Microarray Analysis

using Affymetrix Arrays

Steven Buechler

Department of Mathematics
276B Hurley Hall; 1-6233

Fall, 2007

Outline

Just the Highlights

for analyzing resulting data

- A microarray contains oligonucleotide “probes” that bind mRNA from a sample.
- There may be numerous probes from the coding regions of any given gene.
- mRNA is flourescence labelled and the data captured as an image. Image intensity is correlated with the amount of mRNA.
- One array is hybridized with mRNA from one sample.
- Quality assessment is important. Many arrays need to be discarded.

Just the Highlights

for analyzing resulting data

- A microarray contains oligonucleotide "probes" that bind mRNA from a sample.
- There may be numerous probes from the coding regions of any given gene.
- mRNA is flourescence labelled and the data captured as an image. Image intensity is correlated with the amount of mRNA.
- One array is hybridized with mRNA from one sample.
- Quality assessment is important. Many arrays need to be discarded.

Just the Highlights

for analyzing resulting data

- A microarray contains oligonucleotide “probes” that bind mRNA from a sample.
- There may be numerous probes from the coding regions of any given gene.
- mRNA is flourescence labelled and the data captured as an image. Image intensity is correlated with the amount of mRNA.
- One array is hybridized with mRNA from one sample.
- Quality assessment is important. Many arrays need to be discarded.

Just the Highlights

for analyzing resulting data

- A microarray contains oligonucleotide “probes” that bind mRNA from a sample.
- There may be numerous probes from the coding regions of any given gene.
- mRNA is flourescence labelled and the data captured as an image. Image intensity is correlated with the amount of mRNA.
- One array is hybridized with mRNA from one sample.
- Quality assessment is important. Many arrays need to be discarded.

Just the Highlights

for analyzing resulting data

- A microarray contains oligonucleotide “probes” that bind mRNA from a sample.
- There may be numerous probes from the coding regions of any given gene.
- mRNA is flourescence labelled and the data captured as an image. Image intensity is correlated with the amount of mRNA.
- One array is hybridized with mRNA from one sample.
- Quality assessment is important. Many arrays need to be discarded.

From Image to Expression Values

- Initial processing produces an intensity level for each probe cell. This is a start at measuring expression level. The resulting data is stored in a file xxx.CEL and called a .CEL file.
- A meaningful hypothesis driven experiment requires replicates and often different biological traits. We need to compare and contrast assays from different samples. This requires calculating probe expression levels that are “normalized” across arrays.
- The process of moving from a set of .CEL files to a set of expression levels for all samples in the experiment has several component processes. In *R* the result is an ExpressionSet object.

From Image to Expression Values

- Initial processing produces an intensity level for each probe cell. This is a start at measuring expression level. The resulting data is stored in a file xxx.CEL and called a .CEL file.
- A meaningful hypothesis driven experiment requires replicates and often different biological traits. We need to compare and contrast assays from different samples. This requires calculating probe expression levels that are "normalized" across arrays.
- The process of moving from a set of .CEL files to a set of expression levels for all samples in the experiment has several component processes. In *R* the result is an ExpressionSet object.

From Image to Expression Values

- Initial processing produces an intensity level for each probe cell. This is a start at measuring expression level. The resulting data is stored in a file xxx.CEL and called a .CEL file.
- A meaningful hypothesis driven experiment requires replicates and often different biological traits. We need to compare and contrast assays from different samples. This requires calculating probe expression levels that are "normalized" across arrays.
- The process of moving from a set of .CEL files to a set of expression levels for all samples in the experiment has several component processes. In *R* the result is an ExpressionSet object.

.CEL Files to ExpressionSet

There are several steps to the process, and several accepted methods, based on different algorithms. This is a complicated subject, still evolving, that borrows from image processing and molecular biology.

Two major issues all methods must address are background correction and normalization.

Common methods are MAS 5.0, RMA, and GCRMA.

.CEL Files to ExpressionSet

There are several steps to the process, and several accepted methods, based on different algorithms. This is a complicated subject, still evolving, that borrows from image processing and molecular biology.

Two major issues all methods must address are background correction and normalization.

