DREAM4: Simulated mRNA Expression Data for Assessing Network Inference Software

Paul Shannon

November 1, 2018

Contents

1 Introduction

The *Dialogue for Reverse Engineering Assessments and Methods* (*DREAM*) (`http://www.the-dream-project.org`) organizes annual challenges in which systems biology questions are posed, solutions invited from interested groups, and winners selected. As with the the CASP (`http://predictioncenter.org`) protein structure contest, DREAM poses questions to which the sponsors know the solution, but the contestants do not. DREAM4 challenges fall into two categories: cellular network inference and quantitative model building.

This vignette uses one of the simpler 2009 DREAM4 network inference challenges to introduce Bioconductor users to the inference of genetic regulatory networks from gene expression (and possibly additional) data. Synthetic gene expression data, generated by software, and based upon patterns found in model organisms, provides the basis for inference in the examples presented here. Ancillary data may be helpful when it is available and when, as with networkBMA, the inference package supports it. Both of these scenarios are demonstrated below.

At present, we discuss only the Bioconductor package *networkBMA* (which implements the method described in Lo et al. (2012)). Subsequent revisions of this vignette will include more network inference packages.

A crucial feature of the DREAM4 challenge is that the underlying network (the *gold standard*) is distributed along with the simulated expression data. This allows us to assess the quality of any inference made with the data. The gold standard matrix was withheld during the original challenge, which took place in 2009, and which is described in full here:

`http://wiki.c2b2.columbia.edu/dream/index.php/D4c2`

There were three sub-challenges. Quoting from the DREAM4 website:

- Wild-type, knockouts, knockdowns, multifactorial perturbations, and time series simulated expression data, in five networks of size 10. Participants are challenged to predict the directed unsigned topology of these networks.
- similar to the first one, except that the five networks are of size 100.
- five networks of size 100. In this challenge, we assume that extensive knockout / knockdown or time series experiments can't be performed. Instead, different variations of the network can be observed (e.g., samples from different patients). Thus, only the multifactorial perturbation dataset described below is provided.

2 The Data

2.1 As distributed by DREAM

The simulated expression data is generated by GeneNetWeaver (`http://gnw.sourceforge.net/`) based on network and gene expression characteristics of two well-studied sytems, E.coli and S.cerevisiae. Gene regulatory patterns found in these organisms are used to construct the gold standard network of regulators and targets, from which simulated expression data is then generated. Successful inference consists of reverse-engineering the gold standard network from the simulated expression data.

The DREAM4 dataset is distributed by the DREAM project in multiple directories, each containing a gold standard network and a number of tab-delimited files with expression data generated under different (simulated) conditions: wildtype steady state, time series after perturbation, knockouts, knockdowns, dual knockouts, and "multifactorial". Drawing again from the DREAM4 description:

- Wild-type: the steady-state levels of the unperturbed network
- Time series data: The files **timeseries.tsv* contain time courses showing how the network responds to a perturbation and how it relaxes upon removal of the perturbation. For networks of size 10 we provide 5 different time series, for networks of size 100 we provide 10 time series. Each time series has 21 time points. The initial condition always corresponds to a steady-state measurement of the wild-type. At t=0, a perturbation is applied to the network as described below. The first half of the time series (until t=500) shows the response of the network to the perturbation. At t=500, the perturbation is removed (the wild-type network is restored). The second half of the time series (until t=1000) shows how the gene expression levels go back from the perturbed to the wild-type state. In contrast to the multifactorial perturbations described in the previous section, which affect all the genes simultaneously, the perturbations applied here only affect about a third of all genes, but basal activation of these genes can be strongly increased or decreased. For example, these experiments could correspond to physical or chemical perturbations applied to the cells, which would cause (via regulatory mechanisms not explicitly modeled here) some genes to have an increased or decreased basal activation. The genes that are directly targeted by the perturbation may then cause a change in the expression level of their downstream target genes.
- Multifactorial data: The files **multifactorial.tsv* contain steady-state levels of variations of the network, which are obtained by applying multifactorial perturbations to the original network. Each line gives the steady state of a different perturbation experiment, i.e., of a different variation of the network. One may think of each

experiment as a gene expression profile from a different patient, for example. We simulate multifactorial perturbations by slightly increasing or decreasing the basal activation of all genes of the network simultaneously by different random amounts.

- Knockout and knockdown data are NxN matrices in which the expression of all genes is measured, as the expression of each gene in turn is eliminated (for knockouts) or substantially reduced (for knockdowns).

- Dual knockouts consist of simulating each of the five networks in which two gene are knocked-out simultaneously. Gene expression data for dual knockouts is not provided to the participants. Instead, participants may predict steady-state levels for dual knockouts in the bonus round described in the previous section. The files *dualknockouts_indexes.tsv indicate the pairs of genes for which a dual knockout should be predicted. For example, the line labeled "6 8" means that participants should predict the steady-state of the network after knocking out genes 6 and 8. For networks of size 10 we ask for predictions for 5 dual knockout experiments, for networks of size 100 we ask for 20 predictions.

The data sets are complete, in the sense that the expression levels of all regulators and all of their targets are known. This advantage is offset by a corresponding disadvantage: inasmuch as the synthetic expression data is not from an actual biological system, various sorts of prior and related information about gene regulation is not avaialable. Transcription factor binding motifs, ChIP-seq, DNAseI footprinting, and methylation data simply do not exist.

2.2 As repackaged into Bioc *RangedSummarizedExperiment*s

There are ten artifical gene regulatory datasets in DREAM4, five describing networks of 10 nodes, five describing networks of 100. These networks are all defined by a gold standard adjacency matrix. All networks are accompanied by wild-type, timeseries, knockout, dual-knockout, and knockdown simulated expression data. The hundred-node networks also have multifactorial data. (In the DREAM4 distribution, the multifactorial data is separated out into five additional directories; we have added them to the data object which has the corresponding gold standard matrix and expression data.)

