIC",
            #"Factor Analysis"), pch=16, xpd=TRUE)


            barplot(cbind(score.list[[i]]$mimic.scores[,2],
                 score.list[[i]]$fa.scores[,2]),
                main=paste("Graph ",i, " False Positive Rate", sep=""),
                 col=c(2,2,2,2, 3,3,3,3),
                 ylab="Rate", xlab="Number of observations",
                 ylim=c(0,1.2), beside=T, names=c("250", "500", "1000",
                 "10000","250", "500", "1000", "10000"), yaxt="n", las=2)

                axis(2, at=c(0,.2,.4,.6,.8,1),labels=c(0,.2,.4,.6,.8,1),
                 col.axis="black", las=2)


                legend(x="topright", col=2:3, legend=c("detect.MIMIC",
                 "Factor Analysis"), pch=16, xpd=TRUE)

            barplot(cbind(score.list[[i]]$mimic.scores[,3],
                 score.list[[i]]$fa.scores[,3]),
                main=paste("Graph ",i, " True Discovery Rate", sep=""),
                 col=c(2,2,2,2, 3,3,3,3),
                 ylab="Rate", xlab="Number of observations",
                 ylim=c(0,1.2), beside=T, names=c("250", "500", "1000",
                 "10000","250", "500", "1000", "10000"), yaxt="n", las=2)

                axis(2, at=c(0,.2,.4,.6,.8,1),labels=c(0,.2,.4,.6,.8,1),
                 col.axis="black", las=2)


            #   legend(x="topright", col=2:3, legend=c("detect.MIMIC",
            #    "Factor Analysis"), pch=16, xpd=TRUE)

            barplot(cbind(score.list[[i]]$n.null.mimic/500,
                 score.list[[i]]$n.null.fa/500),
                main=paste("Graph ",i,
                " Precentage of False\n Latent Cases (out of 500)",
                 sep=""), col=c(2,2,2,2, 3,3,3,3),
                names=c("250", "500", "1000",
                 "10000","250", "500", "1000", "10000"),
                ylab="Percentage incorrect",
                xlab="Number of observations",
                 ylim=c(0,1.2), beside=T, yaxt="n", las=2)

                axis(2, at=c(0,.2,.4,.6,.8,1),labels=c(0,.2,.4,.6,.8,1),
                 col.axis="black", las=2)

            #   legend(x="topright", col=2:3, legend=c("detect.MIMIC",
            #    "Factor Analysis"), pch=16, xpd=TRUE)


        }
#       par(mfrow=c(2,2))

#       for(i in 1:length(score.list)){
#           plot(read.dag(paste("sim.graph.", i, ".r.txt", sep="")),
#            main=paste("True Graph ", i, sep=""))
#
#       }

}
```

```r

# TODO: If neccessary, purge variables after initial pc model is found if they
# have both an indegree and outdegree of 0. (for memory optimization
# purposes)
# TODO: Clean up code.
# TODO: Break up long functions into smaller helper functions
# TODO: Comment this spaghetti code

# require(pcalg)
# require(igraph)
# require(gRim)
# require(plotrix)

find.mimic <- function(data, alpha=.01, indepTest=gaussCItest, pval=.05,
    print.intermediate=FALSE, high.dim=FALSE){

    orig.names <- names(data)

    names(data) <- paste("X", 1:ncol(data), sep="")

    pc.model <- find.pc.model(data=data, alpha=alpha, indepTest=indepTest)

    deg.model<- graph::degree((pc.model))
    # If no edges are directed, then return the undirected pc graph.

    if(length(deg.model) < 2 ||
    length(deg.model) >2){return(pc.model)}

    input.outputs <- find.in.out(pc.model)

    # If no outputs have been found
    if(is.null(input.outputs$inputs) ||
        is.null(input.outputs$outputs)){return(pc.model)}

    latent.structure <- finding.latent.structure(input.outputs, pc.model)

    if(high.dim){rm(pc.model)}

    sobers.step <- sobers.criterion(latent.structure, data,
         input.outputs, pval)

    if(high.dim){rm(latent.structure)}

    last.pc <- final.pc.run(data=data, alpha=alpha, indepTest=indepTest)

    mimic.model.list<-convert.list.to.adj.mat(list.obj=sobers.step,
         inputs.and.outputs=input.outputs, var.names=names(data))

    n.lat <- ncol(mimic.model.list)


    names(mimic.model.list) <- c(orig.names, paste("L",
     1:(ncol(mimic.model.list)-ncol(data)), sep=""))


    mimic.model.graph <- igraph.to.graphNEL(graph.adjacency(as.matrix(mimic.model.list)))

    if(high.dim){rm(mimic.model.list); rm(sobers.step)}

        if(print.intermediate){
            print("inputs and outputs")
            print(input.outputs)
            print("latent.structure - in find.MIMIC")
            print(latent.structure)
            print("sobers.step")
            print(sobers.step)
            print("mimic.model.list")
            print(mimic.model.list)
        }

    final.model <- last.step(inputs.outputs=input.outputs, pc.graph=last.pc,
         mimic.graph=mimic.model.graph)


        nodes(final.model) <- c(orig.names, paste("L",
         1:(n.lat-ncol(data)), sep=""))


    if(high.dim){return(list(final.model=final.model))}
    else{
        # names(data) <- c(orig.names)

        pre.sober.model <- convert.list.to.adj.mat(list.obj=latent.structure,
                 inputs.and.outputs=input.outputs, var.names=names(data))


        # names(pre.sober.model) <- nodes(final.model)

        pre.sober.model <-
         igraph.to.graphNEL(graph.adjacency(as.matrix(pre.sober.model)))

        return(list("pc.depth.0"=pc.model, inputs.outputs=input.outputs,
         latent.structure=latent.structure, pre.sober.model=pre.sober.model,
        sobers.step=sobers.step, last.pc=last.pc,
         mimic.model.graph=mimic.model.graph, final.model=final.model))
    }
}

# Finds PC model. Determines optimal depth via recursion.
find.pc.model<-function(data, depth=0, prev.graph=0, indepTest=gaussCItest,
     alpha=0.01, suffStat=0, n=0, p=0){

    n <- nrow(data)
    p <- ncol(data)

    ## define sufficient statistics
    suffStat <- list(C = cor(data), n = n)
    pc.model<-pc(suffStat=suffStat, indepTest=indepTest, p=p, alpha=alpha,
            m.max=depth)@graph
    return(pc.model)
}


