Overview of Causal Reasoning with *CausalR* : Hypothesis Scoring and P-value Calculation

Steven J Barrett, Glyn Bradley

January 4, 2019

Contents

1 Introduction

CausalR provides causal reasoning (causal network analysis) methods for the Bioconductor project, enabling the prediction of regulators of sets of observed endpoints (e.g. genome scale expression measurements) by integration with prior knowledge in the form of a causal interaction network. Predicted regulators (hypotheses) are assigned a score reflecting their fit with the input differiential experimental data. Two levels of signmificance are computed for each hypothesis.

Table 1 is an example of the ranked hypotheses output of *CausalR*, produced by executing functions `CreateCCG()`, `ReadExperimentalData()` then `RankTheHypotheses()` on the example causal network and experimental data used in the following text.

Node	Regulation	Score	Correct	Incorrect	Ambiguous	p-value	Enrichment p-value
Node0	1	4	5	1	1	0.114	1
Node1	1	3	3	0	0	0.121	1
Node3	1	1	1	0	0	0.500	0.875
Node5	-1	1	1	0	0	0.375	0.875
Node6	-1	1	1	0	0	0.375	0.875
Node7	-1	1	1	0	0	0.375	0.875
Node2	1	0	2	2	0	0.629	0.5
Node4	1	0	0	0	0	0.625	1
Node2	-1	0	2	2	0	0.543	0.5
Node4	-1	0	0	0	0	0.500	1
Node5	1	-1	0	1	0	1	0.875
Node6	1	-1	0	1	0	1	0.875
Node7	1	-1	0	1	0	1	0.875
Node3	-1	-1	0	1	0	1	0.875
Node1	-1	-3	0	3	0	0.957	1
Node0	-1	-4	1	5	1	0.971	1

Table 1: *CausalR* `delta=2` output for example inputs.

Reconstruction of regulatory networks from either newly predicted of previous known regulators of the input experimental data is facilitated, outputting .sif network and annotation files for visualisation in Cytposcape.

The *SCAN* (Sequential Causal Analysis of Netwroks) methodology of limited false positives in the prediction of regulators is also provided. For further background on methodological details, suggested analysis approaches and an actual biological causal network, see Bradley and Barrett [**submitted**].

The following tutorial text provides a step-by-step series of examples on using *CausalR*, together with further descriptive explanation of what is happening at each stage. For convenience, the complete set of commands, with details of further functionality, are listed in Appendix A in the form of an R script.

2 Loading Input Network and Experimental Data

After starting the R environment (either the standard R console or via an IDE such as *Rstudio, http://www.rstudio.com*), and setting the working directory to an appropriate location for accessing input files, proceed to loading the *CausalR* library,

```
library(CausalR)
```

The loading of the main package also performs

```
library(igraph)
```

since *igraph* contains network analysis functionality required by CausalR.

These triples represent the network to be used in the example processing,

```
Node0 Activates Node1
Node0    Inhibits    Node2
Node1    Activates    Node3
Node1    Activates    Node4
Node1    Inhibits    Node5
Node2    Inhibits    Node5
Node2    Activates    Node6
Node2    Activates    Node7
```

Each line triple represents a directed edge in the input network of Figure 1.

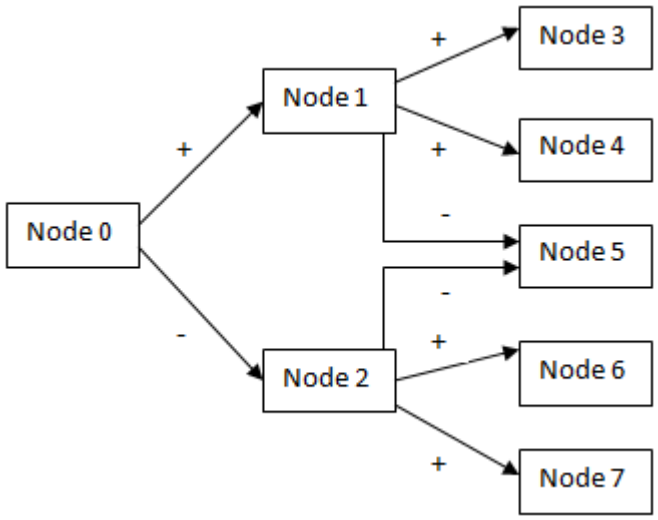


Figure 1: Example input network.

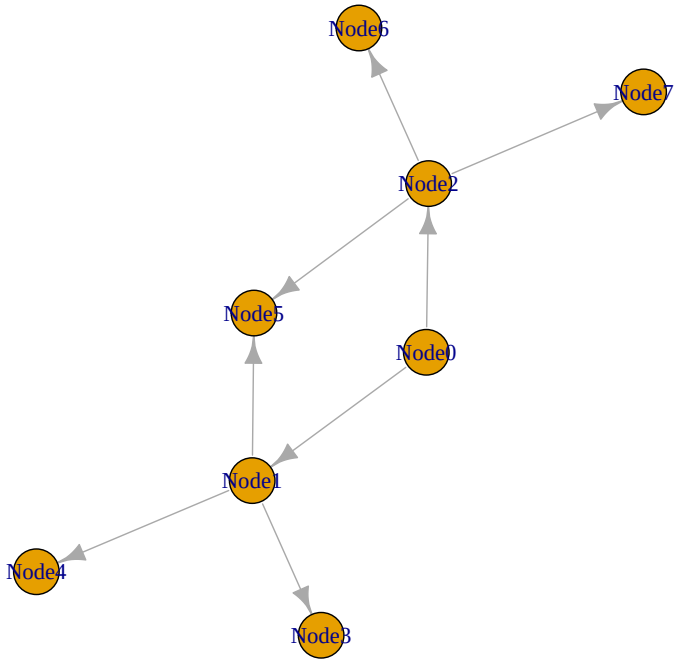
This is read from tab-separated (Cytoscape `.sif` format) file by one of two user functions, `CreateCG()` or `CreateCCG()` which respectively create and store either a causal or a computational causal graph. To understand what these are we first show how the function `CreateCG()` is used to create an internal representation of the network as a Causal Graph (CG).

```
cg <- CreateCG(system.file( "extdata", "testNetwork1.sif",
    package="CausalR"))

## [1] "File read complete - read in 8 lines. Now constructing network"
## [1] "Network has been created - now adding edge properties"
## [1] "Added weights to edges"
```

The CG is stored in the workspace as an *igraph* object, which is a particular type of object that can be used to store and process graphs in R (see *http://igraph.org/r/doc/igraph.pdf*). The Causal Graph can then be visualised via,

```
PlotGraphWithNodeNames(cg)  # producing the following graph.
```



Note that although only directional connectivity is shown, edge sign attribution of the input network is retained. Note also that *igraph* will assign internal numeric nodeIDs (for simplicity, the example network here mirrors these) that are used internally by *CausalR* functions. Most *CausalR* functions will automatically convert *igraph* nodeIDs back to node names recognisable to the user, but there are exceptions that return output with these numbers. For those outputs, the user will need to apply the `GetNodeName()`.

