Reverse engineering transcriptional regulatory networks from gene expression microarray data using qpgraph

Robert Castelo and Alberto Roverato

April 21, 2009

1 Introduction

This vignette describes how to use the package qpgraph in order to reverse engineer a transcriptional regulatory network from a particular gene expression microarray data set of Escherichia coli (E. coli). Concretely, the data corresponds to $n = 43$ experiments of various mutants under oxygen deprivation (Covert et al., 2004). The mutants were designed to monitor the response from E. coli during an oxygen shift in order to target the *a priori* most relevant part of the transcriptional netwok by using six strains with knockouts of the following key transcriptional regulators in the oxygen response: $\Delta arcA$, $\Delta appY$, Δfnr, $\Delta oxyR$, $\Delta soxS$ and the double knockout $\Delta arcA\Delta fnr$. To get started, load the following packages:

```
> library(annotate)
> library(genefilter)
> library(org.EcK12.eg.db)
> library(graph)
> library(qpgraph)
```

Within the qpgraph package there is a data file called EcoliOxygen in which we will find the following objects stored:

```
> data(EcoliOxygen)
> ls()
```

```
[1] "filtered.regulon6.1" "gds680.eset"
```

where filtered.regulon6.1 contains a subset of the E. coli transcriptional network from RegulonDB 6.1 (Gama-Castro et al., 2008) obtained through the filtering steps described in (Castelo and Roverato, 2008) and gds680.eset is an ExpressionSet object with the $n = 43$ microarray experiments of Covert et al. (2004) described before. These experiments provide expression profiles for $p = 4\,205$ genes derived from the original data set downloaded from the Gene Expression Omnibus (Barrett et al., 2007) with accession GDS680 by applying the filtering steps described also in (Castelo and Roverato, 2008). You can see a summary of the data contained in this object by simply typing its name on the R-shell:

```
> gds680.eset
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 4205 features, 43 samples
  element names: exprs
phenoData
  sampleNames: GSM18235, GSM18236, ..., GSM18289  (43 total)
  varLabels and varMetadata description:
    sample: arbitrary numbering
featureData
  featureNames: 947315, 945490, ..., 946370  (4205 total)
  fvarLabels and fvarMetadata description: none
experimentData: use 'experimentData(object)'
  pubMedIds: 15129285
Annotation: org.EcK12.eg.db
```

where the usual probeset identifiers in the `featureNames` slot have been already replaced by the corresponding Entrez IDs according to the filtering steps taken in (Castelo and Roverato, 2008).

2 Preprocessing steps

In order to keep time and space requirements of the calculations at a manageable level for a vignette, we will use a subset of these data. Concretely, we will consider first those genes forming part in RegulonDB of the regulatory modules of the five knocked-out transcription factors and select the 100 genes with largest variability measured by the interquartile range (IQR). In the `qpgraph` package the filtered RegulonDB data is stored in the form of a data frame where each row corresponds to a transcriptional regulatory relationship, the first two columns contain Blattner IDs of the transcription factor (TF) and target (TG) genes, respectively, and the following two correspond to the same genes but specified by Entrez IDs. The fifth column contains the direction of the regulation according to RegulonDB and this is how the first rows look like:

```
> head(filtered.regulon6.1)

  BlID_TF BlID_TG EgID_TF EgID_TG Direction
1   b0464   b0463  945516  945112         -
2   b0464   b0462  945516  945108         -
5   b2213   b4187  946710  948710         +
6   b2213   b2068  946710  947371         +
7   b2213   b2212  946710  946708        +-
8   b4116   b4117  948627  948638         +
```

We select the rows of `filtered.regulon6.1` that correspond to the subnetwork of the 5 knocked-out TFs as follows. First, obtain the Entrez IDs of these genes from their symbols:

```
> knockoutsyms <- c("arcA", "appY", "oxyR", "soxS", "fnr")
> rmap <- revmap(getAnnMap("SYMBOL", "org.EcK12.eg.db"))
> knockoutEgIDs <- unlist(mget(knockoutsyms, rmap))
> knockoutEgIDs
```

```
    arcA     appY     oxyR     soxS      fnr
"948874" "948797" "948462" "948567" "945908"
```

Next, get all transcriptional regulatory relationships from these TFs and obtain the subset of non-redundant genes involved in this subnetwork:

```
> mt <- match(filtered.regulon6.1[, "EgID_TF"], knockoutEgIDs)
> cat("These 5 TFs are involved in", sum(!is.na(mt)), "TF-TG interactions\n")
```

```
These 5 TFs are involved in 462 TF-TG interactions
```

```
> genesO2net <- as.character(unique(as.vector(as.matrix(filtered.regulon6.1[!is.na(mt),
+     c("EgID_TF", "EgID_TG")]))))
> cat("There are", length(genesO2net), "different genes in this subnetwork\n")
```

```
There are 378 different genes in this subnetwork
```

and, finally, select the 100 most variable genes by using the IQR:

```
> IQRs <- apply(exprs(gds680.eset[genesO2net, ]), 1, IQR)
> largestIQRgenesO2net <- names(sort(IQRs, decreasing = TRUE)[1:100])
```