We use the Bioconductor *RangedSummarizedExperiment* class from the *SummarizedExperiment* package to store the DREAM4 data. We provide ten instances, one for each simulated data set. Each can be loaded via a `data` statement, for instance:

```
> data(dream4_010_01)
```

We examine this dataset, loading its *RangedSummarizedExperiment* object into an R session, then getting a brief summary via the `show` command.

But first, some initializations: a convenience function for printing, and a command to set the display width for subsequent R output, so that it is visible within the printed version of this document.

```
> printf <- function(...)print(noquote(sprintf(...)))
> options(width=60)
```

```
> library(DREAM4)
> data(dream4_010_01)
> show(dream4_010_01)
```

```
class: RangedSummarizedExperiment
dim: 10 136
metadata(2): goldStandardAdjacencyMatrix
  doubleKnockoutGenePairs
assays(1): simulated
rownames(10): G1 G2 ... G9 G10
rowData names(0):
colnames(136): wt perturbation.1.t0 ... MF.9 MF.10
colData names(0):
```

This tells us that `dream4_010_01` has

- One assay matrix called *simulated* with 10 rows and 136 columns
- 10 row names: *G1 - G10*
- 136 column names: *wt*, *perturbation.t0*, *... MF.9*, *MF.10*
- No metadata for rows or columns (*rowRanges* and *colData* are both empty)
- Two objects in *metadata*: the gold standard matrix, and double knockout data

The standard Bioconductor *RangedSummarizedExperiment* class, when used to hold real experimental data, encourages the storage of experiment metadata with the experimental measurements. This facilitates reproducibility, reanalysis and simplified data storage. However this class is, in one regard, a slightly awkward fit to the DREAM4 data: the timeseries, knockout, knockdown, double knockout and multiparameter measurements all have different dimensions. *RangedSummarizedExperiments* can only "stack up" multiple measurement arrays if their dimensions are the same. We have worked around this

restriction by combining all of the matrices in each simulation into a single `assay` matrix. We use column names which allow for the easy extraction of the original matrices, as demonstrated below.

We now extract the full 10 x 136 combined expression matrix from the first (and only) element of the list in the `assays` slot, displaying the first 10 rows and 3 columns to give a sense of the data, along with a sampling of the column names:

```
> mtx.all <- assays (dream4_010_01)[[1]]
> dim(mtx.all)
```

```
[1]  10 136
```

```
> mtx.all[1:10, c(1,2,126)]
```

```
           wt perturbation.1.t0    G10.kd
G1  0.6046824         0.6665114 0.6891057
G2  0.1231706         0.1272186 0.1386172
G3  0.3287936         0.3550646 0.4537613
G4  0.6047832         0.7745716 0.7476172
G5  0.1464854         0.1004299 0.0827235
G6  0.3258775         0.2754930 0.3004501
G7  0.4967330         0.6067846 0.6386655
G8  0.6496271         0.7430983 0.6910927
G9  0.6140849         0.6656366 0.8119234
G10 0.7481792         0.6950638 0.2645752
```

```
> set.seed(37)
> print(colnames(mtx.all)[sort(sample(1:ncol(mtx.all), 16))])
```

```
 [1] "wt"
 [2] "perturbation.1.applied.t450"
 [3] "perturbation.1.removed.t850"
 [4] "perturbation.2.applied.t200"
 [5] "perturbation.3.applied.t100"
 [6] "perturbation.3.applied.t150"
 [7] "perturbation.3.applied.t300"
 [8] "perturbation.3.applied.t450"
 [9] "perturbation.4.applied.t100"
```

```
[10] "perturbation.4.applied.t500"
[11] "perturbation.5.applied.t50"
[12] "perturbation.5.applied.t250"
[13] "perturbation.5.applied.t450"
[14] "perturbation.5.removed.t550"
[15] "G4.ko"
[16] "G6.ko"
```

3 The networkBMA Package

3.1 Overview

networkBMA (Regression-based network inference using Bayesian Model Averaging) is a Bioconductor package

`http://www.bioconductor.org/packages/release/bioc/html/networkBMA.html`

designed to work with time-series expression data with the capability to integrate prior knowledge. We will demonstrate networkBMA using the first the five DREAM4 10-node time-series simulated expression sets. One of the strengths of networkBMA is its ability to incorporate prior knowledge in the inference process. We will illustrate the gains such knowledge confers: the first inference run uses no priors, while the second run does.

The gold standard adjacency matrix is stored as the first element in the metadata (or "metadata" slot) of the RangedSummarizedExperiment:

```
> mtx.goldStandard <- metadata (dream4_010_01)[[1]]
```

We want to extract only one of the ten perturbation timeseries included in `dream4_010_01`. Examine these self-describing column names:

```
> grep("perturbation.1.", colnames(mtx.all), v=T, fixed=TRUE)
```

```
 [1] "perturbation.1.t0"
 [2] "perturbation.1.applied.t50"
 [3] "perturbation.1.applied.t100"
 [4] "perturbation.1.applied.t150"
 [5] "perturbation.1.applied.t200"
 [6] "perturbation.1.applied.t250"
```

```
 [7] "perturbation.1.applied.t300"
 [8] "perturbation.1.applied.t350"
 [9] "perturbation.1.applied.t400"
[10] "perturbation.1.applied.t450"
[11] "perturbation.1.applied.t500"
[12] "perturbation.1.removed.t550"
[13] "perturbation.1.removed.t600"
[14] "perturbation.1.removed.t650"
[15] "perturbation.1.removed.t700"
[16] "perturbation.1.removed.t750"
[17] "perturbation.1.removed.t800"
[18] "perturbation.1.removed.t850"
[19] "perturbation.1.removed.t900"
[20] "perturbation.1.removed.t950"
[21] "perturbation.1.removed.t1000"
```