# Determines inputs/outputs. Takes a GraphNEL object as input. returns names of inputs and outputs
find.in.out <- function(graph){
    indegree.0<- graph::degree(graph)$inDegree==0
    inputs <- c(which(indegree.0))

    candidate.outputs <- which(!indegree.0)

    adj.matrix <- as(graph, "matrix")

    adj.matrix <- adj.matrix[,candidate.outputs]

    outputs <- apply(adj.matrix, 2,
         function(pos.output){
            if(sum(which(pos.output==1)%in%inputs)>0){
                return(pos.output)
            }})
    if(class(outputs)=="list"){
        outputs<-remove.null.from.list(outputs)
        # Extracts the list vector, containg the names of each output.
        outputs<-names(outputs)
    }
    else{
        # Extracts the second list vector, containg the col names of matrix
        outputs <- dimnames(outputs)[[2]]
    }
    inputs <- names(inputs)
    return(list(inputs=inputs, outputs=outputs))
}

finding.latent.structure <- function(inputs.and.outputs, graph){
    latent.list <- finding.latents(inputs.and.outputs, graph)
    latent.list <- remove.null.from.list(latent.list)

        if(length(latent.list)==0){
            adj.matrix <- as(graph, "matrix")
            var.names <- names(data.frame(adj.matrix))

            latent.list[['1']]$inputs <-
             var.names[as.numeric(inputs.and.outputs$inputs)]

            latent.list[['1']]$outputs <-
             var.names[as.numeric(inputs.and.outputs$outputs)]


            latent.list[['1']]$latent <- c(1,1)
            return(latent.list)
        }

    for(i in 1:(length(latent.list))){

        latent.pair <- smallest.two.subsets(latent.list,
             n.inputs=length(inputs.and.outputs$inputs))

        smallest <- latent.pair$smallest
        nextSmallest <- latent.pair$nextSmallest

        if(smallest==nextSmallest){next()}

            for(j in 1:length(latent.list)) {
                if(j!=smallest){
                    latent.list[[j]]$inputs <-
                     remove.subset(latent.list[[smallest]]$inputs,
                            latent.list[[j]]$inputs)

                }

            }

            latent.list[[nextSmallest]]$latent <-
             c(latent.list[[nextSmallest]]$latent,
                 as.character(smallest), as.character(nextSmallest))
        }
    return(latent.list)
}

finding.latents <- function(inputs.and.outputs, graph){
    adj.matrix <- as(graph, "matrix")
    var.names <- names(data.frame(adj.matrix))

    inputs  <- inputs.and.outputs$inputs
    outputs <- inputs.and.outputs$outputs

    input.parents <- adj.matrix[inputs,]

    if(class(input.parents)=="matrix"){
    #TODO: Redundant. Check if this section is actually called (or is ever called). Data analysis
version needed changed (had to be fixed).
        input.parents <- data.frame(input.parents[,
            unique(which(as(input.parents, "matrix")==1, arr.ind=T)[,2])])
    }
    else{
        input.parents <- data.frame(matrix(which(input.parents==1), nrow=1),
         row.names=inputs)
        names(input.parents) <- paste("X", input.parents, sep="")
        input.parents[(input.parents>0)]<-1
    }
    latent.list <- construct.latent.list(input.parents,
            var.names=var.names)

    return(latent.list)
}

construct.latent.list <- function(input.parents, var.names){
    latent.list <- c()

    for(i in 1:ncol(input.parents)){

        if(list.exactly.contains(latent.list,
             names(col.same(input.parents, column=i)))){next()}
        else{

            outputs <- names(input.parents)[col.same(input.parents, column=i)]
            latent.list[[i]] <- list(outputs=outputs,
                 inputs=var.names[as.numeric(get.row.names(input.parents,
                     outputs))])
        }
    }
    return(latent.list)
}

sobers.criterion <- function(latent.structure, data, inputs.and.outputs,
     pval=.05){
    inputs<-c()
    outputs<-c()

        latents <- get.latents(latent.structure)

        for(i in 1:length(latents)){
            if(is.null(latents)){break()}
            if(is.null(latent.structure[[i]]$latent)){next()}

            inputs <- c(get.inputs.via.latents(latent.structure,
                 latents[[i]]))
            outputs <- c(get.outputs.via.latents(latent.structure,
                     latents[[i]]))

            inputs <- unique(inputs)
            outputs <- unique(outputs)

            dsep.inputs <- find.dsep(inputs, outputs, data, pval)
            if(is.null(dsep.inputs)){return(latent.structure)}
            min.set <- 1

            for(j in 1:length(dsep.inputs)){
                if(length(dsep.inputs[[j]]) < length(dsep.inputs[[min.set]])){
                    min.set<-j
                }
            }
                latent.structure[[as.numeric(latent.structure[[i]]$latent[2])]]$inputs <-
             unique(c(dsep.inputs[min.set], latent.structure[[as.numeric(
                 latent.structure[[i]]$latent[2])]]$inputs))
                                latent.structure[[as.numeric(latent.structure[[i]]$latent[1])]]
$inputs <-
             unique(c(dsep.inputs[min.set], latent.structure[[as.numeric(
                 latent.structure[[i]]$latent[1])]]$inputs))
```

```r


                if(length(latent.structure[[i]]$latent)==2){
                    latent.structure[[i]]$latent<-NULL
                }
                else{
                    latent.structure[[i]]$latent <- c(latent.structure[[i]]$
                        latent[3:length(latent.structure[[i]]$latent)])

                }

        }
        return(latent.structure)
}

# Last step of the PC run
final.pc.run <- function(data, depth=0, prev.graph=NULL,
     indepTest=gaussCItest, alpha=0.01, suffStat=0, n=0, p=0){

    if(depth==0){
        n <- nrow(data)
        p <- ncol(data)

        ## define sufficient statistics
        suffStat <- list(C = cor(data), n = n)
    }

    new.graph <- pc(suffStat=suffStat, indepTest=indepTest, p=p, alpha=alpha,
         m.max=depth+1)@graph

    if(is.null(prev.graph)){prev.graph<-pc(suffStat=suffStat,
         indepTest=indepTest, p=p, alpha=alpha, m.max=depth)@graph}

    prev.graph <- igraph.from.graphNEL(prev.graph)
    new.graph <- igraph.from.graphNEL(new.graph)

    if(isTRUE(all.equal(as(as.undirected(prev.graph), "matrix"),
     as(as.undirected(new.graph), "matrix")))){
        return(igraph.to.graphNEL(prev.graph))
    }
    else{
        new.graph <- igraph.to.graphNEL(new.graph)
        return(final.pc.run(data=data, depth=depth+1, prev.graph=new.graph,
             indepTest=indepTest, alpha=alpha, suffStat=suffStat, n=n, p=p))
    }