Common methods are MAS 5.0, RMA, and GCRMA.

.CEL Files to ExpressionSet

There are several steps to the process, and several accepted methods, based on different algorithms. This is a complicated subject, still evolving, that borrows from image processing and molecular biology.

Two major issues all methods must address are background correction and normalization.

Common methods are MAS 5.0, RMA, and GCRMA.

Background Correction

In image processsing there is always an issue in measuring true signal versus background noise.

- The MAS 5.0 method developed by Affymetrix does averaging over regions in the array for both PM and MM probe cells.
- The RMA method by Irrizary, *et al*, uses a statistical model of exponential signal and normal noise on only the PM probe cells.

Background Correction

In image processsing there is always an issue in measuring true signal versus background noise.

- The MAS 5.0 method developed by Affymetrix does averaging over regions in the array for both PM and MM probe cells.
- The RMA method by Irrizary, *et al*, uses a statistical model of exponential signal and normal noise on only the PM probe cells.

Normalization

- The Affymetrix scaling method is to select one array as a baseline and then then scale all others to have the same mean intensity as this one.
- The quantile normalization method, used by RMA and GCRMA, imposes the same empirical distribution on all arrays. Array measurements are transformed until all Q-Q plots are linear and diagonal (as much as possible).

Normalization

- The Affymetrix scaling method is to select one array as a baseline and then then scale all others to have the same mean intensity as this one.
- The quantile normalization method, used by RMA and GCRMA, imposes the same empirical distribution on all arrays. Array measurements are transformed until all Q-Q plots are linear and diagonal (as much as possible).

GCRMA

GCRMA is refine ment to RMA that adds a step of adjusting expression values based on the propensity of some probes to undergo non-specific binding. The authors argue that it adds a level of precision to the numbers – they are closer to measuring real concentrations.

What are Expression Values?

- The numbers on an ExpressionSet created with MAS 5.0 are on the same scale as the signal intensities and range from 100 to several thousand. Normally a threshold of 500 is used to decide if the gene is expressed at all. Amount of mRNA is roughly proportional to the measure.
- Expression values in a set created with RMA or GCRMA are on a log2 scale. Values range from 2 to 15 or so. Amount of mRNA is roughly 2^x, where x is the expression value.
- In comparing expression levels of a particular probe across two samples or groups of samples the term fold change is sometimes used. A doubling of mRNA amount is a 2 fold change. To the extent that this is captured by expression values this means a doubling of number in MAS 5.0 data, or an increase of 1 for RMA or GCRMA.

What are Expression Values?

- The numbers on an ExpressionSet created with MAS 5.0 are on the same scale as the signal intensities and range from 100 to several thousand. Normally a threshold of 500 is used to decide if the gene is expressed at all. Amount of mRNA is roughly proportional to the measure.
- Expression values in a set created with RMA or GCRMA are on a log2 scale. Values range from 2 to 15 or so. Amount of mRNA is roughly 2^x, where x is the expression value.
- In comparing expression levels of a particular probe across two samples or groups of samples the term fold change is sometimes used. A doubling of mRNA amount is a 2 fold change. To the extent that this is captured by expression values this means a doubling of number in MAS 5.0 data, or an increase of 1 for RMA or GCRMA.

What are Expression Values?

- The numbers on an ExpressionSet created with MAS 5.0 are on the same scale as the signal intensities and range from 100 to several thousand. Normally a threshold of 500 is used to decide if the gene is expressed at all. Amount of mRNA is roughly proportional to the measure.
- Expression values in a set created with RMA or GCRMA are on a log2 scale. Values range from 2 to 15 or so. Amount of mRNA is roughly 2^x, where x is the expression value.
- In comparing expression levels of a particular probe across two samples or groups of samples the term fold change is sometimes used. A doubling of mRNA amount is a 2 fold change. To the extent that this is captured by expression values this means a doubling of number in MAS 5.0 data, or an increase of 1 for RMA or GCRMA.