The man page for *networkBMA* `?networkBMA` explains that there are two required and eight optional parameters. We will start with the required two, `data`, and `nTimePoints`, to which we add the optional `self`, by which we specify that there are no self-edges (no self-regulating genes) in this data set.

```
       data: A matrix whose columns correspond to variables or genes and
             whose rows correspond to the observations at different time
             points.

nTimePoints: The number of time points at which expression measurements
             are available.  The number of columns in 'data' should be a
             multiple of 'nTimePoints', which could be greater than 1 if
             there are replicates.
```

To prepare the data for `networkBMA` we need to extract one timeseries of the ten offered, and then transpose the result into a matrix in the form expected by `networkBMA`.

```
> ts1.columns <- grep("perturbation.1.", colnames(mtx.all), fixed=TRUE)
> mtx.ts1 <- t(mtx.all[, ts1.columns])
> print(mtx.ts1[1:3, 1:3])
```

```
                                  G1        G2        G3
perturbation.1.t0           0.6665114 0.1272186 0.3550646
perturbation.1.applied.t50  0.3257748 0.1218223 0.3464115
perturbation.1.applied.t100 0.1775012 0.0443587 0.5712888
```

Extract the gold standard matrix, convert it successively to a graphAM, then a graphNEL, and display it with RCytoscape.

```
> print(names(metadata(dream4_010_01)))
```

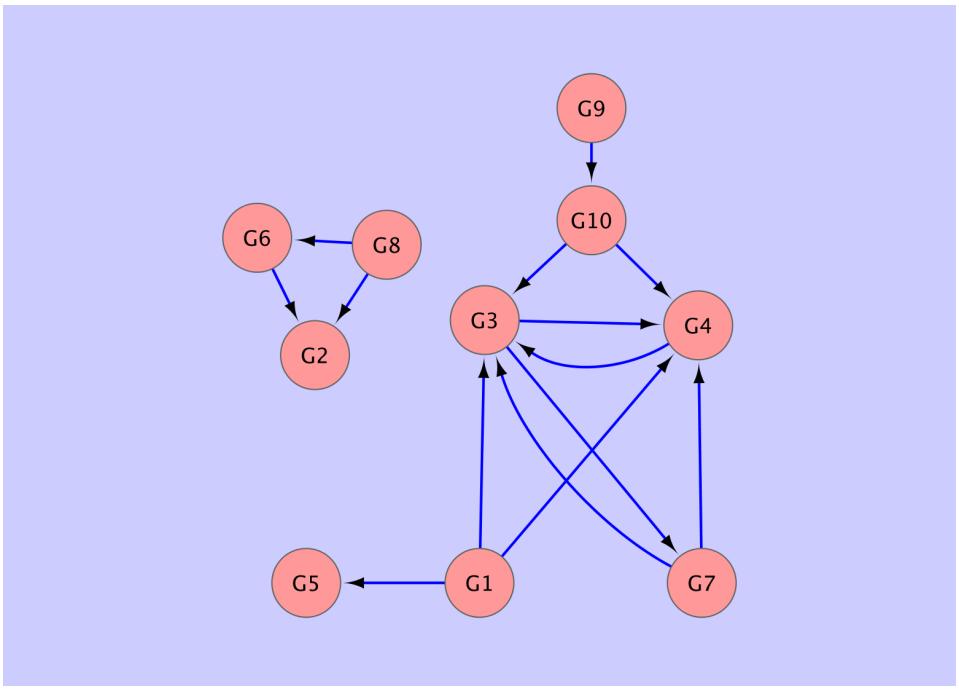
```
[1] "goldStandardAdjacencyMatrix"
[2] "doubleKnockoutGenePairs"
```

```
> mtx.goldStandard <- metadata(dream4_010_01)[[1]]
```

Transform the gold standard matrix into an explict table of regulators and targets.

```
> idx <- which(mtx.goldStandard == 1)
> idx.m1 <- idx -1
> rows <- idx.m1 %% nrow (mtx.goldStandard) + 1
> cols <- idx.m1 %/% nrow (mtx.goldStandard) + 1
> tbl.goldStandard <- data.frame(Regulator=rownames(mtx.goldStandard)[rows],
+                                Target=colnames(mtx.goldStandard)[cols],
+                                Source=rep('goldStandard', length(rows)),
+                                stringsAsFactors=FALSE)
```

Here is an RCytosxcape rendering of this network, followed by the code which produced it.




```
> library(RCytoscape)
> print(dim(mtx.goldStandard))
> print(mtx.goldStandard[1:5, 1:5])
> g.gsam <- graphAM(mtx.goldStandard, edgemode="directed")
> g.gs <- as(g.gsam, "graphNEL")
> g.gs <- initEdgeAttribute(g.gs, "edgeType", "char", "not yet assigned")
> source.nodes <- tbl.goldStandard$Regulator
> target.nodes <- tbl.goldStandard$Target
> edgeData(g.gs, source.nodes, target.nodes, attr="edgeType") <- "regulates"
> cw <- new.CytoscapeWindow("goldStandard 010-01", g.gs, overwriteWindow=TRUE)
> hideAllPanels(cw)
> setWindowSize(cw,800,600)
> showGraphicsDetails(cw,TRUE)
> displayGraph(cw)
> layoutFile <- system.file("extdata", "gs010-01-layout.RData",
+                           package="DREAM4")
> setEdgeTargetArrowRule(cw, "edgeType",
+                        c("not yet assigned", "regulates"),
+                        c("No Arrow", "Arrow"))
> restoreLayout(cw, layoutFile)
> fitContent(cw)
> setZoom(cw, 0.8*getZoom(cw))
```

3.2 Inference Using Expression Data Only

Now we load *networkBMA*, infer the network from twenty-one timepoints of expression over ten genes, assuming no prior knowledge, and sort the predictions.

```
> library(networkBMA)
> tbl.inferred <- networkBMA(mtx.ts1, nTimePoints=nrow(mtx.ts1), self=FALSE)
> tbl.inferred <- tbl.inferred[order(tbl.inferred$PostProb, decreasing=TRUE),]
```

The networkBMA package provides a function with which to assess the inferred network with respect to a reference network. The standard measure of inferential success is *AUPR* – or "area under the precision recall curve". After introducing those quantities and their calculation, we plot the AUPR for this just-completed run.