}

# Converts Sobers step to adj mat.
# TODO: Break into helper functions
convert.list.to.adj.mat <- function(list.obj, inputs.and.outputs, var.names){
    inputs <- inputs.and.outputs$inputs
    outputs <- inputs.and.outputs$outputs
    n.variables <- length(var.names)
    n.latent <- c()
    n.unconnected.latents <- 0


    for(i in 1:length(list.obj)){
        if(!is.null(list.obj[[i]]$latent)){
            n.latent<-(c(n.latent, list.obj[[i]]$latent))
        }
    }

    for(i in 1:length(list.obj)){
        if(is.null(list.obj[[i]]$latent) &&
        isFALSE(as.character(i)%in%n.latent)){
            n.unconnected.latents <- n.unconnected.latents+1
        }
    }

    n.latent<-length(unique(n.latent))+n.unconnected.latents

    adj.mat <- matrix(nrow=length(var.names)+n.latent,
     ncol=length(var.names)+n.latent, data=rep(FALSE,
         (length(var.names)+n.latent)*(length(var.names)+n.latent)))

    adj.mat <- data.frame(adj.mat)

    names(adj.mat) <- c(var.names, 1:n.latent)

    print((var.names))
    print(c(row.names(adj.mat), 1:n.latent))

    row.names(adj.mat) <- c(var.names, 1:n.latent)

    # assign latents to their positions
    for(i in 1:n.latent){
        if(n.latent>=1){
            adj.mat[n.variables+i,
             ] <- c(var.names%in%unlist(list.obj[[i]]$outputs), rep(FALSE,
                 n.latent))

            adj.mat[, n.variables+i
             ] <- c(var.names%in%unlist(list.obj[[i]]$inputs), rep(FALSE,
                 n.latent))


            if(!is.null(list.obj[[i]]$latent)){
                total.latents <- length(list.obj[[i]]$latent)


                for(j in 1:(ceiling(total.latents/2))){
                    # As latents are stored in ordered, pairs, ensures that
                    # the odd position=left, even position=right
                    left.lat <-list.obj[[i]]$latent[2*(j)-1]
                    right.lat <- list.obj[[i]]$latent[2*(j)]

                    adj.mat[length(var.names)+as.numeric(left.lat),
                     length(var.names)+as.numeric(right.lat)]<-TRUE
                }
            }
        }
    }
    # Removes self-causing latents
    diag(adj.mat)<-FALSE
    return(adj.mat)
}

# Removes latent-to-output edges based on output of PC depth>0.
last.step <- function(inputs.outputs, pc.graph, mimic.graph){

    inputs <- as.numeric(inputs.outputs$inputs)
    outputs <- as.numeric(inputs.outputs$outputs)

    pc.adj.matrix <- data.frame(as(pc.graph, "matrix"))
    mimic.adj.matrix <- data.frame(as(mimic.graph, "matrix"))

    n.vars <- ncol(pc.adj.matrix)

    n.latents <- ncol(mimic.adj.matrix)-n.vars

    false.outputs <- names(which(apply(pc.adj.matrix[inputs,
     outputs], 2, sum)==0))
```

```r

    names(mimic.adj.matrix) <- c(names(pc.adj.matrix), paste("L",
     1:n.latents, sep=""))


    if(!is.null(false.outputs) && length(false.outputs)>0){
            for(j in false.outputs){

                false.connected.latents<-(which(mimic.adj.matrix[
                 j]==1))

                mimic.adj.matrix[false.connected.latents,
                 which(names(mimic.adj.matrix)%in%j)] <- 0

            }

        for(i in false.outputs){
            false.connected.latents<-(which(mimic.adj.matrix[
             i]==1))
            mimic.adj.matrix[(1:n.vars),
             which(names(mimic.adj.matrix)%in%i)] <-
             pc.adj.matrix[,which(names(mimic.adj.matrix)%in%i)]

    mimic.adj.matrix[which(names(mimic.adj.matrix)%in%i),
    -(false.connected.latents)] <-
     pc.adj.matrix[which(names(mimic.adj.matrix)%in%i),]

            }

    }
    return(igraph.to.graphNEL(graph.adjacency(as.matrix(mimic.adj.matrix))))
}



# Helper Methods

# Returns the index of columns that are all identical.
col.same <- function(mat, column){
    return(which(colSums(mat[,column]==mat)==nrow(mat)))
}

list.exactly.contains <- function(list.object, search.term){
    return(isTRUE(sum(unlist(lapply(list.object,
        function(item){(search.term %in% item$outputs)})))>0))
}

get.row.names <- function(mat, col.names){
    return(row.names(mat)[unique(which(mat[c(col.names)]==1, arr.ind=T)[,1])])
}

remove.null.from.list <- function(list.object){
    purged.list <- list.object[-which(sapply(list.object,
            is.null),arr.ind=TRUE)]

    if(length(purged.list)==0){return(list.object)}
    else{return(purged.list)}
}

smallest.two.subsets <- function(latent.list, n.inputs){
    smallest <- NULL
    nextSmallest <- NULL

    for(i in 1:(length(latent.list))){
        for(j in 1:length(latent.list)){

            if(i==j){next()}

            if(is.subset(latent.list[[i]]$input, latent.list[[j]]$input)){

                # Ensures that subsets are being compared for smallest/n.small
                if(is.null(smallest)){smallest<-i; nextSmallest<-j}

                if(isTRUE(length(latent.list[[i]]$inputs) <=
                 length(latent.list[[smallest]]$inputs))){smallest<-i}
                else if(isTRUE(length(latent.list[[i]]$inputs) <=
                 length(latent.list[[nextSmallest]]$inputs))){nextSmallest<-i}

            }
            else if (is.subset(latent.list[[j]]$input,
                 latent.list[[i]]$input)){

                # Ensures that subsets are being compared for smallest/n.small
                if(is.null(smallest)){smallest<-j; nextSmallest<-i}

                if(isTRUE(length(latent.list[[j]]$inputs) <=
                 length(latent.list[[smallest]]$inputs))){smallest<-j}
                else if(isTRUE(length(latent.list[[j]]$inputs) <=
                 length(latent.list[[nextSmallest]]$inputs))){nextSmallest<-j}
            }
        }
    }

    if(is.null(smallest)){return(list(smallest=1, nextSmallest=1))}

    return(list(smallest=smallest, nextSmallest=nextSmallest))
}

there.is.a.subset <- function(latent.list, n.inputs){
    size.list <- smallest.two.subsets(latent.list,
         n.inputs=length(inputs.and.outputs$inputs))
        return(size.list[[1]]==size.list[[2]])

}


remove.subset <- function(small.set, larger.set){
    return(larger.set[!larger.set%in%small.set])
}

is.subset <- function(set.1, set.2){

    if(length(set.1)>length(set.2)){return(FALSE)}

    joint.membership <- c(0)
    for(element in set.1){
        joint.membership <- c(joint.membership + sum(set.2==element))
    }

    if(joint.membership==length(set.1)){
        return(TRUE)
    }
    else{
        return(FALSE)
    }