Outline

Outline of the Steps

CEL files to ExpressionSet

- Collect the .CEL files in a separate directory. Optionally include a text file table of tab separated phenotypic data, one line for each sample.
- Use `ReadAffy()` in the `affy` package to create an `AffyBatch` object. This has an intensity measure for each probe cell, and for each array. It is a matrix-like object.
- Use functions from the `simpleaffy` or the `affyQCReport` package to check the quality of arrays. Exclude any samples that fail the quality check and generate a new AffyBatch object.
- Generate the final ExpressionSet object with `expresso`, `rma` (in the `affy` package) or `gcrma` (in the `gcrma` package).
- Other slots of the ExpressionSet object can be filled in "by hand". See the PDF on the ExpressionSet class in `Biobase`.

Outline of the Steps

CEL files to ExpressionSet

- Collect the .CEL files in a separate directory. Optionally include a text file table of tab separated phenotypic data, one line for each sample.
- Use `ReadAffy()` in the `affy` package to create an `AffyBatch` object. This has an intensity measure for each probe cell, and for each array. It is a matrix-like object.
- Use functions from the `simpleaffy` or the `affyQCReport` package to check the quality of arrays. Exclude any samples that fail the quality check and generate a new AffyBatch object.
- Generate the final ExpressionSet object with `expresso`, `rma` (in the `affy` package) or `gcrma` (in the `gcrma` package).
- Other slots of the ExpressionSet object can be filled in "by hand". See the PDF on the ExpressionSet class in `Biobase`.

Outline of the Steps

CEL files to ExpressionSet

- Collect the .CEL files in a separate directory. Optionally include a text file table of tab separated phenotypic data, one line for each sample.
- Use `ReadAffy()` in the `affy` package to create an `AffyBatch` object. This has an intensity measure for each probe cell, and for each array. It is a matrix-like object.
- Use functions from the `simpleaffy` or the `affyQCReport` package to check the quality of arrays. Exclude any samples that fail the quality check and generate a new AffyBatch object.
- Generate the final ExpressionSet object with `expresso`, `rma` (in the `affy` package) or `gcrma` (in the `gcrma` package).
- Other slots of the ExpressionSet object can be filled in "by hand". See the PDF on the ExpressionSet class in `Biobase`.

Outline of the Steps

CEL files to ExpressionSet

- Collect the .CEL files in a separate directory. Optionally include a text file table of tab separated phenotypic data, one line for each sample.
- Use `ReadAffy()` in the `affy` package to create an `AffyBatch` object. This has an intensity measure for each probe cell, and for each array. It is a matrix-like object.
- Use functions from the `simpleaffy` or the `affyQCReport` package to check the quality of arrays. Exclude any samples that fail the quality check and generate a new AffyBatch object.
- Generate the final ExpressionSet object with `expresso`, `rma` (in the `affy` package) or `gcrma` (in the `gcrma` package).
- Other slots of the ExpressionSet object can be filled in "by hand". See the PDF on the ExpressionSet class in Biobase.

Outline of the Steps

CEL files to ExpressionSet

- Collect the .CEL files in a separate directory. Optionally include a text file table of tab separated phenotypic data, one line for each sample.
- Use `ReadAffy()` in the `affy` package to create an `AffyBatch` object. This has an intensity measure for each probe cell, and for each array. It is a matrix-like object.
- Use functions from the `simpleaffy` or the `affyQCReport` package to check the quality of arrays. Exclude any samples that fail the quality check and generate a new AffyBatch object.
- Generate the final ExpressionSet object with `expresso`, `rma` (in the `affy` package) or `gcrma` (in the `gcrma` package).
- Other slots of the ExpressionSet object can be filled in "by hand". See the PDF on the ExpressionSet class in `Biobase`.

Outline

Example with Arrays from Mouse Experiment

The arrays are from an experiment using a mouse model of colon cancer, Apc$^{Min/+}$. In a very small nutshell, the goal is to assess the effect of an NSAID, sulindac, on gene expression in colon adenomas. The mice were divided into two populations, those receiving sulindac and a control group that only received the vehicle by which the drug was administered. These two words will label the phenotypes. The samples will be coded with “S” and “V” to further distinguish.