"Precision" and "Recall" are standard measures of success in the fields of pattern recognition and information retrieval; see Wikipedia, "Precision and recall". In the following, true positives (TP), false positives (FP) and false negative (FN) counts are used.

- *precision*: the fraction of all retrieved instances which are relevant. Formally: TP/(TP+FP)
- *recall*: the fraction of all relevant instances which have been retrieved. Formally: TP/(TP+FN)

Both quantities range from 0.0 to 1.0. In the *AUPR* plot, precision is typically plotted on the Y axis, and recall on the X axis. A perfect retrieval or inference method would always retrieve 100 per cent of all relevant instances, and all retrieved instances would be relevant. In this idealized case, the AUPR would be infinite. Even the best network inference algorithms fare considerably worse than this ideal case. Let us examine the networkBMA inference we just ran, using the `contabs.netwBMA` function provided by the networkBMA package, and the (prc) function. (prc stands for "Precision-Recall Curve".)

```
> tbl.contingency <- contabs.netwBMA(tbl.inferred, tbl.goldStandard[,-3])
```

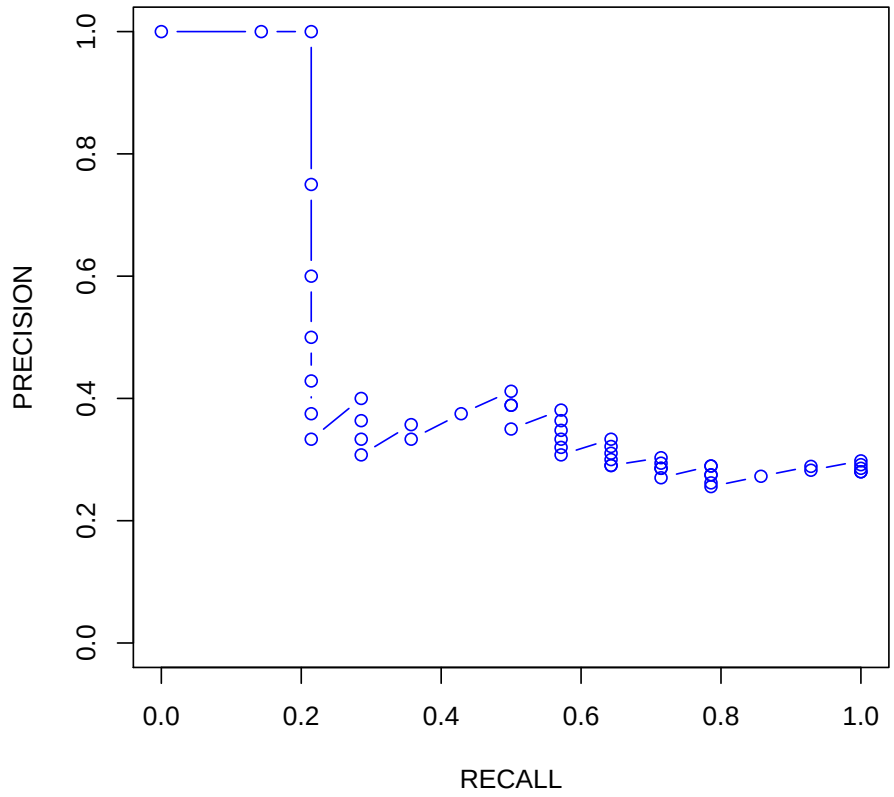
```
[1] "network Before:"
   Regulator TargetGene      prob
37        G1         G5 0.9904764
28        G1         G4 0.9755612
55        G3         G7 0.8032557
```