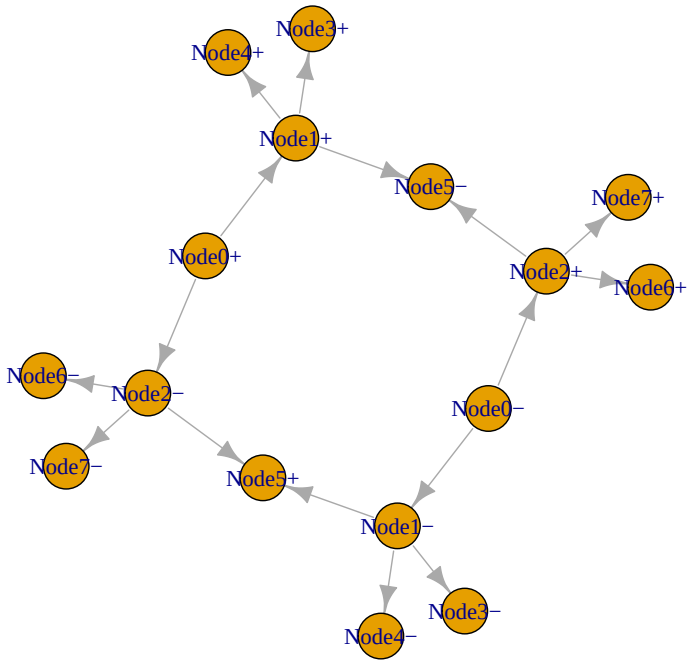
The `CreateCCG()` function creates in the workspace an *igraph* object (here assigned to "ccg") which contains a computational causal network representation of the contents of the input network.sif file.

```
ccg <- CreateCCG(system.file( "extdata", "testNetwork1.sif",
    package="CausalR"))
```

The latter is employed internally instead of a CG for processing efficiency.

A CCG contains double the nodes and vertices, to separately represent up and down regulation of each node, and these are connected according to the original edge relations from the network.sif file. The Computational Causal Graph (CCG) for the input network input can be visualised via,

```
PlotGraphWithNodeNames(ccg) # producing the following graph.
```



The input differential experimental results (used in subsequent worked examples) is as follows,

```
Node0    1
Node1    1
Node2    1
Node3    1
Node4    0
Node5    -1
Node6    -1
Node7    -1
```

In the second column, 1 represents differential activation, -1 differential inhibition and 0 indicates no differential regulation was observed for the entity represented by the node.

The inclusion of all non-differential results is critical to accurate significance calculations (in their absence *CausalR* will otherwise see them as not measured and process them accordingly).

A tab-separated file containing this content is read into the workspace using `ReadExperimentalData()`, as follows,

```
experimentalData <- ReadExperimentalData(system.file( "extdata",
                        "testData1.txt", package="CausalR"),ccg)
```

At this stage checks are performed to identify entities (nodes) not present within the CG/CCG.

These are flagged to the user and excluded from the two column experimental data matrix [*igraph* nodeID, regulation] that is stored in the workspace for subsequent processing.

3 Computing Score: Multiple Causal Hypotheses

To compute a list of score-ranked hypotheses within an interactive session, the user would employ the `RankTheHypotheses()` function as follows,

```
options(width=120)
RankTheHypotheses(ccg, experimentalData, delta=2)


## Number of Nodes to analyse: 8
##        NodeID Regulation Score Correct Incorrect Ambiguous   p-value Enrichment p-value
## Node0+      1          1     4       5         1         1 0.1142857               1.000
## Node1+      2          1     3       3         0         0 0.1214286               1.000
## Node3+      4          1     1       1         0         0 0.5000000               0.875
## Node5-      6         -1     1       1         0         0 0.3750000               0.875
## Node6-      7         -1     1       1         0         0 0.3750000               0.875
## Node7-      8         -1     1       1         0         0 0.3750000               0.875
## Node2+      3          1     0       2         2         0 0.6285714               0.500
## Node4+      5          1     0       0         0         0 0.6250000               1.000
## Node2-      3         -1     0       2         2         0 0.5428571               0.500
## Node4-      5         -1     0       0         0         0 0.5000000               1.000
## Node5+      6          1    -1       0         1         0 1.0000000               0.875
## Node6+      7          1    -1       0         1         0 1.0000000               0.875
## Node7+      8          1    -1       0         1         0 1.0000000               0.875
## Node3-      4         -1    -1       0         1         0 1.0000000               0.875
## Node1-      2         -1    -3       0         3         0 0.9571429               1.000
## Node0-      1         -1    -4       1         5         1 0.9714286               1.000
```

`RankTheHypotheses()` incorporates usability features (see Section 9) and parallelisation is supported (see Appendix B for instructions).

Alternatively, the following is used to output similar (results for both + and - hypotheses will again be generated) for a subset of nodes,

```
options(width=120)
testlist<-c('Node0','Node2','Node3')
RankTheHypotheses(ccg, experimentalData,
                              delta=2, listOfNodes=testlist)


## Number of Nodes to analyse: 3
##        NodeID Regulation Score Correct Incorrect Ambiguous   p-value Enrichment p-value
## Node0+      1          1     4       5         1         1 0.1142857               1.000
## Node3+      4          1     1       1         0         0 0.5000000               0.875
## Node2+      3          1     0       2         2         0 0.6285714               0.500
## Node2-      3         -1     0       2         2         0 0.5428571               0.500
## Node3-      4         -1    -1       0         1         0 1.0000000               0.875
## Node0-      1         -1    -4       1         5         1 0.9714286               1.000
```

`RankTheHypotheses()` takes as arguments the workspace CCG and experimental matrix object (both from the processing described above), together with a user supplied numeric value for the path length, termed delta, *delta*.

The latter represents the desired number of edges in the network the user wishes the algorithm to look forward, from the hypothesis, to observed result nodes.