Using these genes we create a new `ExpressionSet` object, which we shall call `subset.gds680.eset` by subsetting directly from `gds680.eset`:

```
> dim(gds680.eset)
```

```
Features  Samples
    4205       43
```

```
> subset.gds680.eset <- gds680.eset[largestIQRgenesO2net, ]
> dim(subset.gds680.eset)
```

```
Features  Samples
     100       43
```

```
> subset.gds680.eset
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 100 features, 43 samples
  element names: exprs
phenoData
  sampleNames: GSM18235, GSM18236, ..., GSM18289  (43 total)
  varLabels and varMetadata description:
    sample: arbitrary numbering
featureData
  featureNames: 948403, 945316, ..., 948063  (100 total)
  fvarLabels and fvarMetadata description: none
experimentData: use 'experimentData(object)'
  pubMedIds: 15129285
Annotation: org.EcK12.eg.db
```

In order to compare later our results against the transcriptional network from RegulonDB we will extract the subnetwork that involves exclusively these selected 100 genes as follows. First extract the corresponding rows:

```
> mtTF <- match(filtered.regulon6.1[, "EgID_TF"], largestIQRgenesO2net)
> mtTG <- match(filtered.regulon6.1[, "EgID_TG"], largestIQRgenesO2net)
> cat(sprintf("The 100 genes are involved in %d RegulonDB interactions\n",
+     sum(!is.na(mtTF) & !is.na(mtTG))))

The 100 genes are involved in 128 RegulonDB interactions

> subset.filtered.regulon6.1 <- filtered.regulon6.1[!is.na(mtTF) &
+     !is.na(mtTG), ]
```

Next, we need to build an incidence matrix of this subset of interactions, which we shall call `subset.filtered.regulon6.1.I`, in order to ease posterior comparisons with reverse-engineered networks and for this purpose we should first map the Entrez IDs to the indexed position they have within the `ExpressionSet` object and then build the incidence matrix:

```
> TFi <- match(subset.filtered.regulon6.1[, "EgID_TF"], featureNames(subset.gds680.eset))
> TGi <- match(subset.filtered.regulon6.1[, "EgID_TG"], featureNames(subset.gds680.eset))
> subset.filtered.regulon6.1 <- cbind(subset.filtered.regulon6.1,
+     idx_TF = TFi, idx_TG = TGi)
> p <- dim(subset.gds680.eset)["Features"]
> subset.filtered.regulon6.1.I <- matrix(FALSE, nrow = p, ncol = p)
> rownames(subset.filtered.regulon6.1.I) <- featureNames(subset.gds680.eset)
> colnames(subset.filtered.regulon6.1.I) <- featureNames(subset.gds680.eset)
> idxTFTG <- as.matrix(subset.filtered.regulon6.1[, c("idx_TF",
+     "idx_TG")])
> subset.filtered.regulon6.1.I[idxTFTG] <- subset.filtered.regulon6.1.I[cbind(idxTFTG[,
+     2], idxTFTG[, 1])] <- TRUE
```

3 Reverse engineer a transcriptional regulatory network

We are set to reverse engineer a transcriptional regulatory network from the subset of the oxygen deprivation microarray data formed by the selected 100 genes and we will use three methods: 1. the estimation of Pearson correlation coefficients (PCCs); 2. the estimation of average non-rejection rates (avgNRRs); and, as a baseline comparison, 3. the assignment of random correlations drawn from a uniform distribution between -1 and +1 to every pair of genes. We can estimate PCCs for all gene pairs with the function `qpPCC` from the `qpgraph` package as follows:

```
> pcc.estimates <- qpPCC(subset.gds680.eset)
```

which returns a list with two members, one called `R` with the PCCs and another called `P` with the corresponding two-sided P-values for the null hypothesis of zero correlation. Let's take a look to the distribution of absolute PCCs between all possible TF-TG pairs in this subset of 100 genes:

```
> largestIQRgenesO2net_i <- match(largestIQRgenesO2net, featureNames(subset.gds680.eset))
> largestIQRgenesO2netTFs <- largestIQRgenesO2net[!is.na(match(largestIQRgenesO2net,
+     filtered.regulon6.1[, "EgID_TF"]))]
> largestIQRgenesO2netTFs_i <- match(largestIQRgenesO2netTFs, featureNames(subset.gds680.eset))
> TFsbyTGs <- as.matrix(expand.grid(largestIQRgenesO2netTFs_i,
+     setdiff(largestIQRgenesO2net_i, largestIQRgenesO2netTFs_i)))
> TFsbyTGs <- rbind(TFsbyTGs, t(combn(largestIQRgenesO2netTFs_i,
+     2)))
> summary(abs(pcc.estimates$R[TFsbyTGs]))

     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0001023 0.3027000 0.5146000 0.5037000 0.7182000 0.9862000
```