56	G4	G7	0.7077637
57	G6	G7	0.6971455
46	G10	G6	0.6096357
82	G7	G10	0.5810526
38	G4	G5	0.5781810
10	G1	G2	0.5582822
19	G1	G3	0.5490259
47	G7	G6	0.5379027
83	G3	G10	0.5331821
20	G6	G3	0.5279970
21	G7	G3	0.5127646
84	G6	G10	0.5123684
11	G8	G2	0.4854938
85	G9	G10	0.4854879
23	G9	G3	0.4692819
49	G9	G6	0.4664646
50	G3	G6	0.4637373
51	G8	G6	0.4603542
52	G4	G6	0.4602335
60	G1	G7	0.4568257
13	G10	G2	0.4567721
14	G4	G2	0.4538956
15	G9	G2	0.4493123
24	G4	G3	0.4469614
16	G7	G2	0.4467080
53	G1	G6	0.4431868
87	G8	G10	0.4411886
89	G1	G10	0.4405953
26	G10	G3	0.4303525
27	G8	G3	0.4303215
90	G4	G10	0.4297498
17	G3	G2	0.4259330
61	G10	G7	0.4253910
62	G8	G7	0.4231294
18	G6	G2	0.4226473
39	G8	G5	0.4218161
63	G9	G7	0.3997214
29	G8	G4	0.3218993
40	G10	G5	0.2691155
41	G7	G5	0.2651216
31	G7	G4	0.2200543
32	G3	G4	0.2045634

```
42        G9         G5 0.1975534
33       G10         G4 0.1964631
34        G9         G4 0.1942111
35        G6         G4 0.1940862
43        G6         G5 0.1756136
45        G3         G5 0.1667140
[1] "network After:"
   Regulator TargetGene      prob
37        G1         G5 0.9904764
28        G1         G4 0.9755612
55        G3         G7 0.8032557
56        G4         G7 0.7077637
57        G6         G7 0.6971455
46       G10         G6 0.6096357
82        G7        G10 0.5810526
38        G4         G5 0.5781810
10        G1         G2 0.5582822
19        G1         G3 0.5490259
47        G7         G6 0.5379027
83        G3        G10 0.5331821
20        G6         G3 0.5279970
21        G7         G3 0.5127646
84        G6        G10 0.5123684
11        G8         G2 0.4854938
85        G9        G10 0.4854879
23        G9         G3 0.4692819
49        G9         G6 0.4664646
50        G3         G6 0.4637373
51        G8         G6 0.4603542
52        G4         G6 0.4602335
60        G1         G7 0.4568257
13       G10         G2 0.4567721
14        G4         G2 0.4538956
15        G9         G2 0.4493123
24        G4         G3 0.4469614
16        G7         G2 0.4467080
53        G1         G6 0.4431868
87        G8        G10 0.4411886
89        G1        G10 0.4405953
26       G10         G3 0.4303525
27        G8         G3 0.4303215
90        G4        G10 0.4297498
```

```
17        G3         G2 0.4259330
61       G10         G7 0.4253910
62        G8         G7 0.4231294
18        G6         G2 0.4226473
39        G8         G5 0.4218161
63        G9         G7 0.3997214
29        G8         G4 0.3218993
40       G10         G5 0.2691155
41        G7         G5 0.2651216
31        G7         G4 0.2200543
32        G3         G4 0.2045634
42        G9         G5 0.1975534
33       G10         G4 0.1964631
34        G9         G4 0.1942111
35        G6         G4 0.1940862
43        G6         G5 0.1756136
45        G3         G5 0.1667140
[1] "probs:"
 [1] 0.9904764 0.9755612 0.8032557 0.7077637 0.6971455
 [6] 0.6096357 0.5810526 0.5781810 0.5582822 0.5490259
[11] 0.5379027 0.5331821 0.5279970 0.5127646 0.5123684
[16] 0.4854938 0.4854879 0.4692819 0.4664646 0.4637373
[21] 0.4603542 0.4602335 0.4568257 0.4567721 0.4538956
[26] 0.4493123 0.4469614 0.4467080 0.4431868 0.4411886
[31] 0.4405953 0.4303525 0.4303215 0.4297498 0.4259330
[36] 0.4253910 0.4231294 0.4226473 0.4218161 0.3997214
[41] 0.3218993 0.2691155 0.2651216 0.2200543 0.2045634
[46] 0.1975534 0.1964631 0.1942111 0.1940862 0.1756136
[51] 0.1667140
```

```
> pr <- scores(tbl.contingency, what = c("precision","recall"))
> colors <- c ("blue", "darkred")
> plot(pr$recall, pr$precision, type='b',
+      xlab='RECALL', ylab='PRECISION',
+      col=colors[1],
+      xlim=c(0,1), ylim=c(0,1))
```



The AUPR information:

```
> print(prc(tbl.contingency, plotit=FALSE))

    area   sector    width
0.397478 0.397478 1.000000
```

To make sense of these results, let us look at a few rows selected from the the contingency table.

```
> last.row <- nrow(tbl.contingency)
> mid.row <- as.integer(round(last.row)/2)
> print(tbl.contingency[c(1,mid.row,last.row),])

                 TP FN FP TN
0.166714         14  0 36  6
```

```
0.4469614           9  5 18 24
0.990476427458142   0 14  0 42
```

The first row has perfect recall: all regulatory edges from the gold standard have been inferred, but a very large number of non-existent edges have been inferred as well.

The last row has perfect precision: three edges were accurately inferred, no non-existent edges were proposed, eleven edges were missed.

Further insight into the inferred network is provided by examining the first few entries in the sorted output of `networkBMA`.

```
> print(head(tbl.inferred, n=10))
```

```
   Regulator TargetGene  PostProb
37        G1         G5 0.9904764
28        G1         G4 0.9755612
55        G3         G7 0.8032557
56        G4         G7 0.7077637
57        G6         G7 0.6971455
1         G4         G1 0.6867537
2         G9         G1 0.6446486
46       G10         G6 0.6096357
82        G7        G10 0.5810526
38        G4         G5 0.5781810
```

The three regulatory relationships with a posterior probability score of 1 are accurately inferred. The remaining edges are either reversals of edges in the gold standard network, or incorrect.

3.3 Add Prior Probabilities

It can be seen from these results that the inference of gene regulatory relationships from mRNA expression alone, even for an idealized network with complete information, is imperfect. These strategies are among those used to improve inference:

- Incorporate ancillary data (derived, for instance, from ChiP-seq experiments, or interaction databases)

- Create novel algorithms (for instance, Castelo and Roverato (2009), and Danaher et al. (2011))
- Perform meta-inference, in which several inference tools are used together [Greenfield et al. (2010)]

Bayesian methods, of which the *networkBMA* package is one (see the package vignette for more information and references), exploit prior information (prior probabilities) to improve inference. Recent trends in molecular biology, and the ENCODE project in particular, and public interaction databases, provide a large amount of useful prior regulatory information.

DREAM4 is synthetic data, so neither of these broad classes of prior information is available. We can, however, simulate priors by extracting a few regulatory relationships from the gold standard network, adjusting them with randomly generated probabilities to better approximate real data, and thereby demonstrate how to use networkBMA with priors.

There are two arguments to the `networkBMA` function whe used to with prior information:

- *prior.prob*: an adjacency matrix, where each value is a probability between 0 and 1. This is well-suited for, e.g., experimental data, such as ChIP-seq.
- *known*: a two-column matrix with known (unqualified) regulatory matrices.

We shall use the first of these two parameters, leaving the second to default `NULL`.