With each further level searched into the network, the number of connected nodes increases roughly exponentially until the *delta* approximates the network average degree. With deltas close to or above the network average degree the majority of hypothesis nodes will be connected to most other nodes in the network, and consequently the ability of the scoring algorithm to differentiate good hypotheses from bad ones will be degraded.

`RankTheHypotheses()` calls the hidden function `Scorehypothesis()`, which computes scores reflecting the overall level of agreement between predictions based upon an individual hypothesis and the experimentally observed results for nodes within *delta* edges (i.e. to which it can be causally-linked).

4 Computing Score: Specific Causal Hypothesis

To get scores and significance summary for both up- and down-regulation hypotheses concerning an individual node, the `listOfNodes` input can be assigned to a sole node name (note that node names are case sensitive in *CausalR*),

```
options(width=120)
RankTheHypotheses(ccg, experimentalData, 2, listOfNodes='Node0')


## Number of Nodes to analyse: 1
##        NodeID Regulation Score Correct Incorrect Ambiguous   p-value Enrichment p-value
## Node0+      1          1     4       5         1         1 0.1142857                 1
## Node0-      1         -1    -4       1         5         1 0.9714286                 1
```

It may first be useful to check on node content and length of the shortest path from the proposed hypothesis (root) node to any specific (outcome) target node(s) of interest. For this the function `GetShortestPathsFromCCG()` is employed,

```
GetShortestPathsFromCCG(ccg, 'Node0', 'Node3')
```

```
node0+ node1+ node3+
```

An alternative to `RankTheHypotheses()` enables acquisition of further detail on the breakdown of scoring contributions. The `MakePredictionsFromCCG()`

function can be used directly to get scores for a particular signed hypothesis of a node,

```
predictions <- MakePredictionsFromCCG('Node0',+1,ccg,2)
predictions
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    1
## [3,]    3   -1
## [4,]    4    1
## [5,]    5    1
## [6,]    6    0
## [7,]    7   -1
## [8,]    8   -1
```

The input `node0` is the root (or hypothesis) node, +1 is the direction of regulation, ccg is the network, and the last value, 2, is the `delta` value (these inputs are explained in more detail in the following section). The predictions will be returned as above, with the *igraph* assigned node ID in the first column and the direction of regulation in the second.

The `ScoreHypothesis()` function can then be used to provide a summary of the comparisons between the predictions from the given hypothesis against the observed outcomes in the experimental data,

```
ScoreHypothesis(predictions, experimentalData)
```

```
## [1] 4 5 1 1
```

This returns 4 integer values, which (in order) are the score (the number of correct predictions minus the number of incorrect predictions), number of correct predictions, number of incorrect predictions and number of ambiguous predictions (these will be explained later within the Scoring Section).

To discover how each predicted node outcome contributes to the score, the `CompareHypothesis()` function is used to provide a comparison between experiment and prediction for each node separately,

```
GetNodeName(ccg,CompareHypothesis(predictions, experimentalData))
```

```
##   NodeID Prediction Experiment Score
## 1  Node0          1          1     1
## 2  Node1          1          1     1
## 4  Node3          1          1     1
## 7  Node6         -1         -1     1
## 8  Node7         -1         -1     1
## 5  Node4          1          0     0
## 6  Node5          0         -1     0
## 3  Node2         -1          1    -1
```

The use of function `GetNodeName()` is required here to convert *igraph* nodeIDs, output by `CompareHypothesis()`, back to user recognisable node names, as was supplied in the inputs.

Functions `CalculateSignificance()` and `CalculateEnrichmentPValue()` can be used to calculate the statistical significance of a given signed hypothesis, as detailed in the R code of Appendix A.

4.1 Hypothesis-specific CCGs

During scoring, hypothesis-specific CCGs are sequentially created and scored. Each hypothesis will be represented as the unique root node of its corresponding CCG, whilst the maximum depth for any hypothesisCCG is equal to the path length, `delta`.

Since each hypothesis observes a either upward or downward regulation, the number of individual hypothesisCCGs computed and scored will be equal to twice the number of nodes in the input network, if all possible hypotheses are tested, as this is the default behaviour for function `RankTheHypotheses()`.

Prior to the construction of each hypothesisCCG, nodes beyond `delta` edges from the hypothesis are removed from the processed network and untested nodes (those without experimental results, apart from the hypothesis node itself) are also excluded. To construct the final hypothesisCCG, regulatory relations are then applied between the hypothesis along all paths toward the remaining nodes, observing the original network edge values.

5 Scoring

To aid scoring the CCG object represents a partitioning of the set of nodes in the experimental data, represented by $T|G|$, into three sets: the positively regulated, negatively regulated and those that are unaffected. These sets are denoted as S_h+, S_h- and S_h0 respectively, as defined in [**2012a**].

Thus in the terminology of the Methods: Scoring hypotheses section of [**2012a**], the observed experimental input (as above) would be represented as :

$$\begin{aligned} G_+ &= \{\text{Node0}, \text{Node1}, \text{Node2}, \text{Node3}\} \\ G_- &= \{\text{Node5}, \text{Node6}, \text{Node7}\} \\ G_0 &= \{\text{Node4}\} \end{aligned}$$

Any node greater than `delta` edges from the hypothesis node is automatically added to S_h0, since the path length to the hypothesis is deemed too distant to include in scoring, according to user parameterisation.

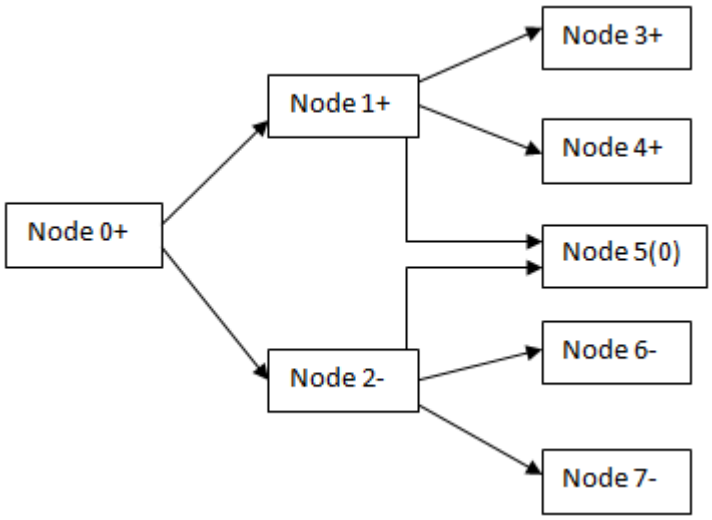


Figure 2: Network consequences of `Node0+`

5.1 Example Causal hypothesis to be Scored

In the following example calculations, the causal hypothesis to be scored is up-regulation of Node0, denoted as `Node0+`. Figure 2 above shows the downstream consequences of up-regulating Node0, as derived by traversing its hypothesisCCG, observing how the original network edge values translate the regulatory sign of the (root) hypothesis along the paths to predicted outcomes. Note that Node 5 is both positively and negatively regulated by `Node0+`.