}

get.latents <- function(latent.structure){
    latents <- c()
    for(i in 1:length(latent.structure)){
        latents[[i]] <- latent.structure[[i]]$latent
    }
    return(latents)
```

```
}


get.inputs.via.latents <- function(latent.structure, latents){
    latents <- as.numeric(latents)
    inputs <- c()

    left.latent <- latents[1]
    right.latent <- latents[2]

    inputs <- c(latent.structure[[left.latent]]$inputs,
        latent.structure[[right.latent]]$inputs)

    return(inputs)
}

get.outputs.via.latents <- function(latent.structure, latents){
    latents <- as.numeric(latents)
    outputs <- c()

    left.latent <- latents[1]
    right.latent <- latents[2]

    outputs <- c(latent.structure[[left.latent]]$outputs,
        latent.structure[[right.latent]]$outputs)

    return(outputs)
}

find.dsep <- function(inputs, outputs, data, pval=.05){
        variations.list <- c()
        n.inputs <- length(inputs)

        for(i in 1:n.inputs){
            variations.list[[i]] <- list(combn(inputs, m=i))
        }

        condi.sets<-variations.list

        for(i in 1:length(condi.sets)){
            dsep.set<-get.cond.combo(data, outputs, condi.sets[i], pval)
            if(!is.null(dsep.set)){
                # print(dsep.set)
                return(dsep.set)
            }
        }
    return()
}

get.cond.combo <- function(data, outputs, input, pval){

    input <- destroy.list(input)

    # Handles single input cases.
    if(is.null(dim(input))){

        for(i in 1:length(input)){
            if(!ciTest(data, set=c(outputs[c(1, length(outputs))],
             (input[i])))$p.val<=pval){return(input[i])}
        }
    }
    else{
        for(i in 1:ncol(input)){
            if(!ciTest(data, set=c(outputs[c(1, length(outputs))],
             (input[,i])))$p.val<=pval){return(input[,i])}
        }
    return(c())
    }
}

destroy.list <- function(list.obj){
    if(class(list.obj)=="list"&&
        listDepth(list.obj)>1){

        destroy.list(unlist(list.obj, recursive=F))
    }
    else{return(list.obj[[1]])}
}

isFALSE <- function(truth.vector){return(!isTRUE(truth.vector))}

```

```r
source("new\ code/high_dim_indep.R")
#Running steps:
library(plotrix)

#Data formating

# Reading in output variables.
ov_tumor_expression.df <- read.csv("OV_tumor_expression_matrix_fold.csv")
ov_tumor_expression.df <-
 apply(ov_tumor_expression.df[,2:ncol(ov_tumor_expression.df)], 2,
 as.numeric)

#Reading in input variables.
ov_tumor_sga.df <- read.csv("OV_tumor_SGA_matrix.csv")
ov_tumor_sga.df <- apply(ov_tumor_sga.df[,2:ncol(ov_tumor_sga.df)], 2,
 as.numeric)


# Converting both datasets into factor variables.
ov_tumor_expression.df<-apply( ov_tumor_expression.df, 2, function(data){factor(data, levels=c(-1,0,1))})
ov_tumor_sga.df<-apply( ov_tumor_sga.df, 2, function(data){factor(data, levels=c(0,1))})

#converting from matrix to data.frame
ov_tumor_expression.df <- data.frame(ov_tumor_expression.df)
ov_tumor_sga.df <- data.frame(ov_tumor_sga.df)

#Combining datasets into a single dataset.
ov_tumor.df <- data.frame(ov_tumor_sga.df, ov_tumor_expression.df)

library(glmnet)

#converts to needed format for glmnet
ov_tumor.matrix <- data.matrix(ov_tumor.df)

is.constant <- function(vect){
    return(sum(vect[1]==vect)==length(vect))
}

inputs <- names(ov_tumor_sga.df)
inputs <- 1:length(inputs)
outputs <- names(ov_tumor_expression.df)
outputs <- (length(inputs)+1):(ncol(ov_tumor.df))

#Gets list of non-constant variables
exclude.constants<-!apply(ov_tumor.df, 2, is.constant)
exclude.constants<-which(exclude.constants)

inputs.cleaned <- which(exclude.constants<=max(inputs))
outputs.cleaned <- which(exclude.constants>max(inputs))

library(doParallel)
registerDoParallel(8)

#removing inputs
cv.lasso.results<-list()
for(i in (outputs.cleaned)){
    cv.lasso.results[[i]]<-cv.glmnet(alpha=1, x=ov_tumor.matrix[,inputs.cleaned], y=ov_tumor.matrix[,i], standardize=F, parallel=TRUE)

    cv.lasso.results[[i]]<-coef(cv.lasso.results[[i]], s=cv.lasso.results[[i]]$lambda.min)

    # The [[1]] part of the code selects the correct part of the Dimnames object. See str(cv.lasso.results[[i]]) for why.
    cv.lasso.results[[i]]<- cv.lasso.results[[i]]@Dimnames[[1]][which(cv.lasso.results[[i]]>0)]
}

```

```r


# removing outputs
cv.lasso.results.inputs<-list()
for(i in (inputs.cleaned)){
    cv.lasso.results.inputs[[i]]<-cv.glmnet(alpha=1, x=ov_tumor.matrix[,outputs.cleaned], y=ov_tumor.matrix[,i], standardize=F, parallel=TRUE)

    cv.lasso.results.inputs[[i]]<-coef(cv.lasso.results.inputs[[i]], s=cv.lasso.results.inputs[[i]]$lambda.min)

    # The [[1]] part of the code selects the correct part of the Dimnames object. See str(cv.lasso.results[[i]]) for why.
    cv.lasso.results.inputs[[i]]<- cv.lasso.results.inputs[[i]]@Dimnames[[1]][which(cv.lasso.results.inputs[[i]]>0)]

}

# Initial attempt
#reduced.dataset<-ov_tumor.df[,(names(ov_tumor.df[, inputs])%in%(unique(unlist(cv.lasso.results))))]
#inputs<-1:length(ov_tumor.df[,(names(ov_tumor.df[, inputs])%in%(unique(unlist(cv.lasso.results))))])

#reduced.dataset<-data.frame(reduced.dataset, ov_tumor.df[,(names(ov_tumor.df[, outputs])%in%(unique(unlist(cv.lasso.results.inputs))))])
#outputs<-(length(ov_tumor.df[,(names(ov_tumor.df[, inputs])%in%(unique(unlist(cv.lasso.results))))])+1):ncol(reduced.dataset)