Note that in the inference run above, none of the actual targets of regulator *G10* were reported with a significant score:

```
> print(subset(tbl.inferred, Regulator=="G10"))
```

```
   Regulator TargetGene  PostProb
46       G10         G6 0.6096357
13       G10         G2 0.4567721
26       G10         G3 0.4303525
61       G10         G7 0.4253910
72       G10         G8 0.4224411
81       G10         G9 0.4209458
6        G10         G1 0.4006032
40       G10         G5 0.2691155
33       G10         G4 0.1964631
```

But the gold standard shows two targets:

```
> mtx.goldStandard["G10",]
```

```
 G1  G2  G3  G4  G5  G6  G7  G8  G9 G10
  0   0   1   1   0   0   0   0   0   0
```

Since this is synthetic network, we have no experimental data with which to improve the inference. However, we can simulate ChIP-seq data for G10, first by setting low probabilities for all interactions, then adding two probabilities for the G10->G3 and G10->G4 edges:

```
> set.seed(37)
> tbl.priors <- matrix(data=runif(100, 0, 0.4), nrow=10, ncol=10,
+                      dimnames=list(rownames(mtx.goldStandard),
+                                    colnames(mtx.goldStandard)))
> tbl.priors["G10", c("G3", "G4")] <- runif(2, 0.8, 1.0)
```

Run the inference again:

```
> tbl.inferredWithPriors <- networkBMA(mtx.ts1, nTimePoints=nrow(mtx.ts1),
+                                      prior.prob=tbl.priors, self=FALSE)
> tbl.inferredWithPriors <-
+    tbl.inferredWithPriors[order(tbl.inferredWithPriors$PostProb,
+                                 decreasing=TRUE),]
> print(tbl.inferredWithPriors)
```

```
   Regulator TargetGene     PostProb
28        G1         G4 0.9999710531
37        G1         G5 0.9952946000
19       G10         G3 0.9546089000
1         G4         G1 0.7338241000
38        G4         G5 0.5894235000
55        G3         G7 0.4965834000
46       G10         G6 0.4407034000
56        G6         G7 0.4177721000
82        G6        G10 0.4080539000
20        G7         G3 0.3964939000
73        G7         G9 0.3747885000
```

21	G6	G3	0.3679018000
64	G7	G8	0.3579010000
83	G5	G10	0.3389112000
2	G5	G1	0.3388313000
10	G10	G2	0.3362281000
84	G8	G10	0.3201253000
47	G9	G6	0.3044811000
57	G2	G7	0.2998596000
22	G2	G3	0.2778042000
23	G9	G3	0.2757589000
11	G7	G2	0.2752194000
24	G8	G3	0.2647638000
12	G5	G2	0.2578058000
48	G3	G6	0.2548358000
58	G4	G7	0.2510126000
13	G6	G2	0.2409387000
74	G2	G9	0.2067339000
59	G5	G7	0.2067174000
85	G2	G10	0.2031888000
86	G1	G10	0.1960115000
25	G5	G3	0.1824667000
60	G8	G7	0.1769256000
75	G1	G9	0.1733800000
14	G9	G2	0.1695565000
49	G5	G6	0.1657558000
61	G1	G7	0.1628025000
15	G1	G2	0.1607274000
65	G9	G8	0.1507207000
66	G5	G8	0.1499059000
67	G1	G8	0.1480656000
50	G2	G6	0.1479644000
68	G4	G8	0.1474138000
87	G3	G10	0.1362227000
51	G8	G6	0.1209756000
26	G1	G3	0.1150399000
69	G2	G8	0.1054485000
27	G4	G3	0.1044979000
39	G10	G5	0.1027690000
52	G1	G6	0.0966448000
3	G6	G1	0.0935315000
88	G7	G10	0.0875563000
29	G5	G4	0.0861099000

```
16        G3         G2 0.0859564000
40        G2         G5 0.0825089000
53        G4         G6 0.0822431000
41        G8         G5 0.0794257000
70        G6         G8 0.0782316000
42        G3         G5 0.0772084000
71       G10         G8 0.0742550000
4         G9         G1 0.0720636000
76        G6         G9 0.0654681000
62        G9         G7 0.0614382000
5         G3         G1 0.0600934000
6         G8         G1 0.0574313000
77        G4         G9 0.0573925000
89        G9        G10 0.0521405000
7         G7         G1 0.0487031000
72        G3         G8 0.0473790000
78        G8         G9 0.0341217000
43        G9         G5 0.0336247000
90        G4        G10 0.0271082000
79        G3         G9 0.0239630000
80       G10         G9 0.0238524000
17        G8         G2 0.0225216000
81        G5         G9 0.0205909000
30       G10         G4 0.0199499000
8        G10         G1 0.0177882000
31        G7         G4 0.0171970000
54        G7         G6 0.0088348915
63       G10         G7 0.0072199000
32        G8         G4 0.0068228941
33        G2         G4 0.0064695132
44        G6         G5 0.0064087000
9         G2         G1 0.0048108000
34        G9         G4 0.0044105779
35        G6         G4 0.0032741339
45        G7         G5 0.0009414626
18        G4         G2 0.0006409585
36        G3         G4 0.0003465549
```

G10->G3 and G10->G4 now show non-negligible posterior probabilities

Here we plot the precision-recall curves for both runs together, for easy comparison:

```
> tbl.contingencyWithPriors <- contabs.netwBMA(tbl.inferredWithPriors, tbl.goldStanda

[1] "network Before:"
   Regulator TargetGene         prob
28        G1         G4 0.9999710531
37        G1         G5 0.9952946000
19       G10         G3 0.9546089000
38        G4         G5 0.5894235000
55        G3         G7 0.4965834000
46       G10         G6 0.4407034000
56        G6         G7 0.4177721000
82        G6        G10 0.4080539000
20        G7         G3 0.3964939000
21        G6         G3 0.3679018000
10       G10         G2 0.3362281000
84        G8        G10 0.3201253000
47        G9         G6 0.3044811000
23        G9         G3 0.2757589000
11        G7         G2 0.2752194000
24        G8         G3 0.2647638000
48        G3         G6 0.2548358000
58        G4         G7 0.2510126000
13        G6         G2 0.2409387000
86        G1        G10 0.1960115000
60        G8         G7 0.1769256000
14        G9         G2 0.1695565000
61        G1         G7 0.1628025000
15        G1         G2 0.1607274000
87        G3        G10 0.1362227000
51        G8         G6 0.1209756000
26        G1         G3 0.1150399000
27        G4         G3 0.1044979000
39       G10         G5 0.1027690000
52        G1         G6 0.0966448000
88        G7        G10 0.0875563000
16        G3         G2 0.0859564000
53        G4         G6 0.0822431000
41        G8         G5 0.0794257000
42        G3         G5 0.0772084000
62        G9         G7 0.0614382000
89        G9        G10 0.0521405000
43        G9         G5 0.0336247000
```