5.2 Network Predictions based upon Hypothesis `Node0+`

The predictions $|TG_c|$ from the network (represented as per [**2012a**], Methods: Scoring hypotheses) would be:

$$\begin{aligned} S_h+ &= \{\text{Node0}, \text{Node1}, \text{Node3}, \text{Node4}\} \\ S_h- &= \{\text{Node2}, \text{Node6}, \text{Node7}\} \\ S_h0 &= \{\text{Node5}\} \end{aligned}$$

5.3 Scoring Calculation

[Example for `delta=2`] Comparing $T|G|$ to $T|G_c|$ yields the correct predictions from this hypothesis as: Node 0, Node 1, Node 3, Node 6 and Node 7. The only incorrect prediction from this hypothesis is: Node 2.

The *CausalR* scoring algorithm traces the shortest path from hypothesis to outcome node and then evaluates its sign to achieve a predicted regulation for that outcome node. The hypothesis here yields a so called "ambiguous" prediction-outcome match for Node 5, since there are two shortest paths to it and their predictions disagree, hence it doesn't contribute to the scoring.

Node 4 is also excluded from the score calculation because the observed experimental outcome for it was non-differentially regulated (unchanged) and this is a result that cannot be predicted by CausalR, as only an up or down regulation can be inferred from a signed graph.

Subtracting number of incorrect predictions (1) from the number that was correct (5) yields a score of 4 for hypothesis `Node0+`.

6 Hypothesis Score Significance Calculation

To account for the scenario where the more heavily connected hypothesis nodes have greater potential to score more highly than less connected ones, the significance of each hypothesis score needs to be individually calculated.

A lower significance (higher p-value) cut-off can then be applied as a filter to the overall scores ranking, as decided via the biological expertise of the user.

CausalR provides both p-values and enrichment p-values for this purpose. There are two alternatives for p-value calculation: a computationally expensive exact "quartic" algorithm (see section 2.4 in [**2012a**]) and (for larger inputs) a very much more efficient (minutes instead of hours, hence set as default) approximate "cubic" algorithm (see pages 6-7 in [**2012b**]).

6.1 Calculation of p-value

[Continuation of previous example for `delta=2`] Within the context of network causal reasoning (see section 2.4, [**2012a**]), we are looking for the probability of obtaining a hypothesis (node) score at least as extreme as that which was observed, assuming the null hypothesis, that the hypothesis node was not responsible for producing the experimental outcomes, is true (i.e. result is not significantly different from random).

To calculate the p-value significance of this score for hypothesis `Node0+`, these predictions must be scored against all possible predictions with $|G+| = 4$, and $|G-| = 3$.

Given that there are 8 nodes to pick each combination from, the total number of combinations is: ${}^{8}C_4 \times {}^{4}C_3 = 280$

There is only one way to get the highest possible score of 7.

There are two approaches to achieve a score of 6. The first is to choose all 4 nodes in $G+$ from the nodes in S_h+, and two of the nodes in $G-$ from S_h-: there are ${}^{4}C_4$ x ${}^{3}C_2 = 3$ ways of doing this. The other possibility is to choose 3 nodes in S_h+ and 1 in S_h0 as the nodes in $G+$. Then choose all

three of the nodes in $G-$ from S_h- there are ${}^4C_3$ x ${}^3C_3 = 4$ ways of doing this. Hence there are 7 ways of scoring 6 in total.

A score of 5 cannot be achieved.

There are two ways to score 4. (The first is to choose 3 nodes from S_h+ and 1 from S_h- as the nodes in $G+$, then choose 2 nodes from S_h- and the 1 in S_h0 as the nodes in $G-$. There are ${}^4C_3$ x ${}^3C_1 = 12$ ways of doing this. The second way is to choose 3 nodes from S_h+ and the 1 from S_h0 as the nodes in $G+$, then choose 2 nodes from S_h- and 1 in S_h+ as the nodes in $G-$. There are ${}^4C_3$ x ${}^3C_2 = 12$ ways of doing this. This means in total there are 24 ways of scoring 4.

We now have all the information we need to calculate the p-value of the hypothesis above. There are $24+7+1 = 32$ ways to score 4 or higher, hence the p-value should be $32/280 = 0.114$, as can be seen for the up-regulated node 0 hypothesis in Table 1.

6.2 Calculation of Enrichment p-value

Enrichment $p-$values, p_E, are calculated using standard approaches used for gene ontology enrichment (see [**GOEAST**])

Where $n_{++} + n_{+-} + n_{-+} + n_{--} = m$ (i.e. the number of differentially expressed genes, m in Table 2), and $T = T|G|$ is the total number of experimental transcripts; then p_E is the probability of getting m or more differential nodes from a set of T nodes given the fraction of nodes predicted to be differential.

This computation is done using the Fisher exact test (see [**Fisher**]), as implemented in the fisher.test function in base R.

The goal is to compare the actual number of differential nodes to the number of predicted ones, given the values of $q_+, q_-, q_0, n_+, n_-, n_0, n_{++}, n_{+-}, n_{-+}$ and n_{--} in Table 2 below.

Predicted Causal Network	*Measured* Differential	*Measured* Non-differential	*Measured* Total
Differentially Expressed	$n_{++} + n_{+-} + n_{-+} + n_{--} = m$	$n_{+_0} + n_{-0}$	$q_+ + q_-$
Non-differentially Expressed	$n_{0+} + n_{0-} = n_+ + n_- - m$	n_{00}	q_0
Total	$n_+ + n_-$	n_0	$T\|G\|$

Table 2: Result outcome categorisations used in significance calculations.