# Need even more:
# 95% quantile: 42, 111, drops to 7085 variables.
# 99% quantile: 69, 141, drops to 4369 variables. verified.
temp<-cv.lasso.results[unlist(lapply(cv.lasso.results, length)>69)]
temp2<-cv.lasso.results.inputs[unlist(lapply(cv.lasso.results.inputs, length)>141)]

#Need to fix, as the renaning using .1 leads to too many variables being missed.
reduced.dataset.final<-ov_tumor.df[,(names(ov_tumor.df)%in%(unique(unlist(temp))))]
inputs.final<-1:length(ov_tumor.df[,(names(ov_tumor.df)%in%(unique(unlist(temp))))])

reduced.dataset.final<-data.frame(reduced.dataset.final, ov_tumor.df[,(names(ov_tumor.df)%in%(unique(unlist(temp2))))])
outputs.final<-(length(ov_tumor.df[,(names(ov_tumor.df)%in%(unique(unlist(temp))))])+1):ncol(reduced.dataset.final)


# Checking to see that have correctly subseted dataset. Note that 2 is subtracted as the (intercept) carries over.
#length(unique(unlist(temp2)))+length(unique(unlist(temp)))-2
#dim(reduced.dataset.final)


# This dataset has 4367 variables.
write.csv(reduced.dataset.final, "reduced.datasetv2.csv", row.names=F)

orig.names <-names(reduced.dataset.final)
names(reduced.dataset.final) <-paste("X", 1:ncol(reduced.dataset.final), sep="")

initial.adj.mat <- alternative.to.pc(inputs.final, outputs.final, dataset.df=reduced.dataset.final, p.val=.000001)

init.latent.struct <- finding.latent.structure(inputs.and.outputs=list(inputs=inputs.final, outputs=outputs.final), adj.matrix=initial.adj.mat)

sobers.step <- sobers.criterion(init.latent.struct, reduced.dataset.final,
          inputs.and.outputs=list(inputs=inputs.final, outputs=outputs.final), pval=.000001, categorical=TRUE)

#Converts list type object to desired data.frame version of adj.matrix. Data frame is used so that
```

```r
variables can easily be renamed.
mimic.model.list<-convert.list.to.adj.mat(list.obj=sobers.step,
         inputs.and.outputs=list(inputs=inputs.final, outputs=outputs.final), var.names=names
(reduced.dataset.final))


n.lat <- ncol(mimic.model.list)

#Renames variables to preserve original variable names
names(mimic.model.list) <- c(orig.names, paste("L",
     1:(ncol(mimic.model.list)-ncol(data)), sep=""))

library(igraph)
#Coverts dataframe to graphNEL object.
mimic.model.graph <- igraph.to.graphNEL(graph.adjacency(as.matrix(mimic.model.list)))


#Adds back in the variable names. 1:139 names the latents.
nodes(mimic.model.graph) <- c(orig.names, 1:139)

# Number of nodes with an indegree>0
summary(graph::degree(mimic.model.graph)$inDegree>0)

# Number of nodes with an outdegree>0
summary(graph::degree(mimic.model.graph)$outDegree>0)

# Number of nodes with a total degree>0
length(names(which((graph::degree(mimic.model.graph)$outDegree +graph::degree(mimic.model.graph)
$inDegree)>0)))

#Extracting a subgraph only containing nodes connected to other nodes.
reduced.final.graph<-igraph.from.graphNEL(subGraph(snodes=names(which((graph::degree
(mimic.model.graph)$outDegree +graph::degree(mimic.model.graph)$inDegree)>0)),
graph=mimic.model.graph))


latent.nodes<-which(V(reduced.final.graph)$name%in%1:n.lat)
input.nodes <- which(V(reduced.final.graph)$name%in%(names(which(degree(reduced.final.graph,
mode="in")==0))))
output.nodes <- which(V(reduced.final.graph)$name%in%(names(which(degree(reduced.final.graph,
mode="out")==0))))

#Assigning color based on whether node is input, latent, or output node. 1, 2, and 3 are colors.
cluster.mem.color <- ifelse(1:length(V(reduced.final.graph)$name)%in%latent.nodes, 1,
                ifelse(1:length(V(reduced.final.graph)$name)%in%output.nodes, 3,
                    ifelse(1:length(V(reduced.final.graph)$name)%in%input.nodes, 2, 0)))

# Full protein signalling network.
pdf("signal_network.pdf")
plot.igraph(reduced.final.graph, vertex.label=NA, edge.width=1, edge.arrow.size=0, vertex.size=3,
vertex.color=cluster.mem.color, vertex.frame.color=NA, layout=layout.fruchterman.reingold
(reduced.final.graph),
    main="Protein Signalling Network")
legend("topright", pch=16, col=c(1,2,3), legend=c("Latent","Input","Output"))
dev.off()



# Extracts a subgraph built from a given starting node.
plot.interesting.subgraph <- function(reduced.final.graph, starting.nodes=c("75"), ...){

    reduced.subgraph <- induced.subgraph(reduced.final.graph, v=get.subgraph(reduced.final.graph,
starting.nodes=starting.nodes))


    latent.nodes<-which(V(reduced.subgraph)$name%in%1:(length(V(reduced.final.graph))))

    input.nodes <- which(V(reduced.subgraph)$name%in%(names(which(degree(reduced.subgraph,
mode="in")==0))))

    output.nodes <- which(V(reduced.subgraph)$name%in%(names(which(degree(reduced.subgraph,
mode="out")==0))))

    cluster.mem.color <- ifelse(1:length(V(reduced.subgraph)$name)%in%latent.nodes, 1,
                    ifelse((1:length(V(reduced.subgraph)$name)%in%output.nodes), 3,
                        ifelse((1:length(V(reduced.subgraph)$name)%in%input.nodes), 2, 0)))

    plot.igraph(reduced.subgraph, layout=layout.fruchterman.reingold(reduced.subgraph,
area=40*vcount(reduced.subgraph)^2), vertex.label.color=cluster.mem.color, ...)
}



# Extracts a the maximal subgraph were all of the nodes are connected to one another, given a
starting node to search from.
get.subgraph <- function(orig.graph, starting.nodes=c(), searched.nodes=c()){
    adj.nodes<-c()
    for(i in 1:length(starting.nodes)){

        new.adj <- getElement(get.adjlist(reduced.final.graph), starting.nodes[i])
        adj.indices <- new.adj

        adj.nodes <- unique(c(adj.nodes, V(orig.graph)$name[as.numeric(adj.indices)]))
    }

    if(sum(adj.nodes %in% searched.nodes)==length(adj.nodes)){
        return(unique(searched.nodes))
    }
    else{
        return(unique(get.subgraph(orig.graph, starting.node=adj.nodes, searched.nodes=unique(c
(adj.nodes, searched.nodes)))))
    }
}