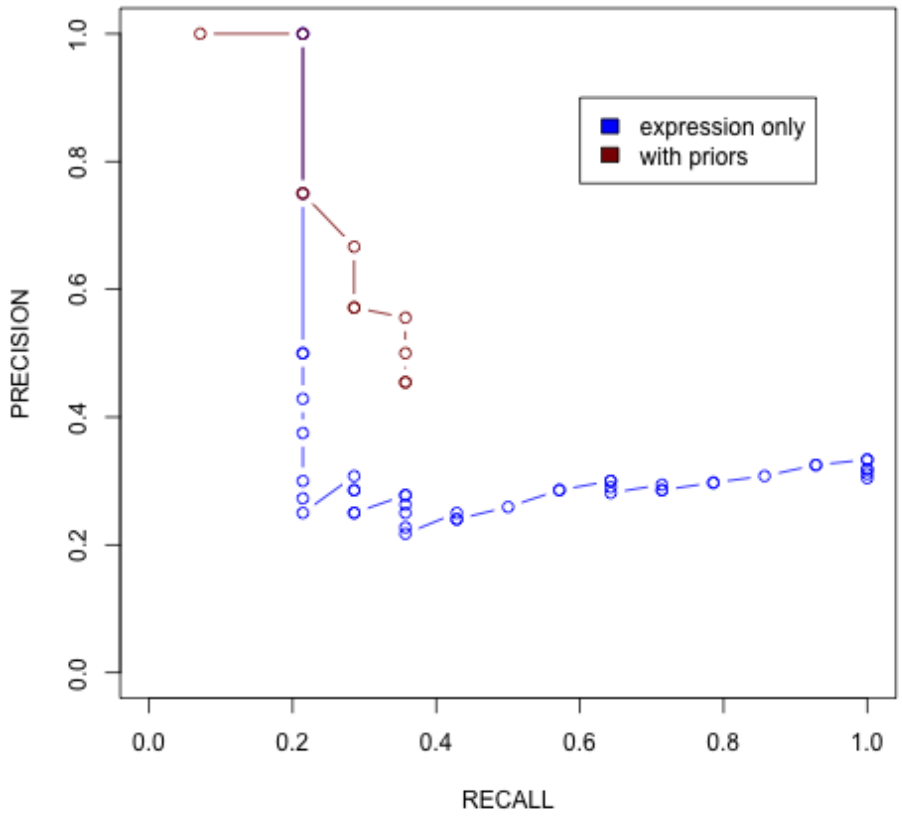
```
90        G4        G10 0.0271082000
17        G8         G2 0.0225216000
30       G10         G4 0.0199499000
31        G7         G4 0.0171970000
54        G7         G6 0.0088348915
63       G10         G7 0.0072199000
32        G8         G4 0.0068228941
44        G6         G5 0.0064087000
34        G9         G4 0.0044105779
35        G6         G4 0.0032741339
45        G7         G5 0.0009414626
18        G4         G2 0.0006409585
36        G3         G4 0.0003465549
[1] "network After:"
   Regulator TargetGene         prob
28        G1         G4 0.9999710531
37        G1         G5 0.9952946000
19       G10         G3 0.9546089000
38        G4         G5 0.5894235000
55        G3         G7 0.4965834000
46       G10         G6 0.4407034000
56        G6         G7 0.4177721000
82        G6        G10 0.4080539000
20        G7         G3 0.3964939000
21        G6         G3 0.3679018000
10       G10         G2 0.3362281000
84        G8        G10 0.3201253000
47        G9         G6 0.3044811000
23        G9         G3 0.2757589000
11        G7         G2 0.2752194000
24        G8         G3 0.2647638000
48        G3         G6 0.2548358000
58        G4         G7 0.2510126000
13        G6         G2 0.2409387000
86        G1        G10 0.1960115000
60        G8         G7 0.1769256000
14        G9         G2 0.1695565000
61        G1         G7 0.1628025000
15        G1         G2 0.1607274000
87        G3        G10 0.1362227000
51        G8         G6 0.1209756000
26        G1         G3 0.1150399000
```

```
27        G4         G3 0.1044979000
39       G10         G5 0.1027690000
52        G1         G6 0.0966448000
88        G7        G10 0.0875563000
16        G3         G2 0.0859564000
53        G4         G6 0.0822431000
41        G8         G5 0.0794257000
42        G3         G5 0.0772084000
62        G9         G7 0.0614382000
89        G9        G10 0.0521405000
43        G9         G5 0.0336247000
90        G4        G10 0.0271082000
17        G8         G2 0.0225216000
30       G10         G4 0.0199499000
31        G7         G4 0.0171970000
54        G7         G6 0.0088348915
63       G10         G7 0.0072199000
32        G8         G4 0.0068228941
44        G6         G5 0.0064087000
34        G9         G4 0.0044105779
35        G6         G4 0.0032741339
45        G7         G5 0.0009414626
18        G4         G2 0.0006409585
36        G3         G4 0.0003465549
[1] "probs:"
 [1] 0.9999710531 0.9952946000 0.9546089000 0.5894235000
 [5] 0.4965834000 0.4407034000 0.4177721000 0.4080539000
 [9] 0.3964939000 0.3679018000 0.3362281000 0.3201253000
[13] 0.3044811000 0.2757589000 0.2752194000 0.2647638000
[17] 0.2548358000 0.2510126000 0.2409387000 0.1960115000
[21] 0.1769256000 0.1695565000 0.1628025000 0.1607274000
[25] 0.1362227000 0.1209756000 0.1150399000 0.1044979000
[29] 0.1027690000 0.0966448000 0.0875563000 0.0859564000
[33] 0.0822431000 0.0794257000 0.0772084000 0.0614382000
[37] 0.0521405000 0.0336247000 0.0271082000 0.0225216000
[41] 0.0199499000 0.0171970000 0.0088348915 0.0072199000
[45] 0.0068228941 0.0064087000 0.0044105779 0.0032741339
[49] 0.0009414626 0.0006409585 0.0003465549
```

```
> plot(pr$recall, pr$precision, type='b',
+      xlab='RECALL', ylab='PRECISION',
+      col=colors[1],
```

```
+      xlim=c(0,1), ylim=c(0,1))
> prWithPriors <- scores( tbl.contingencyWithPriors, what = c("precision","recall"))
> lines(prWithPriors$recall, prWithPriors$precision, type='b',
+       xlab='RECALL', ylab='PRECISION',
+       col=colors[2],
+       xlim=c(0,1), ylim=c(0,1))
> legend.titles = c("expression only", "with priors")
> legend (0.6, 0.9, legend.titles, colors)
> print(prc(tbl.contingencyWithPriors, plotit=FALSE))

     area    sector     width
0.3441728 0.3263156 0.8571429
```