Note that they are distributed almost uniformly at random throughout the entire range [0,1] while if we look at the distribution of the PCC estimates for the entire RegulonDB data, i.e., for all possible TF-TG pairs among the initial $p = 4\,205$ genes:

```
> regulonDBgenes <- as.character(unique(c(filtered.regulon6.1[,
+     "EgID_TF"], filtered.regulon6.1[, "EgID_TG"])))
> cat(sprintf("The RegulonDB transcriptional network involves %d genes",
+     length(regulonDBgenes)))

The RegulonDB transcriptional network involves 1428 genes

> pcc.allRegulonDB.estimates <- qpPCC(gds680.eset[regulonDBgenes,
+     ])
> allTFs_i <- match(unique(filtered.regulon6.1[, "EgID_TF"]), regulonDBgenes)
> allTFsbyTGs <- as.matrix(expand.grid(allTFs_i, setdiff(1:length(regulonDBgenes),
+     allTFs_i)))
> allTFsbyTGs <- rbind(allTFsbyTGs, t(combn(allTFs_i, 2)))
> summary(abs(pcc.allRegulonDB.estimates$R[allTFsbyTGs]))

     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
2.393e-06 1.038e-01 2.204e-01 2.555e-01 3.739e-01 9.862e-01
```

we see that, opposite to what happens in the subset of 100 genes, most of the absolute PCC values for all (i.e., present and absent from RegulonDB) TF-TG pairs are small. The high level of correlation among most of the 100 genes is probably due to the coordinated transcriptional program to which all these genes belong to, since they form part of some of the key regulatory modules in the response to oxygen deprivation. Recall that five TFs in these regulatory modules were knocked-out in the assayed experimental conditions and we selected the most variable 100 genes. Concretely, among the five TFs the following ones were finally included in these 100 most variable genes:

```
> mt <- match(knockoutEgIDs, largestIQRgenesO2net)
> unlist(mget(largestIQRgenesO2net[mt[!is.na(mt)]], org.EcK12.egSYMBOL))

948874 948797 948567 945908
"arcA" "appY" "soxS"  "fnr"
```

If we look now to the distribution of absolute PCC values for only those TF-TG pairs that are present in the subset of RegulonDB involved in the 100 genes:

```
> maskRegulonTFTG <- subset.filtered.regulon6.1.I & upper.tri(subset.filtered.regulon6.1.I)
> summary(abs(pcc.estimates$R[maskRegulonTFTG]))

     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
0.0001023 0.1561000 0.2896000 0.3304000 0.4574000 0.9862000
```

they show much lower values (50% < 0.3) and thus we can expect that a substantial number of TF-TG pairs absent from RegulonDB but with strong PCC values will sneak in as false positives in our assessment below of the estimation of PCCs as a reverse engineering method. If we look at the distribution of the PCC values from the RegulonDB interactions separetely by each of the regulatory modules within these 100 genes (i.e., by each of the TFs) we can see that *fnr* is one of the responsibles for having low PCCs in a large fraction of this subset of RegulonDB. We have used the R code below to produce Figure 1 where this is shown.

```
> par(mar = c(5, 4, 5, 2))
> pccsbyTF <- list()
> for (TFi in subset.filtered.regulon6.1[, "idx_TF"]) pccsbyTF[[featureNames(subset.gds680.eset
+     subset.filtered.regulon6.1.I[TFi, ]])
> bp <- boxplot(pccsbyTF, names = sprintf("%s", mget(names(pccsbyTF),
+     org.EcK12.egSYMBOL)), ylab = "Pearson correlation coefficient (PCC)",
+     main = paste("Distribution of PCCs in each RegulonDB", "regulatory module within the 100
+         sep = "\n"))
> nint <- sprintf("(%d)", sapply(names(pccsbyTF), function(x) sum(subset.filtered.regulon6.1.I[
+     ])))
> mtext(nint, at = seq(bp$n), line = +2, side = 1)
> mtext("Transcription factor (# RegulonDB interactions)", side = 1,
+     line = +4)
```

As observed by Covert et al. (2004) when *fnr* becomes active under anaerobic conditions its mRNA level is significantly reduced and we hypothesize that this fact probably leads to weak correlations of the expression level with its target genes.

Now we will show how can we use qp-graphs to tackle such a challenging situation. We should start by estimating avgNRRs with the function `qpAvgNrr` but before we do that, and for the sake of reproducibility of the results of this vignette, we should take into account that because the non-rejection rate is estimated by a random sampling procedure (see Castelo and Roverato, 2006), its value may vary slightly from run to run and thus edges with very similar avgNRR values may alternate their positions when ranking them and thus show up differently in different qp-graphs obtained from different runs if, within the ranking, they lie at the boundary of the precision threshold we may be using later. For this reason, and in order to let the reader reproduce exactly the results contained in this vignette, we will specify a particular seed to the random number generator as follows:

```
> set.seed(12345)
```