# Plots an interesting subgraph.
pdf("subgraph_final.pdf")
plot.interesting.subgraph(reduced.final.graph, edge.width=1, vertex.label=c(), edge.arrow.size=0.3,
vertex.label.cex=.5, vertex.color=0,
vertex.size=7, vertex.frame.color=NA, main="Protein Signalling Network", starting.nodes="76")
legend("topright", pch=16, col=c(1,2,3), legend=c("Latent","Input","Output"))
dev.off()


#Plot all subgraphs with 4 or more variables.
pdf("all_subgraph_final.pdf")
already.tested<-c()
    for(i in 1:length(latent.nodes)){
    temp.subgraph <- get.subgraph(reduced.final.graph, starting.nodes=i)
    if(length(temp.subgraph)<4 || sum(temp.subgraph%in%unlist(already.tested))==length
(temp.subgraph)){next()}
        else{
        plot.interesting.subgraph(reduced.final.graph, edge.width=1, vertex.label=c(),
edge.arrow.size=0.3, vertex.label.cex=.5,
         vertex.color=0,vertex.size=7, vertex.frame.color=NA, main="Protein Signalling Network",
starting.nodes=(i))
        legend("topright", pch=16, col=c(1,2,3), legend=c("Latent","Input","Output"))

        already.tested[[i]]<-temp.subgraph

    }
}
dev.off()

#EDA
```

```

pdf("eda_final.pdf")
# N.abnormalities (numeric vector) per subject
freq.of.abnormalities <- apply(ov_tumor_sga.df==1, 1, sum)

barplot(sort(freq.of.abnormalities), xlab="Subject", ylab="Number of abnormalities", main="Number
of Mutations\n for each Subject",
 ylim=c(0,max(freq.of.abnormalities)+100))


# N.subjects (numeric vector) per mutation
freq.of.subject <- apply(ov_tumor_sga.df==1, 2, sum)


# Bar Plot of frequency of abnormality.
barplot(sort(freq.of.subject), xlab="Gene", ylab="Number of subjects with a particular genetic
mutation", main="Mutation Frequency for each Gene", ylim=c(0,
     max(freq.of.subject)+100), names.arg=NA)

dev.off()
```

```r
require(gRim)

#require(pcalg)
#Takes vector containg index of inputs in dataset.df (same for outputs).
alternative.to.pc <- function(inputs, outputs, dataset.df, p.val){
    adj.mat<-matrix(nrow=ncol(dataset.df), ncol=ncol(dataset.df), 0)

    for(i in min(inputs):max(inputs)){
    #get number of levels for factor
        n.levels.input <- length(levels((dataset.df[,i])))

        for(j in min(outputs):max(outputs)){
            #get number of levels for factor
            n.levels.output <- length(levels((dataset.df[,j])))

        #If one of the variables is a constant, skip it.
        if(n.levels.output>1 && n.levels.input>1){
            # If variables i and j are dependent, write a one in the ith column and jth row of the adj. matrix.
            adj.mat[i, j] <- (fisher.test(x=dataset.df[,i], y=dataset.df[,j])$p.value<=p.val)
        }

    }

    }
    return(adj.mat)
}

finding.latent.structure <- function(inputs.and.outputs, adj.matrix){

    var.names <- paste("X", 1:ncol(adj.matrix), sep="")

    latent.list <- finding.latents(inputs.and.outputs, adj.matrix, var.names)
    latent.list <- remove.null.from.list(latent.list)

        if(length(latent.list)==0){
            #adj.matrix <- as(graph, "matrix")
            #var.names <- names(data.frame(adj.matrix))

            latent.list[['1']]$inputs <-
             var.names[as.numeric(inputs.and.outputs$inputs)]

            latent.list[['1']]$outputs <-
             var.names[as.numeric(inputs.and.outputs$outputs)]


            latent.list[['1']]$latent <- c(1,1)
            return(latent.list)
        }

    for(i in 1:(length(latent.list))){

        latent.pair <- smallest.two.subsets(latent.list,
             n.inputs=length(inputs.and.outputs$inputs))

        smallest <- latent.pair$smallest
        nextSmallest <- latent.pair$nextSmallest


        if(smallest==nextSmallest){next()}
        else{
            for(j in 1:length(latent.list)) {
                if(j!=smallest  && length(latent.list[[j]]$inputs)>1){

                    latent.list[[j]]$inputs <-
                     remove.subset(latent.list[[smallest]]$inputs,
                            latent.list[[j]]$inputs)
```

```

                 }

             }
         }

             latent.list[[nextSmallest]]$latent <-
              c(latent.list[[nextSmallest]]$latent,
                 as.character(smallest), as.character(nextSmallest))
         }
     return(latent.list)
}

 finding.latents <- function(inputs.and.outputs, adj.matrix, var.names){
    #adj.matrix <- as(graph, "matrix")
    #var.names <- names(data.frame(adj.matrix))


    inputs  <- inputs.and.outputs$inputs
    outputs <- inputs.and.outputs$outputs

    #print(inputs)

    # Note: This is a line that is adding alot of memory usage.
    input.parents <- adj.matrix[inputs,]

    if(class(input.parents)=="matrix"){

        output.names <- paste("X", which(apply(input.parents,2, function(column){sum(column)>0})),
sep="")
        input.parents <- data.frame(input.parents[,
            unique(which(as(input.parents, "matrix")==1, arr.ind=T)[,2])])
        names(input.parents) <- output.names
    }
    else{       #  1 in column => column var. effect of row var where 1 occurs.
        input.parents <- data.frame(matrix(which(input.parents==1), nrow=1),
         row.names=inputs)
        names(input.parents) <- var.names
        input.parents[(input.parents>0)]<-1
    }

    latent.list <- construct.latent.list(input.parents,
            var.names=var.names)

    return(latent.list)
}

construct.latent.list <- function(input.parents, var.names){
    latent.list <- c()

    for(i in 1:ncol(input.parents)){

        if(list.exactly.contains(latent.list,
             names(col.same(input.parents, column=i)))){next()}
        else{

            outputs <- names(input.parents)[col.same(input.parents, column=i)]
            latent.list[[i]] <- list(outputs=outputs,
                 inputs=var.names[as.numeric(get.row.names(input.parents,
                     outputs))])
        }
    }
    return(latent.list)
}

sobers.criterion <- function(latent.structure, data, inputs.and.outputs,
     pval=.05, categorical=TRUE){
```