And examine the contingency table of the second run:

```
> print(tbl.contingencyWithPriors)
```

	TP	FN	FP	TN
0.000346554862016032	13	1	37	5
0.00064095850563976	13	1	36	6
0.000941462613987145	13	1	36	6
0.00327413390919384	13	1	34	8
0.00441057789712965	13	1	33	9
0.0064087	13	1	33	9
0.006822894058	13	1	32	10
0.0072199	13	1	31	11
0.00883489148941119	13	1	30	12
0.017197	13	1	29	13
0.0199499	12	2	29	13
0.0225216	11	3	29	13
0.0271082	10	4	29	13
0.0336247	10	4	28	14
0.0521405	10	4	27	15
0.0614382	9	5	27	15
0.0772084	9	5	26	16
0.0794257	9	5	25	17
0.0822431	9	5	23	19
0.0859564	9	5	22	20
0.0875563	9	5	22	20
0.0966448	9	5	20	22
0.102769	9	5	20	22
0.1044979	9	5	19	23
0.1150399	8	6	19	23
0.1209756	7	7	19	23
0.1362227	6	8	19	23
0.1607274	6	8	18	24
0.1628025	6	8	17	25
0.1695565	6	8	16	26
0.1769256	6	8	15	27
0.1960115	6	8	14	28
0.2409387	5	9	13	29
0.2510126	5	9	13	29
0.2548358	5	9	12	30
0.2647638	5	9	11	31
0.2752194	5	9	10	32
0.2757589	5	9	9	33
0.3044811	5	9	8	34
0.3201253	5	9	7	35
0.3362281	5	9	6	36

```
0.3679018              5  9  5 37
0.3964939              4 10  4 38
0.4080539              4 10  4 38
0.4177721              4 10  3 39
0.4407034              4 10  2 40
0.4965834              4 10  1 41
0.5894235              3 11  1 41
0.9546089              3 11  0 42
0.9952946              2 12  0 42
0.999971053129436      1 13  0 42
```

4 Conclusion

The DREAM4 synthetic mRNA expression data, and the gold standard networks from which they are generated, offer a useful test for gene regulatory network inference. One result of such tests is the realization of how difficult such inference is. Incorporating prior information into inference is beneficial. We anticipate that in the near future many other kinds of regulatory data will begin to match mRNA expression for completeness, accuracy and cost, leading to increasingly constrained, and therefore increasingly accurate network inference.

5 Bibliography

R. Castelo and A. Roverato. Reverse engineering molecular regulatory networks from microarray data with qp-graphs. *Journal of Computational Biology*, 16(2):213–227, 2009.

P. Danaher, P. Wang, and D.M. Witten. The joint graphical lasso for inverse covariance estimation across multiple classes. *arXiv preprint arXiv:1111.0324*, 2011.

C. Fraley, K.Y. Yeung, and A. Raftery. networkBMA: Regression-based network inference using Bayesian Model Averaging. `http://www.bioconductor.org/packages/release/bioc/html/networkBMA.html`, 2012. R package version 1.1.0.

A. Greenfield, A. Madar, H. Ostrer, and R. Bonneau. Dream4: Combining genetic and dynamic information to identify biological networks and dynamical models. *PloS one*, 5(10):e13397, 2010.

K. Lo, A.E. Raftery, K.M. Dombek, J. Zhu, E.E. Schadt, R.E. Bumgarner, and K.Y. Yeung. Integrating external biological knowledge in the construction of regulatory networks from time-series expression data. *BMC Systems Biology*, 6(1):101, 2012.

D. Marbach, T. Schaffter, C. Mattiussi, and D. Floreano. Generating realistic in silico gene networks for performance assessment of reverse engineering methods. *Journal of Computational Biology*, 16(2):229–239, 2009.

D. Marbach, R.J. Prill, T. Schaffter, C. Mattiussi, D. Floreano, and G. Stolovitzky. Revealing strengths and weaknesses of methods for gene network inference. *Proceedings of the National Academy of Sciences*, 107(14):6286–6291, 2010.

R.J. Prill, D. Marbach, J. Saez-Rodriguez, P.K. Sorger, L.G. Alexopoulos, X. Xue, N.D. Clarke, G. Altan-Bonnet, and G. Stolovitzky. Towards a rigorous assessment of systems biology models: the dream3 challenges. *PloS one*, 5(2):e9202, 2010.

R.J. Prill, J. Saez-Rodriguez, L.G. Alexopoulos, P.K. Sorger, and G. Stolovitzky. Crowdsourcing network inference: the dream predictive signaling network challenge. *Science Signalling*, 4(189):mr7, 2011.

K.Y. Yeung, K.M. Dombek, K. Lo, J.E. Mittler, J. Zhu, E.E. Schadt, R.E. Bumgarner, and A.E. Raftery. Construction of regulatory networks using expression time-series data of a genotyped population. *Proceedings of the National Academy of Sciences*, 108(48):19436–19441, 2011.