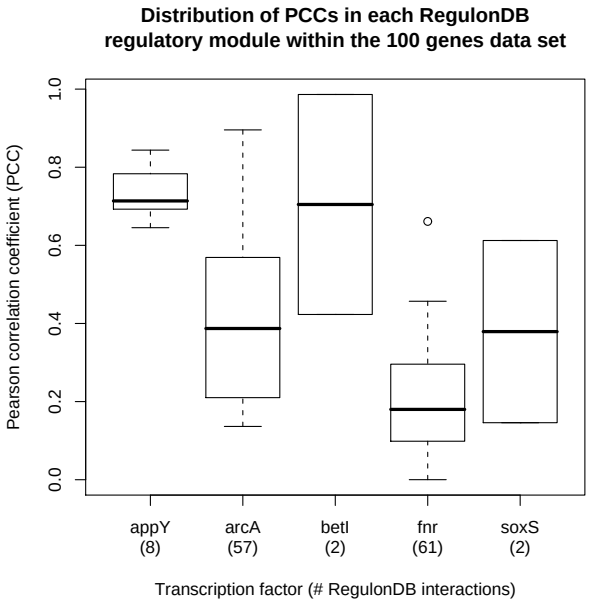


Figure 1: Distribution of Pearson correlation coefficients (PCCs) calculated from the Covert et al. (2004) oxygen deprivation data between genes forming RegulonDB interactions. Distributed values are shown separately by each regulatory module defined as a transcription factor (TF) and its set of target genes.

Moreover, in this exercise, we are only interested in TF-TG relationships and thus we will speed-up the calculations by restricting the formation of gene pairs with the parameters `pairup.i` and `pairup.j` in the following way:

```
> avgnrr.estimates <- qpAvgNrr(subset.gds680.eset, pairup.i = largestIQRgenesO2netTFs,
+     pairup.j = largestIQRgenesO2net)

q=1
0.........10.........20.........30.........40.........50.........60.........70.........80.....
q=11
0.........10.........20.........30.........40.........50.........60.........70.........80.....
q=21
0.........10.........20.........30.........40.........50.........60.........70.........80.....
q=31
0.........10.........20.........30.........40.........50.........60.........70.........80.....
```

The function `qpAvgNrr` uses by default four equidistant q-values along the available range and returns a matrix with the estimates for all gene pairs except when, as in this case, we restrict the genes allowed to pair with each other. In order to assess the accuracy of the PCC and qp-graph methods we will use the transcriptional regulatory relationships in the subset of RegulonDB that we selected before and calculate precision-recall curves (Fawcett, 2006) using the `qpPrecisionRecall` function from the `qpgraph` package.

We have to be careful with the fact that while we calculated avgNRRs only for TF-TG pairs, the matrix `pcc.estimates$R` contains PCC values for all pairs of genes and thus in order to obtain comparable precision-recall curves we will have to inform `qpPrecisionRecall` of the pairs that should be considered when giving it the matrix of PCC values. This is not necessary with avgNRRs as the matrix has `NA` values on the cells corresponding to pairs where no calculation was performed (on the pairs of non-transcription factor genes).

```
> pcc.prerec <- qpPrecisionRecall(abs(pcc.estimates$R), subset.filtered.regulon6.1.I,
+     decreasing = TRUE, pairup.i = largestIQRgenesO2netTFs, pairup.j = largestIQRgenesO2net,
+     recallSteps = c(seq(0, 0.1, 0.01), seq(0.2, 1, 0.1)))
```

Note also that, opposite to PCCs, in avgNRR estimates the value indicating the smallest strength of the interaction is 1 instead of 0 and therefore we should set `decreasing=FALSE`:

```
> avgnrr.prerec <- qpPrecisionRecall(avgnrr.estimates, subset.filtered.regulon6.1.I,
+     decreasing = FALSE, recallSteps = c(seq(0, 0.1, 0.01), seq(0.2,
+         1, 0.1)))
```

Finally, in order to have the assignment of random correlations as a baseline comparison we should do the following:

```
> set.seed(12345)
> rndcor <- matrix(runif(10000, min = -1, max = 1), nrow = 100,
+     ncol = 100)
> rownames(rndcor) <- colnames(rndcor) <- rownames(avgnrr.estimates)
> random.prerec <- qpPrecisionRecall(abs(rndcor), subset.filtered.regulon6.1.I,
+     decreasing = TRUE, pairup.i = largestIQRgenesO2netTFs, pairup.j = largestIQRgenesO2net,
+     recallSteps = c(seq(0, 0.1, 0.01), seq(0.2, 1, 0.1)))
```

where again we have specified a seed for the random number generator in order to enforce reproducing the same random correlations each time we run this vignette.