First Inspect the CEL File Directory

```
> list.files(path = "/Users/steve/Documents/Bio/Castellino/

 [1] "S1.cel"          "S2.cel"
 [3] "S3.cel"          "S4.cel"
 [5] "S5.cel"          "S6.cel"
 [7] "S7.cel"          "S8.cel"
 [9] "S9.cel"          "V1.cel"
[11] "V2.cel"          "V3.cel"
[13] "V4.cel"          "V5.cel"
[15] "V6.cel"          "V7.cel"
[17] "V8.cel"          "V9.cel"
[19] "phenodata.txt"
```

What Does phenodata.txt Look ?

The phenodata.txt file contains the labels needed to separate the two groups.

```
> phenofilepath <- "/Users/steve/Documents/Bio/Castellino/s
> pd1 <- read.table(file = phenofilepath,
+     header = TRUE)
> pd1[1:3, ]
```

```
[1] S S S
Levels: S V
```

```
> pd1[15:18, ]
```

```
[1] V V V V
Levels: S V
```

Read CEL files into AffyBatch Object

Load the `affy` library and create an AffyBatch object with the `ReadAffy()` function. This can simultaneously read in the phenodata from the file specified.

```
> library(affy)
```

Read CEL files into AffyBatch Object
continued

```
> celpath <- "/Users/steve/Documents/Bio/Castellino/sulinda
> batch1 <- ReadAffy(celfile.path = celpath,
+     phenoData = phenofilepath)

> batch1

AffyBatch object
size of arrays=712x712 features (10 kb)
cdf=MOE430A (22690 affyids)
number of samples=18
number of genes=22690
annotation=moe430a
notes=
```

Browse the AffyBatch Object

The AffyBatch object is very basic. It just contains basic uncorrected intensity measures for the probe cells. To learn about the slot names and applicable methods execute

```
> class?AffyBatch
```

Just to see what slots make up the object:

```
> slotNames(batch1)
```

```
[1] "cdfName"              "nrow"
[3] "ncol"                 "assayData"
[5] "phenoData"            "featureData"
[7] "experimentData"       "annotation"
[9] ".__classVersion__"
```

The phenoData is Here

```
> phenoData(batch1)

  rowNames: S1.cel, S2.cel, ..., V9.cel (18 total)
  varLabels and varMetadata:
    treatment: read from file

> phenoData(batch1)$treatment

 [1] S S S S S S S S S V V V V V V V V V
Levels: S V
```

Outline

Next Step is Quality Assessment

For Affymetrix arrays there are several quality checks that should be done. These can all be done through one of several packages, like `simpleaffy` or `affyQCReport`. The tests here were developed by Affymetrix to judge the quality of the hybridization, relative RNA concentrations, scanning quality, etc. Read the documentation in QCandSimpleaffy.pdf. I won't try to explain these steps, just illustrate why some arrays were rejected.

First Check the Image Quality

The crudest check is of basic image quality. This can spot scratches and other anomalies on the array. Executing `image(batch1)` produces 18 plots, one at a time. Here we plot 4 interesting ones, all in one 2×2 matrix.

```
> oldpar <- par(mfrow = c(2, 2))
> image(batch1[, 5:8])
> par(oldpar)
```

Execute the QC Function

The main function in `simpleaffy` for quality control is qc. It returns an object of class QCstats.

```
> qc.batch1 <- qc(batch1)
```

```
> slotNames(qc.batch1)
```

```
[1] "scale.factors"      "target"
[3] "percent.present"    "average.background"
[5] "minimum.background" "maximum.background"
[7] "spikes"             "qc.probes"
[9] "bioBCalls"
```

Inspect the QC Results

Average Background

```
> qc.batch1@average.background

S1.cel S2.cel S3.cel S4.cel S5.cel S6.cel S7.cel
105.76 162.13 106.99  74.46 100.96  72.28 145.50
S8.cel S9.cel V1.cel V2.cel V3.cel V4.cel V5.cel
109.04  64.95 157.94 117.82 116.47  77.87  99.23
V6.cel V7.cel V8.cel V9.cel
 64.91  57.58 109.13  77.59
```

S2, S7 and V1 are problematic. This could be due to differing concentrations in RNA, incorporating more or less label, or other factors in producing images with varying overall intensity.