The probability of getting this arrangement of values is given by the hypergeometric distribution,

$$p = \frac{\binom{q_+ + q_-}{m}\binom{q_0}{n_{0+}+n_{0-}}}{\binom{|T(G)|}{n_+ + n_-}}$$

To obtain p_E, the probability of getting m or more elements in the intersection between the sets of predicted versus experimentally-observed differential nodes, needs to be calculated. The formula employed to calculate the enrichment p-value is therefore,

$$p_E = \sum_{i=m}^{q_+ + q_-} \frac{\binom{q_+ + q_-}{i}\binom{q_0}{n_+ + n_- - i}}{\binom{|T(G)|}{n_+ + n_-}}$$

6.3 Efficiency and Accuracy of Significance Calculation

As stated earlier, it is crucial to include non-differential outcomes in the experimental input for accurate significance calculations. The problem with this is that for large NGS study data compute times increase greatly and much of the added effort will be on significance calculations for poor hypotheses.

This can be avoided by resetting the `correctPredictionsThreshold` value used by the `RankTheHypotheses()` function. This is normally set to minus infinity, which means that p and p_E values will be calculated regardless of

hypothesis score. When reset to a positive value it restricts significance calculations to a subset of hypotheses with a minimum number of correct predictions (all other hypotheses will automatically get `NA` values).

```
options(width=120)
Rankfor4<-RankTheHypotheses(ccg, experimentalData, 2,
                            correctPredictionsThreshold=4)

## Number of Nodes to analyse: 8

Rankfor4   # For example output only

##        NodeID Regulation Score Correct Incorrect Ambiguous   p-value Enrichment p-value
## Node0+      1          1     4       5         1         1 0.1142857                 1
## Node1+      2          1     3       3         0         0        NA                NA
## Node3+      4          1     1       1         0         0        NA                NA
## Node5-      6         -1     1       1         0         0        NA                NA
## Node6-      7         -1     1       1         0         0        NA                NA
## Node7-      8         -1     1       1         0         0        NA                NA
## Node2+      3          1     0       2         2         0        NA                NA
## Node4+      5          1     0       0         0         0        NA                NA
## Node2-      3         -1     0       2         2         0        NA                NA
## Node4-      5         -1     0       0         0         0        NA                NA
## Node5+      6          1    -1       0         1         0        NA                NA
## Node6+      7          1    -1       0         1         0        NA                NA
## Node7+      8          1    -1       0         1         0        NA                NA
## Node3-      4         -1    -1       0         1         0        NA                NA
## Node1-      2         -1    -3       0         3         0        NA                NA
## Node0-      1         -1    -4       1         5         1        NA                NA

subset(Rankfor4,Correct>=4)

##        NodeID Regulation Score Correct Incorrect Ambiguous   p-value Enrichment p-value
## Node0+      1          1     4       5         1         1 0.1142857                 1
```

This can also be useful where the user wishes to employ the more computationally expensive exact quartic (1a) algorithm on a subset of interesting hypotheses, instead of using the default cubic (1b) approximation algorithm.

7 Sequential Causal Analysis of Networks (SCAN)

Given that true causal regulators are likely to regulate nodes at various distances along the pathways in which they feature, the use of a range of different deltas can be employed to improve enrichment for them. This was the thinking behind Sequential Causal Analysis of Networks (*SCAN*) approach introduced in Bradley and Barrett [**submitted**].

By invoking the `runSCANR()` function, the `RankThehypotheses()` function is iteratively run for a range of path lengths (`deltas`), starting from `delta=1` up to the value of the user-supplied variable `NumberOfDeltaToScan` (default is 5), to discover nodes that consistently score in the highest `topNumGenes` (default is 150) across all SCAN deltas run.

So for example given the following experimental data,

```
NodeA    1
NodeB    -1
NodeC    1
NodeD    -1
NodeE    1
NodeF    -1
NodeG    1
NodeH    -1
NodeI    1
NodeJ    -1
```

and a more complex network, as in the `Cytoscape` (*http:www.cytoscape.org*) depiction of `testnet.sif` in Figure 3 below).

The `runSCANR()` function will first write the number of nodes to analyse to the R console window (this will be repeated for each delta run). Then the next `NumberOfDeltaToScan` lines output will list the top hypotheses from each delta that was run, whilst the final line gives the SCAN results for the top hypotheses in common between these deltas.

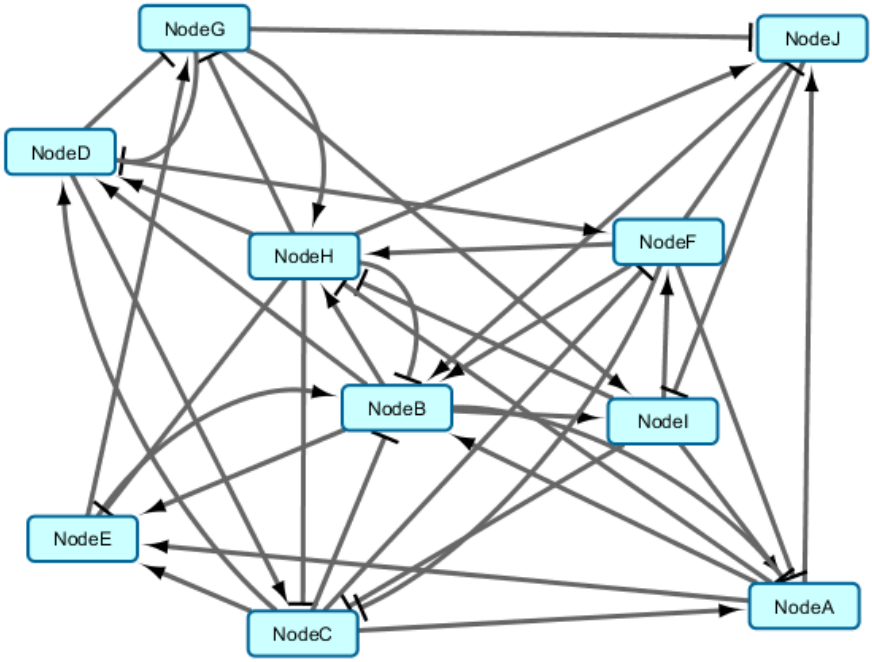


Figure 3: Network input for SCAN Example.

```
runSCANR(ccg, experimentalData, numberOfDeltaToScan=2, topNumGenes=4,
         correctPredictionsThreshold=1,writeResultFiles = TRUE,
         writeNetworkFiles = "none",quiet=TRUE)
```

```
## $testList
##        1      2
## 1 Node1+ Node0+
## 2 Node0+ Node1+
## 3 Node3+ Node3+
## 4 Node5- Node5-
##
## $uniqueNodes
##    names maxDelta
## 1 Node1+        2
## 2 Node0+        2
## 3 Node3+        2
## 4 Node5-        2
```

```
##
## $commonNodes
## [1] "Node1+" "Node0+" "Node3+" "Node5-"
```

`testList` contains the hypotheses found at each delta, as restricted by topNumGenes.