A way to quantitatively compare these three precision-recall curves is to calculate the area under these curves where the larger it is, the more accurate the method is:

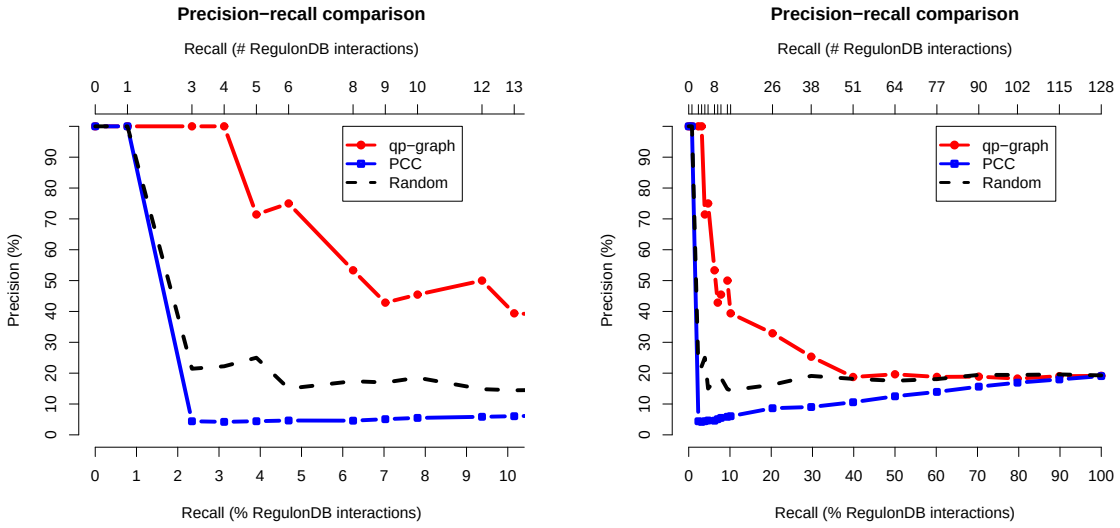
```
> f <- approxfun(pcc.prerec[, c("Recall", "Precision")])
> area <- integrate(f, 0, 1)$value
> f <- approxfun(avgnrr.prerec[, c("Recall", "Precision")])
> area <- cbind(area, integrate(f, 0, 1)$value)
> f <- approxfun(random.prerec[, c("Recall", "Precision")])
> area <- cbind(area, integrate(f, 0, 1)$value)
> colnames(area) <- c("PCC", "avgNRR", "Random")
> rownames(area) <- "AreaPrecisionRecall"
> printCoefmat(area)

                         PCC  avgNRR Random
AreaPrecisionRecall 0.13747 0.27206 0.1955
```

From these values we may conclude that, for these data ($n = 43$ microarray experiments on $p = 100$ genes among which 7 are TFs, and with 128 transcriptional regulatory relationships from RegulonDB for comparison), the random method outperforms the usage of PCCs but it performs worse than the qp-graph method with avgNRRs which, therefore, constitutes the best solution among these three approaches. While it may sound a bit counter-intuitive that the assignment of a random correlation provides better results than using PCCs, the reason for this lies in the fact that with these data we have $7 \times 93 + \binom{7}{2} = 672$ possible TF-TG interactions out of which 128 from RegulonDB form our gold-standard. This yields a bottomline precision of $(128/672) \times 100 \approx 19\%$ which is quickly attained by drawing random correlations. However, we saw before that absolute PCCs of the RegulonDB interactions forming our gold-standard are most of them distributed under 0.5 and this yields, for this particular data set, a performance that is worse than random at regions of high-precision. We may see this situation depicted in Figure 2 whose left panel has been produced with the following R code:

```
> par(mai = c(0.5, 0.5, 1, 0.5), mar = c(5, 4, 7, 2) + 0.1)
> plot(avgnrr.prerec[, c(1, 2)], type = "b", lty = 1, pch = 19,
+     cex = 0.65, lwd = 4, col = "red", xlim = c(0, 0.1), ylim = c(0,
+         1), axes = FALSE, xlab = "Recall (% RegulonDB interactions)",
+     ylab = "Precision (%)")
> axis(1, at = seq(0, 1, 0.01), labels = seq(0, 100, 1))
> axis(2, at = seq(0, 1, 0.1), labels = seq(0, 100, 10))
> axis(3, at = avgnrr.prerec[, "Recall"], labels = round(avgnrr.prerec[,
+     "Recall"] * dim(subset.filtered.regulon6.1)[1], digits = 0))
> title(main = "Precision-recall comparison", line = +5)
> lines(pcc.prerec[, c(1, 2)], type = "b", lty = 1, pch = 22, cex = 0.65,
+     lwd = 4, col = "blue")
> lines(random.prerec[, c(1, 2)], type = "l", lty = 2, lwd = 4,
+     col = "black")
> mtext("Recall (# RegulonDB interactions)", 3, line = +3)
> legend(0.06, 1, c("qp-graph", "PCC", "Random"), col = c("red",
+     "blue", "black"), lty = c(1, 1, 2), pch = c(19, 22, -1),
+     lwd = 3, bg = "white", pt.cex = 0.85)
```