```
    inputs<-c()
    outputs<-c()

        latents <- get.latents(latent.structure)

        for(i in 1:length(latents)){
            if(is.null(latents)){break()}
            if(is.null(latent.structure[[i]]$latent)){next()}

            inputs <- c(get.inputs.via.latents(latent.structure,
                 latents[[i]]))
            outputs <- c(get.outputs.via.latents(latent.structure,
                     latents[[i]]))

            inputs <- unique(inputs)
            outputs <- unique(outputs)

            dsep.inputs <- find.dsep(inputs, outputs, data, pval, categorical)
            if(is.null(dsep.inputs)){return(latent.structure)}
            min.set <- 1

            for(j in 1:length(dsep.inputs)){
                if(length(dsep.inputs[[j]]) < length(dsep.inputs[[min.set]])){
                    min.set<-j
                }
            }
                latent.structure[[as.numeric(latent.structure[[i]]$latent[2])]]$inputs <-
             unique(c(dsep.inputs[min.set], latent.structure[[as.numeric(
                 latent.structure[[i]]$latent[2])]]$inputs))

                latent.structure[[as.numeric(latent.structure[[i]]$latent[1])]]$inputs <-
             unique(c(dsep.inputs[min.set], latent.structure[[as.numeric(
                 latent.structure[[i]]$latent[1])]]$inputs))


                if(length(latent.structure[[i]]$latent)==2){
                    latent.structure[[i]]$latent<-NULL
                }
                else{
                    latent.structure[[i]]$latent <- c(latent.structure[[i]]$
                        latent[3:length(latent.structure[[i]]$latent)])

                }

        }
        return(latent.structure)
}


# Converts Sobers step to adj mat.
# TODO: Break into helper functions.
# TODO: Fix memory stuff in this.
convert.list.to.adj.mat <- function(list.obj, inputs.and.outputs, var.names){
    inputs <- inputs.and.outputs$inputs
    outputs <- inputs.and.outputs$outputs
    n.variables <- length(var.names)
    n.latent <- c()
    n.unconnected.latents <- 0

    for(i in 1:length(list.obj)){
        if(!is.null(list.obj[[i]]$latent)){
            n.latent<-(c(n.latent, list.obj[[i]]$latent))
        }
    }

    for(i in 1:length(list.obj)){
        if(is.null(list.obj[[i]]$latent) &&
```

```r
        isFALSE(as.character(i)%in%n.latent)){
            n.unconnected.latents <- n.unconnected.latents+1
        }
    }

    n.latent<-length(unique(n.latent))+n.unconnected.latents

    adj.mat <- matrix(nrow=length(var.names)+n.latent,
     ncol=length(var.names)+n.latent, data=rep(FALSE,
         (length(var.names)+n.latent)*(length(var.names)+n.latent)))

    adj.mat <- data.frame(adj.mat)

    names(adj.mat) <- c(var.names, 1:n.latent)

    print((var.names))
    print(c(row.names(adj.mat), 1:n.latent))

    row.names(adj.mat) <- c(var.names, 1:n.latent)

    # assign latents to their positions
    for(i in 1:n.latent){
        if(n.latent>=1){
            adj.mat[n.variables+i,
             ] <- c(var.names%in%unlist(list.obj[[i]]$outputs), rep(FALSE,
                 n.latent))

            adj.mat[, n.variables+i
             ] <- c(var.names%in%unlist(list.obj[[i]]$inputs), rep(FALSE,
                 n.latent))


            if(!is.null(list.obj[[i]]$latent)){
                total.latents <- length(list.obj[[i]]$latent)


                for(j in 1:(ceiling(total.latents/2))){
                    # As latents are stored in ordered, pairs, ensures that
                    # the odd position=left, even position=right
                    left.lat <-list.obj[[i]]$latent[2*(j)-1]
                    right.lat <- list.obj[[i]]$latent[2*(j)]

                    adj.mat[length(var.names)+as.numeric(left.lat),
                     length(var.names)+as.numeric(right.lat)]<-TRUE
                }
            }
        }
    }
    # Removes self-causing latents
    diag(adj.mat)<-FALSE
    return(adj.mat)
}

# Helper Methods

# Returns the index of columns that are all identical.
col.same <- function(mat, column){
    return(which(colSums(mat[,column]==mat)==nrow(mat)))
}

list.exactly.contains <- function(list.object, search.term){
    return(isTRUE(sum(unlist(lapply(list.object,
        function(item){(search.term %in% item$outputs)})))>0))
}

get.row.names <- function(mat, col.names){
    return(row.names(mat)[unique(which(mat[c(col.names)]==1, arr.ind=T)[,1])])
```

```r
}

remove.null.from.list <- function(list.object){
    purged.list <- list.object[-which(sapply(list.object,
            is.null),arr.ind=TRUE)]

    if(length(purged.list)==0){return(list.object)}
    else{return(purged.list)}
}

smallest.two.subsets <- function(latent.list, n.inputs){
    smallest <- NULL
    nextSmallest <- NULL

    for(i in 1:(length(latent.list))){
        for(j in 1:length(latent.list)){

            if(i==j){next()}

            if(is.subset(latent.list[[i]]$input, latent.list[[j]]$input)){

                # Ensures that subsets are being compared for smallest/n.small
                if(is.null(smallest)){smallest<-i; nextSmallest<-j}

                if(isTRUE(length(latent.list[[i]]$inputs) <=
                 length(latent.list[[smallest]]$inputs))){smallest<-i}
                else if(isTRUE(length(latent.list[[i]]$inputs) <=
                 length(latent.list[[nextSmallest]]$inputs))){nextSmallest<-i}

            }
            else if (is.subset(latent.list[[j]]$input,
                 latent.list[[i]]$input)){

                # Ensures that subsets are being compared for smallest/n.small
                if(is.null(smallest)){smallest<-j; nextSmallest<-i}

                if(isTRUE(length(latent.list[[j]]$inputs) <=
                 length(latent.list[[smallest]]$inputs))){smallest<-j}
                else if(isTRUE(length(latent.list[[j]]$inputs) <=
                 length(latent.list[[nextSmallest]]$inputs))){nextSmallest<-j}
            }
        }
    }

    if(is.null(smallest)){return(list(smallest=1, nextSmallest=1))}

    return(list(smallest=smallest, nextSmallest=nextSmallest))
}

there.is.a.subset <- function(latent.list, n.inputs){
    size.list <- smallest.two.subsets(latent.list,
         n.inputs=length(inputs.and.outputs$inputs))
        return(size.list[[1]]==size.list[[2]])

}


remove.subset <- function(small.set, larger.set){
    return(larger.set[!larger.set%in%small.set])
}

is.subset <- function(set.1, set.2){

    if(length(set.1)>length(set.2)){return(FALSE)}

    joint.membership <- c(0)
    for(element in set.1){
        joint.membership <- c(joint.membership + sum(set.2==element))
    }

    if(joint.membership==length(set.1)){
        return(TRUE)
    }
    else{
        return(FALSE)
    }