`uniqueNodes` contains all hypotheses, with the maximum delta that returned them.

`commonNodes` contains only the hypotheses subset ranked in the first topNumGenes of all hypotheses across ALL deltas run, and these are ranked in order found at the highest delta.

Additionally, under the above settings, three files are created in the current R workspace:

1. A common nodes file: CommonNodes-testNetwork1-testData1-deltaScanned2-top4.txt
Containing:

```
Node1+
Node0+
Node3+
Node5-
```

This content is equivalent to that in `commonNodes` above.

2. A top nodes file: TopNodes-testNetwork1-testData1-deltaScanned2-top4.txt
Containing:

```
names  maxDelta
Node1+ 2
Node0+ 2
Node3+ 2
Node5-2
```

This content is equivalent to that in uniqueNodes above.

3. A ResultsTable: ResultsTable-NetworkFileName-ExptlDataFileName-deltaused.txt containing RankTheHypotheses()output for the highest delta run.

Note that the above files are auto-named according to the input files and content-defining parameters (more details in Section 9).

Hypothesis-specific regulatory networks can be generated directly by `runSCANR()` , from the commonNode list, however the returned number of common hypotheses won't apriori be known (see Section 8 for details of direct and indirect hypothesis regulatory network file creation).

From experiments with *CausalR* on signals for known pathways (see Bradley and Barrett [**submitted**]), setting of the `topNumGenes` parameter to represent 1 percent of the number of nodes present in the input network provides a reliable starting heuristic for identifying the true regulators.

A SCAN run covering 1-5 deltas can typically take overnight with large inputs, however parallelisation is supported via setting `doParallel=TRUE`). Various additional steps can also be taken to improve run times:

1. Use of the R compiler (see Appendix B)
2. Excluding scoring of hypotheses for non-differential nodes,

```
AllData<-read.table(file="testData1.txt", sep = "\t")
DifferentialData<-AllData[AllData[,2]!=0,]
write.table(DifferentialData, file="DifferentialData.txt",
    sep="\t", row.names=FALSE, col.names=FALSE, quote=FALSE)

runSCANR(ccg, ReadExperimentalData("DifferentialData.txt", ccg),
        NumberOfDeltaToScan=2,topNumGenes=100,
        correctPredictionsThreshold=2)
```

3. Suppression of significance calculations. Note that these will be inaccurate and unused anyway if non-differential results are excluded, as with *SCAN* we are relying more upon biological logic rather statistics. As this is the usual scenario, the `correctPredictionsThreshold` parameter is set with a default value of Inf (infinity, so as to not compute

p-values, substituting NAs for them. Scoring and rankings will still be computed for all hypotheses). `correctPredictionsThreshold` can be set to a lower value if accurate p-values are required for a partial or full set of hypotheses scored.

Accurate p-values and pE-values can also later be computed separately for the subset of interesting hypotheses coming from *SCAN* using,

```
testlist<-c('Node0','Node3','Node2')
RankTheHypotheses(ccg, experimentalData,2,listOfNodes=testlist)
```

which gives up- and down-regulated hypothesis results for each named input node in the testlist.

Alternatively, for an individual signed node hypothesis, the user can start from `MakePredictionsFromCCG()` as described earlier under section "Calculating Score for a Specific Causal Hypothesis".

8 Visualising the Network of Nodes explained by a Specific Hypothesis

Hypothesis-specific regulatory sub-networks can be generated indirectly (i.e. after running SCAN) for individual hypotheses or all hypotheses, as well as directly via `runSCANR()` itself.

8.1 Indirect Individual Hypothesis Network Generation

The function `WriteExplainedNodesToSifFile()` allows production of a network of explained nodes, for a particular hypothesis, in .sif file format for visualising in Cytoscape. The function is called as follows:

```
WriteExplainedNodesToSifFile("Node1", +1,ccg,experimentalData,delta=2)
```

Unless directed elsewhere, this creates six new files in the working directory:

```
corExplainedNodes-testNetwork1-testData1-delta2-Node1+.sif
corExplainedNodes-testNetwork1-testData1-delta2-Node1+_anno.txt

incorExplainedNodes-testNetwork1-testData1-delta2-Node1+.sif
incorExplainedNodes-testNetwork1-testData1-delta2-Node1+_anno.txt

ambExplainedNodes-testNetwork1-testData1-delta2-Node1+.sif
ambExplainedNodes-testNetwork1-testData1-delta2-Node1+_anno.txt
```

The correctly explained nodes (corExplainedNodes) .sif file contains interactions representing all of the explained nodes (i.e. nodes that have the same direction of regulation in both the network and the experimental data, as predicted under the hypothesis of up-regulated Node1+, with content structured as follows,

```
Node1  Activates Node1
Node1 Activates Node3
Node1 Inhibits Node5
```

The corresponding (corExplainedNodes) annotation file contains the respective explained node regulation values,

```
NodeID  Regulation
Node1 1
Node3 1
Node5 -1
```

These files can be directly loaded into Cytoscape to visualise the regulatory network of hypothesis-explained nodes, such as in figure 4.

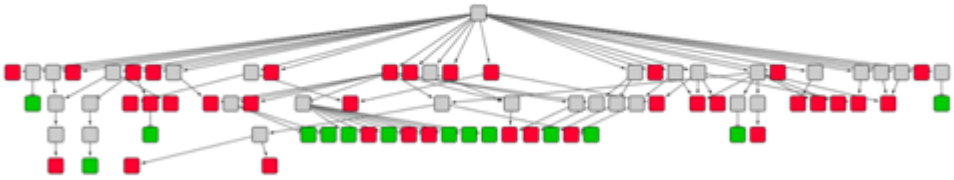


Figure 4: Example hypothesis-specific regulatory network (Cytoscape view).