The final step in this analysis is to get a transcriptional regulatory network from a qp-graph using avgNNRs and, if possible, obtain estimates of partial correlation coefficients (PAC) for the interactions. A qp-graph can be obtained by thresholding on the avgNRRs using the function `qpGraph`. When, as in our case now, we have a gold-standard network like RegulonDB, a sensible strategy to decide on a particular threshold is to derive it from a nominal precision level with respect to the gold-standard network. We can do this with the function `qpPRscoreThreshold` which reads the output of `qpPrecisionRecall` and takes a desired precision or recall level. We will use it with a nominal precision level of 50%:

```
> thr <- qpPRscoreThreshold(avgnrr.prerec, level = 0.5, recall.level = FALSE,
+     max.score = 0)
> thr
ScoreThreshold
        0.5175
```



(a) (b)

Figure 2: Comparison of precision-recall curves for various reverse-engineering methods with panel (a) showing a high-precision recall region of [0,0.1] and panel(b) showing the entire recall range.

```
> g <- qpGraph(avgnrr.estimates, threshold = thr, return.type = "graphNEL")
```

In the previous call to `qpGraph` we have set `return.type="graphNEL"` in order to get a `graphNEL` object which we can directly plot using the `Rgraphviz` library as we will show later, but before that we want to estimate the corresponding PACs for the interactions. First, we should see if this is at all possible by calculating the size of the largest clique in this undirected graph with the `qpCliqueNumber` function from the `qpgraph` package:

```
> qpCliqueNumber(g, verbose = FALSE)
```

```
[1] 2
```

The maximum clique size (aka clique number) is smaller than the number of observations in the data ($n = 43$) and therefore we can go on with the PAC estimation (see Lauritzen, 1996, for further details on this):

```
> pac.estimates <- qpPAC(subset.gds680.eset, g, verbose = FALSE)
```

In order to build the final reverse engineered transcriptional regulatory network from this 50%-precision qp-graph we need to transform the undirected graph `g` into a `graphNEL` directed graph with arrows pointing from TFs to their targets. This can be accomplished by first setting the `graphNEL` object `g` that stores the qp-graph as a directed graph:

```
> edgemode(g) <- "directed"
```

This operation leaves `g` with directed edges between all genes that were adjacent in the former undirected graph. Therefore, we should next remove those directed edges that go from TG genes to TF genes:

```
> gTFvertices <- nodes(g)[!is.na(match(nodes(g), filtered.regulon6.1[,
+     "EgID_TF"]))]
> wrongEdges <- boundary(setdiff(nodes(g), gTFvertices), g)
> wrongEdges <- wrongEdges[unlist(lapply(wrongEdges, length)) >
+     0]
> wrongEdges <- matrix(unlist(sapply(names(wrongEdges), function(x) t(cbind(x,
+     wrongEdges[[x]])), USE.NAMES = FALSE)), ncol = 2, byrow = TRUE)
> g <- removeEdge(from = wrongEdges[, 1], to = wrongEdges[, 2],
+     g)
```

Before making a graphical representation of the transcriptional regulatory network we have in g we would like to make a text-based summary of the interactions, more amenable for an occasional automatic processing of them outside R, including their presence or absence of RegulonDB and corresponding avgNRRs, PACs and PCCs. We start by building a matrix of the directed edges,

```
> edL <- edges(g)[names(edges(g))[unlist(lapply(edges(g), length)) >
+     0]]
> edM <- matrix(unlist(sapply(names(edL), function(x) t(cbind(x,
+     edL[[x]])), USE.NAMES = FALSE)), ncol = 2, byrow = TRUE)
```

and continue by gathering all the necessary information on these edges,

```
> edSymbols <- cbind(unlist(mget(edM[, 1], org.EcK12.egSYMBOL)),
+     unlist(mget(edM[, 2], org.EcK12.egSYMBOL)))
> idxTF <- match(edM[, 1], featureNames(subset.gds680.eset))
> idxTG <- match(edM[, 2], featureNames(subset.gds680.eset))
> nrrs <- avgnrr.estimates[cbind(idxTF, idxTG)]
> pacs.rho <- pac.estimates$R[cbind(idxTF, idxTG)]
> pacs.pva <- pac.estimates$P[cbind(idxTF, idxTG)]
> pccs.rho <- pcc.estimates$R[cbind(idxTF, idxTG)]
> pccs.pva <- pcc.estimates$P[cbind(idxTF, idxTG)]
> idxRegDB <- apply(edM, 1, function(x) {
+     regdbmask <- apply(cbind(match(subset.filtered.regulon6.1[,
+         "EgID_TF"], x[1]), match(subset.filtered.regulon6.1[,
+         "EgID_TG"], x[2])), 1, function(y) sum(!is.na(y))) ==
+         2
+     if (sum(regdbmask) > 0)
+         (1:dim(subset.filtered.regulon6.1)[1])[regdbmask]
+     else NA
+ })
> isinRegDB <- matrix(c("present", "absent"), nrow = 2, ncol = length(idxRegDB))[t(cbind(!is.na
+     is.na(idxRegDB)))]
```