Inspect the QC Results

Scale Factor

```
> qc.batch1@scale.factors

 [1] 0.6192 0.5281 0.6585 0.6132 0.4181 0.5863
 [7] 0.7668 0.5760 1.0212 0.4765 0.5405 0.6316
[13] 0.9635 0.8385 0.8244 1.1304 0.6489 0.7243
```

Scale factor is a measure of how mean intensities vary across the arrays. They should be within 3-fold of each other. These are OK.

Inspect the QC Results

Percent Present

```
> qc.batch1@percent.present[1:15]
S1.cel.present S2.cel.present S3.cel.present
         44.06          44.57          43.13
S4.cel.present S5.cel.present S6.cel.present
         47.29          48.10          49.38
S7.cel.present S8.cel.present S9.cel.present
         37.56          47.41          45.40
V1.cel.present V2.cel.present V3.cel.present
         43.26          43.26          41.95
V4.cel.present V5.cel.present V6.cel.present
         48.97          47.18          47.43
```

A probe pair is present if PM is significantly larger than MM. This number measure the percentage of pairs present. It should be between 35 and 55, with little variation across the arrays. OK here.

Conclusions from QC

Should exclude S5 due to the spot. We should also eliminate S2, S7 and V1 for excessive average background.

The loss in degrees of freedom is painful, however bad arrays increase variance and make it harder to find differentially expressed genes.

Create a new directory of CEL files containing only the good ones.

Outline

Create New AffyBatch Object

using the good CEL files

First, redo the steps generating the AffyBatch object using only the good CEL files.

```
> newcelpath <- "/Users/steve/Documents/Bio/Rcourse/Lect10/
> newphenopath <- "/Users/steve/Documents/Bio/Rcourse/Lect1
> batch2 <- ReadAffy(celfile.path = newcelpath,
+     phenoData = newphenopath)
> batch2
AffyBatch object
size of arrays=712x712 features (10 kb)
cdf=MOE430A (22690 affyids)
number of samples=14
number of genes=22690
annotation=moe430a
notes=
```

Run GCRMA

Executing GCRMA does background correction and normalization, resulting in an ExpressionSet object.

```
> library(gcrma)

> sulEset1 <- gcrma(batch2)
```

GCRMA Output

It takes some time

```
Adjusting for optical effect.............Done.
Computing affinities.Done.
Adjusting for non-specific binding.............Done.
Normalizing
Calculating Expression
```

The Resulting Object

```
> sulEset1

ExpressionSet (storageMode: lockedEnvironment)
assayData: 22690 features, 14 samples
  element names: exprs
phenoData
  rowNames: S1.cel, S3.cel, ..., V9.cel (14 total)
  varLabels and varMetadata:
    treatment: read from file
featureData
  featureNames: 1415670_at, 1415671_at, ..., AFFX-r2-P1-cr
  varLabels and varMetadata: none
experimentData: use 'experimentData(object)'
Annotation [1] "moe430a"
```

Important Slots

The point of this is the expression values. The matrix is extracted with the exprs method (function).

```
> expSul <- exprs(sulEset1)
> expSul[1:4, 1:3]
```

```
           S1.cel S3.cel S4.cel
1415670_at  8.045  8.336  7.983
1415671_at  8.119  8.615  8.664
1415672_at 12.004 11.444 11.655
1415673_at  3.566  3.336  3.297
```

Samples and Probes (Features)

```
> sampleNames(sulEset1)

 [1] "S1.cel" "S3.cel" "S4.cel" "S6.cel" "S8.cel"
 [6] "S9.cel" "V2.cel" "V3.cel" "V4.cel" "V5.cel"
[11] "V6.cel" "V7.cel" "V8.cel" "V9.cel"

> featureNames(sulEset1)[1:6]

[1] "1415670_at"   "1415671_at"   "1415672_at"
[4] "1415673_at"   "1415674_a_at" "1415675_at"
```