8.2 Indirect Network Generation for All Hypotheses found Common across SCAN Deltas

An additional function, `WriteAllExplainedNodesToSifFile()`, accepts the `commonNodes` component of the `scanResults` workspace object (assigned to store the `runSCANR()` output) to generate networks for all the hypotheses found in common across all SCAN delta settings:

```
scanResults <- runSCANR(ccg, experimentalData, numberOfDeltaToScan=2,
  topNumGenes=4,correctPredictionsThreshold=1,
  writeResultFiles = FALSE, writeNetworkFiles = "none",quiet=FALSE)
WriteAllExplainedNodesToSifFile(scanResults, ccg, experimentalData,
  delta=2, correctlyExplainedOnly = TRUE, quiet = TRUE)
```

This produces pairs of `corExplainedNodes` .sif and annotation network files for all hypotheses (Node0+, Node1+, Node3+ and Node5-), in each case including only those experimental data nodes that are correctly explained by that hypothesis (i.e. as defined by using writeNetworkFiles = "correct" instead of "all").

8.3 Direct Network Generation for All Hypotheses found Common across SCAN Deltas

The following command shows how to generate the same networks as in the latter example above, only directly from `runSCANR()`

```
runSCANR(ccg, experimentalData, numberOfDeltaToScan=2,
  topNumGenes=4,correctPredictionsThreshold=1,quiet=TRUE,
  writeResultFiles = TRUE, writeNetworkFiles = "correct")
```

Files generated in this way get the full input auto-naming, i.e. corExplainedNodes-testNetwork1-testData1-delta2-top4-Node0+.sif, but care needs to be taken to sensibly try to restrict the set of common nodes produced by `runSCANR()` otherwise there may be very many!

9 Additional Usability Features

9.1 Automated File Naming

The `RankTheHypotheses()` and `runSCANR()` functions will auto-name their primary output files with names that identify all relevant inputs and parameterisations employed.

These filenames will therefore contain the input network filename, input experimental signal filename and the input `delta` parameter value.

The results table produced by `runSCANR()` will also feature the `topNumGenes` parameter value.

All network files output by functions `WriteExplainedNodesToSifFile()` and `WriteAllExplainedNodesToSifFile()` will contain in their filename the specified hypothesis node name with the +/- direction of regulation.

9.2 Output Suppression, Re-direction and Quiet Mode

All the functions mentioned in the above section will by default output to the current R working directory, however they can to configured to send outputs to a named directory specified via the `outputDir` parameter (this can accept a path with directory name, if the folder is not to be in working directory). If the specified directory doesn't exist it will be created.

`RankTheHypotheses()` and `runSCANR()` functions have a quiet flag that is set to FALSE by default. Changing this to TRUE will suppress output of some or all results to the screen (depending upon other settings;useful for batch scripted runs).

Similarly, there are flags to suppress results files creation:

`RankTheHypotheses()` has a writeFile flag set to TRUE so as to write a file by default. When set to `FALSE` no output files are created (useful when outputs are to be stored a workspace object, i.e. via assignment using the `<-` assignment operator.

`runSCANR()` has a `writeResultsFiles` flag which works in a similar way to control writing of the usual SCAN text file outputs. The default `TRUE` setting will result in three text files. `FALSE` will switch this off, i.e. for when results are directed into the R workspace via assignment to an object.

`RunSCANR()` also accepts a further `writeNetworkFiles` flag for control of automated regulator network files generation. This is set to `all` by default.

9.3 Operating Systems

`CausalR` as a Bioconductor package has been built for Linux, PC and Mac operating systems. Operation and functionality is identical regardless of the OS used.

9.4 Parallelisation

The `RankTheHypotheses()` and `RunSCANR()` functions are fully parallelised. Parallelisation can be switched-on by including `doParallel=TRUE` in their parameters. The user can optionally specify the number of cores by setting the `numCores` flag to an integer value (default is `NULL`, whereby the number of available cores will be automatically detected) . See `Appendix B` for further advice on speeding-up `CausalR` processing.

9.5 Network Filtering

The `CreateCCG()` function can read an optional node inclusion text file containing a list of node names to filter the network interactions against. The direction of filtering is controlled by an optional `excludeNodesInFile` parameter to specifiy including (`excludeNodesInFile=FALSE`) only those text file nodes (and their interactions) in the `CCG` network object, or to (default setting) exclude those nodes from all interactions in the `CCG`.

```
CreateCCG(filename, nodeInclusionFile = 'NodesList.txt',
                                    excludeNodesInFile = TRUE)
```

Note that both the influencing and influenced nodes will be searched against across all interactions within the input network. Individual interactions will be included or excluded if one (or both) node name(s) match against any specified within the node inclusion file.

References

[1] Bradley, G. and Barrett, S.J. *(submitted) CausalR*: extracting mechanistic sense from genome scale data. Bioinformatics, application note.

[2] Chindelevitch, L. *et al.* (2012a) Causal reasoning on biological networks: interpreting transcriptional changes. Bioinformatics, 28(8):1114-21.

[3] Chindelevitch, L. *et al.* (2012b) Assessing statistical significance in causal graphs (Methodology article). BMC Bioinformatics, 13:35- 48.

[4] Fischer, R. A. (1970) Statistical Methods for Research Workers. Oliver & Boyd.

[5] Zheng, Q. and Wang, X-J. (2008) GOEAST: a web-based software toolkit for Gene Ontology enrichment analysis. Nucleic Acids Research, 36, W358–W363.