to end up creating a data frame that includes all the information,

```
> txregnet <- data.frame(RegulonDB = isinRegDB, RegDBdir = subset.filtered.regulon6.1[idxRegDB,
+     "Direction"], AvgNRR = round(nrrs, digits = 2), PCC.rho = round(pccs.rho,
```

```
+     digits = 2), PCC.pva = format(pccs.pva, scientific = TRUE,
+     digits = 3), PAC.rho = round(pacs.rho, digits = 2), PAC.pva = format(pacs.pva,
+     scientific = TRUE, digits = 3))
> rownames(txregnet) <- paste(edSymbols[, 1], edSymbols[, 2], sep = " -> ")
```

and which allows us to display the transcriptional regulatory network as a list of edges ordering them, for instance, by the avgNRR from the stronger (0.0) to the weaker (1.0) support for the presence of that interaction in the network:

```
> txregnet[sort(txregnet[["AvgNRR"]], index.return = TRUE)$ix,
+     ]
```

	RegulonDB	RegDBdir	AvgNRR	PCC.rho	PCC.pva	PAC.rho	PAC.pva
appY -> appC	present	+	0.07	0.84	1.21e-12	0.46	2.35e-05
betI -> betB	present	-	0.08	0.99	0.00e+00	0.88	4.13e-06
arcA -> fadB	present	-	0.10	-0.90	5.23e-16	-0.74	1.22e-05
fnr -> yfiD	present	+-	0.11	0.66	1.39e-06	0.64	2.98e-04
fnr -> flu	absent	<NA>	0.18	0.32	3.90e-02	0.19	4.82e-02
appY -> flu	absent	<NA>	0.20	0.66	1.48e-06	0.25	3.04e-04
arcA -> lpd	present	-	0.34	-0.64	4.58e-06	-0.17	4.33e-04
appY -> appB	present	+	0.35	0.81	2.98e-11	0.42	3.45e-05
caiF -> narG	absent	<NA>	0.37	0.81	6.56e-11	0.51	3.84e-05
caiF -> dmsC	absent	<NA>	0.40	0.86	1.63e-13	0.63	1.92e-05
betI -> sucC	absent	<NA>	0.41	0.92	0.00e+00	0.36	8.62e-06
arcA -> glcB	present	-	0.43	-0.78	4.83e-10	-0.38	5.18e-05
betI -> lpd	absent	<NA>	0.44	0.89	3.11e-15	0.26	1.38e-05
lldR -> glcB	absent	<NA>	0.46	0.75	8.14e-09	0.20	8.57e-05
appY -> hyaF	present	+	0.46	0.75	6.18e-09	0.34	8.12e-05
lldR -> nrfA	absent	<NA>	0.48	-0.68	6.29e-07	-0.21	2.38e-04
lldR -> sucA	absent	<NA>	0.49	0.95	0.00e+00	0.69	6.44e-06
lldR -> aldA	absent	<NA>	0.50	0.93	0.00e+00	0.59	7.85e-06
caiF -> dcuC	absent	<NA>	0.50	0.78	8.22e-10	0.46	5.65e-05
soxS -> hyaB	absent	<NA>	0.50	-0.57	6.07e-05	-0.44	1.12e-03
appY -> hyaD	present	+	0.51	0.71	1.00e-07	0.30	1.48e-04
appY -> hyaC	present	+	0.52	0.68	6.63e-07	0.27	2.41e-04
appY -> hyaB	present	+	0.52	0.72	6.94e-08	0.27	1.36e-04
appY -> hyaE	present	+	0.52	0.71	8.68e-08	0.30	1.43e-04

We can plot the network, which should look like the one in Figure 3, by loading first the `Rgraphviz` library:

```
> library(Rgraphviz)
```

Next, get the genes forming connected components with more than 1 vertex (i.e., those involved in edges) using the `connComp` function from the `graph` package:

```
> gCc <- connComp(g)
> gCc_gt1 <- gCc[(1:length(gCc))[unlist(lapply(gCc, length)) >
+     1]]
> genesCcGt1 <- unique(unlist(gCc_gt1))
```

We extract the subgraph involving those genes, label them with their symbol names and make TF genes look darker:

```
> gSub <- subGraph(genesCcGt1, g)
> nodattr <- makeNodeAttrs(gSub, label = unlist(mget(nodes(gSub),
+     org.EcK12.egSYMBOL)), shape = "ellipse", fixedsize = FALSE,
+     fillcolor = grey(0.9))
> gSubTFnodes <- nodes(gSub)[!is.na(match(nodes(gSub), filtered.regulon6.1[,
+     "EgID_TF"]))]
> nodattr$fillcolor[gSubTFnodes] <- grey(0.65)
```

Set different colors for edges representing RegulonDB interactions, novel interactions, positive PACs and negative PACs:

```
> edgecolors <- rep("darkred", length(isinRegDB))
> edgecolors[isinRegDB == "absent" & pacs.rho > 0] <- "orange"
> edgecolors[isinRegDB == "present" & pacs.rho < 0] <- "darkblue"
> edgecolors[isinRegDB == "absent" & pacs.rho < 0] <- "deepskyblue"
> names(edgecolors) <- paste(edM[, 1], edM[, 2], sep = "~")
> edgattrs <- list(color = edgecolors)
```