Expand on Supporting Info

The phenoData slot has minimal information. Let's give a better explanation of the treatment. We must unravel the phenoData object and learn what slot of it should be altered and how to do it. Extract the slot and work with it on the side.

```
> pd <- phenoData(sulEset1)
> slotNames(pd)
```

```
[1] "varMetadata"        "data"
[3] "dimLabels"          ".__classVersion__"
```

PhenoData Data

```
> pd@data
         treatment
S1.cel           S
S3.cel           S
S4.cel           S
S6.cel           S
S8.cel           S
S9.cel           S
V2.cel           V
V3.cel           V
V4.cel           V
V5.cel           V
V6.cel           V
V7.cel           V
V8.cel           V
```

Editing the phenoData

```
> pd@varMetadata
          labelDescription
treatment   read from file
> class(pd@varMetadata)
[1] "data.frame"
> dim(pd@varMetadata)
[1] 1 1
```

Editing the phenoData

```
> pd@varMetadata[1, 1] <- "Sulindac treated versus vehicle
> phenoData(sulEset1) <- pd
> sulEset1
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 22690 features, 14 samples
  element names: exprs
phenoData
  rowNames: S1.cel, S3.cel, ..., V9.cel (14 total)
  varLabels and varMetadata:
    treatment: Sulindac treated versus vehicle (control)
featureData
  featureNames: 1415670_at, 1415671_at, ..., AFFX-r2-P1-cre
  varLabels and varMetadata: none
experimentData: use 'experimentData(object)'
Annotation [1] "moe430a"
```

Fill in Other Slots

The experimentData slot contains the MIAME information needed to properly attribute the data. This is empty until you create the object and assign it to the slot.

Outline

Tie Probes to Genes

and what we know about them

Interpreting the result of an array analysis requires connecting probes to genes and the available biological information about the genes. The Annotation slot in the ExpressionSet object reports the array type. This was read from the CEL file. The annotation information about the probes are accessed through a package with the same name as the array.

```
> library(moe430a)
```

Annotation Consists of Environments

An environment in *R* is a set of key-value pairs. Here the keys are probe IDS and the values are traits of the corresponding oligonucleotide; or the key is a code for a trait and the value is a vector of probe IDS with that trait. Documentation is not good, but all packages and environments are accessed the same way.

After loading the library (`moe430a` in our case) the list of environments is generated by

```
> moe430a()
```

Environment List

```
Quality control information for moe430a
Date built: Created: Mon Apr 23 12:48:20 2007

Number of probes: 22690
Probe number mismatch: None
Probe missmatch: None
Mappings found for probe based rda files:
        moe430aACCNUM found 22690 of 22690
        moe430aCHR found 22360 of 22690
        moe430aCHRLOC found 21260 of 22690
        moe430aENZYME found 2787 of 22690
        moe430aENTREZID found 22388 of 22690
        moe430aGENENAME found 22388 of 22690
        moe430aGO found 20374 of 22690
        moe430aMAP found 21712 of 22690
```

Environment List

```
        moe430aPATH found 6035 of 22690
        moe430aPFAM found 21854 of 22690
        moe430aPMID found 22289 of 22690
        moe430aPROSITE found 21854 of 22690
        moe430aREFSEQ found 21972 of 22690
        moe430aSYMBOL found 22388 of 22690
        moe430aUNIGENE found 22289 of 22690
Mappings found for non-probe based rda files:
        moe430aCHRLENGTHS found 21
        moe430aENZYME2PROBE found 737
        moe430aGO2ALLPROBES found 7873
        moe430aGO2PROBE found 5784
        moe430aPATH2PROBE found 184
        moe430aPMID2PROBE found 95903
```

Example Environment Values

Select a probe ID and extract the values of different environments for this "key" as follows. Work with the probe 1423110_at.

```
> moe430aSYMBOL$"1423110_at"

[1] "Col1a2"

> moe430aGENENAME$"1423110_at"

[1] "procollagen, type I, alpha 2"

> moe430aUNIGENE$"1423110_at"

[1] "Mm.277792"

> moe430aCHRLOC$"1423110_at"

      6
4455696
```

Getting Many Values

Normally we don't just want values for a single probe. We've found a list of probes that are differentially expressed and we want the list of corresponding symbols. The relevant function in *R* is `mget`.