Appendices

A Command Sequence Listing for Examples

```
# Set-up
library(CausalR)
library(igraph)

# Load network, create CG and plot
cg <- CreateCG('testNetwork1.sif')

PlotGraphWithNodeNames(cg)
```

```
# Load network, create CCG and plot
ccg <- CreateCCG(system.file( "extdata", "testNetwork1.sif",
    package="CausalR"))
```

```
PlotGraphWithNodeNames(ccg)
```

```
# Load experimental data
experimentalData <- ReadExperimentalData(system.file( "extdata",
                        "testData1.txt", package="CausalR"),ccg)
```

```
# Make predictions for all hypotheses, with pathlength set to 2.
RankTheHypotheses(ccg, experimentalData, 2)
```

```
# Make predictions for all hypotheses, running in parallel
# NOTE: this requires further set-up as detailed in Appendix B.
RankTheHypotheses(ccg,experimentalData,delta,doParallel=TRUE)
```

```
# Make predictions for a single node (results for + and -
# hypotheses for the node will be generated),
RankTheHypotheses(ccg, experimentalData,2,listOfNodes='Node0')
```

```
# Make predictions for an arbitrary list of nodes (gives results
# for up- and down-regulated hypotheses for each named node),
testlist <- c('Node0','Node3','Node2')
RankTheHypotheses(ccg, experimentalData,2,listOfNodes=testlist)
```

```
# An example of making predictions for a particular signed hypo-
# -thesis at delta=2, for up-regulated node0, i.e.node0+.
# (shown to help understanding of hidden functionality)
predictions<-MakePredictionsFromCCG('Node0',+1,ccg,2)
GetNodeName(ccg,CompareHypothesis(predictions,experimentalData))
```

```
# Scoring the hypothesis predictions
ScoreHypothesis(predictions,experimentalData)
```

```
# Compute statistics required for Calculating Significance
# p-value
Score<-ScoreHypothesis(predictions,experimentalData)
CalculateSignificance(Score, predictions, experimentalData)
PreexperimentalDataStats <-
    GetNumberOfPositiveAndNegativeEntries(experimentalData)

    #this gives integer values for n_+ and n_- for the
    #experimental data,as shown in Table 2.

    PreexperimentalDataStats

    # add required value for n_0, number of non-differential
    # experimental results,
    experimentalDataStats<-c(PreexperimentalDataStats,1)
    # then use,
    AnalysePredictionsList(predictions,8)
    # ...to output integer values q_+, q_- and q_0 for
    #    significance calculations (see Table 2)
    # then store this in the workspace for later use,
    predictionListStats<-AnalysePredictionsList(predictions,8)
```

```
# Compute Significance p-value using default cubic algorithm
CalculateSignificance(Score,predictionListStats,
    experimentalDataStats, useCubicAlgorithm=TRUE)
# or simply,
CalculateSignificance(Score,predictionListStats,
    experimentalDataStats)
# as use cubic algorithm is the default setting.
```

```
# Compute Significance p-value using default quartic algorithm
CalculateSignificance(Score,predictionListStats,
                experimentalDataStats,useCubicAlgorithm=FALSE)
```

```
# Compute enrichment p-value
CalculateEnrichmentPvalue(predictions, experimentalData)
```

```
# Running SCAN whilst excluding scoring of hypotheses for non-
# -differential nodes
AllData<-read.table(file="testData1.txt", sep="\t")
DifferentialData<-AllData[AllData[,2]!=0,]
write.table(DifferentialData, file="DifferentialData.txt",
    sep="\t",row.names=FALSE, col.names=FALSE, quote=FALSE )

runSCANR(ccg, ReadExperimentalData("DifferentialData.txt", ccg),
        NumberOfDeltaToScan=3, topNumGenes=100,
        correctPredictionsThreshold=3)
```

```
# Indirect Individual Hypothesis Network Generation (after running SCAN)
WriteExplainedNodesToSifFile("Node1", +1,ccg,experimentalData,delta=2)
```

```
# Indirect Network Generation for All Hypotheses (after running SCAN)
scanResults <- runSCANR(ccg, experimentalData, numberOfDeltaToScan=2,
  topNumGenes=4,correctPredictionsThreshold=1,
  writeResultFiles = FALSE, writeNetworkFiles = "none",quiet=FALSE)
WriteAllExplainedNodesToSifFile(scanResults, ccg, experimentalData,
  delta=2, correctlyExplainedOnly = TRUE, quiet = TRUE)
```

```
# Direct Network Generation for All Hypotheses (whilst running SCAN)
runSCANR(ccg, experimentalData, numberOfDeltaToScan=2,
  topNumGenes=4,correctPredictionsThreshold=1,quiet=TRUE,
  writeResultFiles = TRUE, writeNetworkFiles = "correct")
```

B Speeding-up CausalR Processing

The runtimes for `RankTheHypotheses()` increase with the size of network and experimental input. This can be offset via the use of parallelisation or the R (JIT) byte code compiler (no additional package installations are needed to use these as they are integral to the latest versions of base R). Note that run times will not be improved for very small examples where the cost of setting-up paralleisation or JIT outweighs its benefit.

Parallelisation is recommended for architectures featuring a multicore processor(s), whilst JIT is advised for older single or dual core machines.

Running `RankTheHypotheses()` in parallel :

Parallel processing can be activated via the `doParallel` flag,

```
RankTheHypotheses(ccg,experimentalData,delta,doParallel=TRUE)
```

By default the function will attempt to detect the number of available processer cores and use all of them bar one.

Alternatively, the number of cores to use can be set directly via the `numCores` flag,

```
RankTheHypotheses(ccg,experimentalData,delta,
                                    doParallel=TRUE, numCores=3)
```

To get the number of processor cores on their system:

Linux users can use the `lscpu` or `nproc -all` commands. In the absence of user privileges for these, try `grep processor /proc/cpuinfo`. The Linux top command can be used to check existing CPU utilisation.

Windows users can open a DOS window and type,

```
WMIC CPU Get DeviceID,NumberOfCores,NumberOfLogicalProcessors
```

or `WMIC CPU Get /Format:List` can be used.

Alternatively, they can look in the 'CPU Usage History' under the Windows Task manager 'Performance' tab.

Running `RankTheHypotheses()` with JIT compilation :

In order to activate the R byte code compiler functionality, R must be started with an additional flag command, R_COMPILE_PACKAGES=1. This can be done from the *Linux* or *Windows* command line when invoking R.

Alternatively, prior to starting R from the desktop icon on a *Windows* machine, place the R_COMPILE_PACKAGES=1 after the path to Rgui.exe in the *Target* field, under the *Shortcut* tab of the R icon *Properties* (accessed by right mouse click on the R icon).

Upon starting R, from double clicking the icon, the commands to use prior to loading *CausalR* are:

```
library(compiler)
enableJIT=3
```

When set to greater than zero, `enableJIT` invokes Just-In-Time (JIT) compilation, whilst $enableJIT = 0$ disables it. JIT can also be enabled by starting R with the environment variable `R_ENABLE_JIT` set to 1, 2 or 3, depending upon the level required. *CausalR* works with the level set to 3.

For further details see, *http://www.inside-r.org/r-doc/compiler/compile* or, alternatively, type ??*compiler* :: *compile* at the R command prompt to pull up the relevant R help page.

The JIT compiler compiles the package the first time it is run. Therefore, in order to achieve speed-up, the user needs to run a small *CausalR* example prior to running a very large one. JIT can also be used for medium sized inputs where the user wishes to run the computationally-expensive exact p-value calculation algorithm.

Note: when running very large networks/input the memory available to R may need reconfiguration (this is explained in the R documentation, but one easy way is to start-up R with the command line flag, -max-mem-size=????M, where ???? is a value in KB such that 4000M = 4MB (this can

also be inserted in the *Windows* icon *Target* field, see above). Use of 64bit R is also recommended in order to be able to address sufficient memory.