We can finally do the plotting and put a legend as follows:

```
> rag <- plot(gSub, "twopi", nodeAttrs = nodattr, edgeAttrs = edgattrs,
+     lwd = 2, main = "50%-precision qp-graph transcriptional regulatory network")
> bb <- c(boundBox(rag)@upRight@x, boundBox(rag)@upRight@y)
> legend(bb[1] * 0.1, bb[2] * 1, c("RegulonDB +", "RegulonDB -",
+     "Novel +", "Novel -"), lwd = 4, col = c("darkred", "darkblue",
+     "orange", "deepskyblue"))
```

```
> toLatex(sessionInfo())
```

- R version 2.9.0 (2009-04-17), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=en_US;LC_NUMERIC=C;LC_TIME=en_US;LC_COLLATE=en_US;LC_MONETARY=C;LC_MESSAGE`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, tools, utils
- Other packages: annotate 1.22.0, AnnotationDbi 1.6.0, Biobase 2.4.0, DBI 0.2-4, genefilter 1.24.0, graph 1.22.0, org.EcK12.eg.db 2.2.11, qpgraph 1.0.0, Rgraphviz 1.22.0, RSQLite 0.7-1
- Loaded via a namespace (and not attached): cluster 1.11.13, splines 2.9.0, survival 2.35-4, xtable 1.5-5

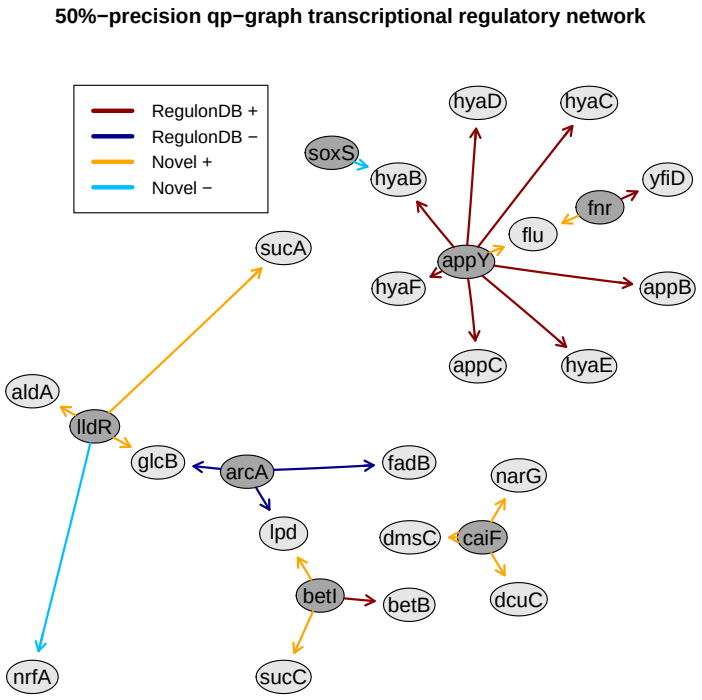


Figure 3: Reverse-engineered transcriptional network using a qp-graph at a nominal 50% precision.

References

Barrett, T., Troup, D. B., Wilhite, S. E., Ledoux, P., Rudnev, D., Evangelista, C., Kim, I. F., Soboleva, A., Tomashevsky, M., and Edgar, R. (2007). NCBI GEO: mining tens of millions of expression profiles–database and tools update. *Nucleic Acids Res*, 35(Database issue):D760–5.

Castelo, R. and Roverato, A. (2006). A robust procedure for gaussian graphical model search from microarray data with p larger than n. *J Mach Learn Res*, 7:2621–2650.

Castelo, R. and Roverato, A. (2008). Reverse engineering molecular regulatory networks from microarray data with qp-graphs. *J Comput Biol, accepted*.

Covert, M. W., Knight, E. M., Reed, J. L., Herrgard, M. J., and Palsson, B. O. (2004). Integrating high-throughput and computational data elucidates bacterial networks. *Nature*, 429(6987):92–96.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recogn Lett*, 27:861–874.

Gama-Castro, S., Jimenez-Jacinto, V., Peralta-Gil, M., Santos-Zavaleta, A., Penaloza-Spinola, M. I., Contreras-Moreira, B., Segura-Salazar, J., Muniz-Rascado, L., Martinez-Flores, I., Salgado, H., Bonavides-Martinez, C., Abreu-Goodger, C., Rodriguez-Penagos, C., Miranda-Rios, J., Morett, E., Merino, E., Huerta, A. M., Trevino-Quintanilla, L., and Collado-Vides, J. (2008). RegulonDB (version 6.0): gene regulation model of Escherichia coli K-12 beyond transcription, active (experimental) annotated promoters and textpresso navigation. *Nucleic Acids Res*, 36(Database issue):D120–4.

Lauritzen, S. (1996). *Graphical models*. Oxford University Press.