A vector of probe IDs is provided as `prbVec`. It has length 10.

Mget the Symbols

```
> prbSymsL <- mget(prbVec, envir = moe430aSYMBOL)
> class(prbSymsL)
```

```
[1] "list"
```

```
> prbSyms <- unlist(prbSymsL)
> prbSyms
```

```
  1423110_at   1424131_at 1434369_a_at
    "Col1a2"     "Col6a3"      "Cryab"
1454959_s_at   1421934_at   1460336_at
     "Gnai1"       "Cbx5"   "Ppargc1a"
  1416194_at 1438651_a_at   1416040_at
    "Cyp4b1"     "Agtrl1"       "Lipf"
  1417292_at
     "Ifi47"
```

Mget Returns a List

with names the probes

Normally the values in an environment aren't simple characters or numbers. `mget` must return a list to capture the information.

Getting Functional Information

Just having a list of gene names may not help understand the biology. More useful is to fold in information about the biological processes involving these genes. Bioconductor annotations use the Gene Ontology (GO) `http://www.geneontology.org/`, and pathway information at KEGG `http://www.genome.jp/kegg/`. You should read the documentation and browse the sites (especially GO) to understand this functional information.

Get Associated GO Terms

and then dig deeper

A list of GO terms associated with a probe can be obtained as follows. We use our collagen probe as an example and store the result for further inspection.

```
> colGO <- moe430aGO$"1423110_at"
> class(colGO)
```

```
[1] "list"
```

```
> names(colGO)
```

```
 [1] "GO:0006817" "GO:0007155" "GO:0007169"
 [4] "GO:0005578" "GO:0005581" "GO:0005615"
 [7] "GO:0005737" "GO:0005198" "GO:0005201"
[10] "GO:0005515" "GO:0030020"
```

Digging Deeper into GO Terms

What does `colGO` contain for one of the components; i.e., one of the Go terms?

```
> colGO[[1]]

$GOID
[1] "GO:0006817"

$Evidence
[1] "IEA"

$Ontology
[1] "BP"
```

Digging Deeper into GO Terms

It's possible to simply go to the GO site, enter the term into the search box and inspect the results. But, if there are many terms it may be easier to use Bioconductor's internal tools. In the GO package there is an environment, called `GOTERMS`, from which you can extract the English description. To use this, load two more libraries.

```
> library(annotate)
> library(GO)
```

What the Terms Mean

```
> GOTERM$"GO:0006817"
GOID = GO:0006817

Term = phosphate transport

Definition = The directed movement of
     phosphate into, out of, within or
     between cells.

Ontology = BP
```

Can Apply over Vectors

If you have a vector of GO term IDs, like the `names(colGO)`, you can extract all of the Term slots of the corresponding GOTERM objects using `lapply` and some good programming.

Probes for a Given GO term

At least as important as finding GO terms for a given probe is finding all probes on the array associated with a given term. A search of the GO website reveals GO terms of special interest, like `GO:0003700`, "transcription factor activity". Then the associated probes are found as follows.

```
> tfPrbs <- moe430aGO2PROBE$"GO:0003700"
> length(tfPrbs)
```

```
[1] 1379
```

This will be used later to focus differential expression analyses to selected processes of particular interest.

Annotate has Many Uses

The `annotate` package has many additional functions that perform Web queries to NCBI to get data for a given GenBank accession number. For example, `getSEQ` gets the sequence; `getPMID` fetches a vector of PubMed IDs referencing the nucleotide.

KEGG Pathway Annotations

KEGG has created databases of pathways, linked to information about the component genes. KEGG assigns a 5 digit code to each path. Bioconductor metadata packages can access these associations. For a given probe, the environment `moe430aPATH` finds all paths using an associated protein. Conversely, given a KEGG pathway code, `moe430aPATH2PROBE` gives the vector of probes involved in the pathway. This can be useful for focusing a differnetial expression experiment on a given pathway. The caveat is that pathway annotations are far from complete.