}

get.latents <- function(latent.structure){
    latents <- c()
    for(i in 1:length(latent.structure)){
        latents[[i]] <- latent.structure[[i]]$latent
    }
    return(latents)
}


get.inputs.via.latents <- function(latent.structure, latents){
    latents <- as.numeric(latents)
    inputs <- c()

    left.latent <- latents[1]
    right.latent <- latents[2]

    inputs <- c(latent.structure[[left.latent]]$inputs,
        latent.structure[[right.latent]]$inputs)

    return(inputs)
}

get.outputs.via.latents <- function(latent.structure, latents){
    latents <- as.numeric(latents)
    outputs <- c()

    left.latent <- latents[1]
    right.latent <- latents[2]

    outputs <- c(latent.structure[[left.latent]]$outputs,
        latent.structure[[right.latent]]$outputs)

    return(outputs)
}

find.dsep <- function(inputs, outputs, data, pval=.05, categorical=FALSE){
        variations.list <- c()
        n.inputs <- length(inputs)

        for(i in 1:n.inputs){
            variations.list[[i]] <- list(combn(inputs, m=i))
        }

        condi.sets<-variations.list

        for(i in 1:length(condi.sets)){
            dsep.set<-get.cond.combo(data, outputs, condi.sets[i], pval, categorical)
            if(!is.null(dsep.set)){
                # print(dsep.set)
                return(dsep.set)
            }
        }
    return()
}

```

```r
#TODO comment this fuction. The nested conditionals/flow control especially.
get.cond.combo <- function(data, outputs, input, pval, categorical=FALSE){

    input <- destroy.list(input)

    # Handles single input cases.
    if(is.null(dim(input))){

        for(i in 1:length(input)){

            if(isTRUE(categorical)){
                if(!ciTest(data, set=c(outputs[c(1, length(outputs))],
                (input[i])))$p.val<=pval){return(input[i])}
            }
            else{
                if(!ciTest(data, set=c(outputs[c(1, length(outputs))],
                (input[i])))$p.val<=pval){return(input[i])}
            }
        }
    }
    else{
        for(i in 1:ncol(input)){
            if(isTRUE(categorical)){

            }
            else{
            if(!ciTest(data, set=c(outputs[c(1, length(outputs))],
             (input[,i])))$p.val<=pval){return(input[,i])}
            }
        }
    return(c())
    }
}

destroy.list <- function(list.obj){
    if(class(list.obj)=="list"&&
        listDepth(list.obj)>1){

        destroy.list(unlist(list.obj, recursive=F))
    }
    else{return(list.obj[[1]])}
}

isFALSE <- function(truth.vector){return(!isTRUE(truth.vector))}
```

Bibliography

Brodie, A. (2014). Identifying endogenous latent causal structure under linearity and sparsity assumptions.

Bühn, A. and Schneider, F. G. (2008). Mimic models, cointegration and error correction: an application to the french shadow economy. Technical report, CESifo working paper.

Craig, E. and Hoskin, M. (1992). Hegel and the seven planets. *Journal for the History of Astronomy*, 23(3).

DellAnno, R. and Schneider, F. (2006). Estimating the Underground Economy by Using MIMIC Models: A Response to T. Breusch´s Critique. Technical report.

Giles, D. (1999). Measuring the Hidden Economy: Implications for Econometric Modelling. *The Economic Journal*, 109(456):370–380.

Glymour, C., Scheines, R., Spirtes, P., and Kelly, K. (1987). *Discovering Causal Structure: Artificial Intelligence, Philosophy of Science, and Statistical Modeling*. Academic Press.

Hair, J., Tatham, R., R.E., A., and Black, W. (1998). *Multivariate Data Analysis*. Prentice-Hall International.

Hastie, T., Tibshirani, R., Friedman, J., Hastie, T., Friedman, J., and Tibshirani, R. (2009). *The elements of statistical learning*, volume 2. Springer.

Hempel, C. G. (1985). Thoughts on the limitations of discovery by computer. *Logic of discovery and diagnosis in medicine*, pages 115–122.

Kant, I., Guyer, P., and Wood, A. W. (1998). *Critique of pure reason.* Cambridge University Press.

Kummerfeld, E., Ramsey, J., Yang, R., Spirtes, P., and Scheines, R. (2014). Causal clustering for 2-factor measurement models. *Proceedings of the European Conference on Machine Learning. (Forthcoming).*

Lester, L. (2008). A Multiple Indicators and Multiple Causes (MIMIC) Model of Immigrant Settlement Success.

Logothetis, N. K. (2008). What we can do and what we cannot do with fmri. *Nature*, 453(7197):869–878.

Markowetz, F., Kostka, D., Troyanskaya, O. G., and Spang, R. (2007). Nested effects models for high-dimensional phenotyping screens. *Bioinformatics*, 23(13):i305–i312.

Mayo, D. G. (1996). *Error and the growth of experimental knowledge.* University of Chicago Press.

Network, C. G. A. R. et al. (2011). Integrated genomic analyses of ovarian carcinoma. *Nature*, 474(7353):609–615.

Ríos-Bedoya, C., Pomerleau, C., Neuman, R., and Pomerleau, O. (2009). Using MIMIC Models to Examine the Relationship Between Current Smoking and Early Smoking Experiences. *Nicotine & Tobacco Research*, 11(9):1035–1041.

Shalizi, C. R. (2012). Advanced data analysis from an elementary point of view. *Preprint of book found at http://www. stat. cmu. edu/ cshalizi/ADAfaEPoV.*

Silva, R., Scheine, R., Glymour, C., and Spirtes, P. (2006). Learning the Structure of Linear Latent Variable Models. *The Journal of Machine Learning Research*, 7:191–246.

Sober, E. (1998). Black Box Inference: When Should Intervening Variables be Postulated? *The British Journal for the Philosophy of Science*, 49(3):469–498.

Spirtes, P., Glymour, C., and Scheines, R. (2000). *Causation Prediction & Search 2e*. Adaptive Computation and Machine Learning Series. MIT Press.

Tedds, L. (1998). Measuring the Size of the Hidden Economy in Canada: A Latent Variable/MIMIC Model Approach. *MA Extended Essay, Department of Economics, University of Victoria.*

Thurstone, L. (1934). The Vectors of Mind. *Psychological Review; Psychological Review*, 41(1):1.

Tresch, A. and Markowetz, F. (2008). Structure learning in nested effects models. *Statistical Applications in Genetics and Molecular Biology*, 7(1).