goseq: Gene Ontology testing for RNA-seq datasets

Matthew D. Young Matthew J. Wakefield Gordon K. Smyth
myoung@wehi.edu.au

Alicia Oshlack

30 March 2011

1 Introduction

This document gives an introduction to the use of the goseq R Bioconductor package [Young et al., 2010]. This package provides methods for performing Gene Ontology analysis of RNA-seq data, taking length bias into account [Oshlack and Wakefield, 2009]. The methods and software used by goseq are equally applicable to other category based test of RNA-seq data, such as KEGG pathway analysis.

Once installed, the goseq package can be easily loaded into R using:

```
> library(goseq)
```

In order to perform a GO analysis of your RNA-seq data, goseq only requires a simple named vector, which contains two pieces of information.

1. `Measured genes`: all genes for which RNA-seq data was gathered for your experiment. Each element of your vector should be named by a unique gene identifier.

2. `Differentially expressed genes`: each element of your vector should be either a 1 or a 0, where 1 indicates that the gene is differentially expressed and 0 that it is not.

If the organism, gene identifier or category test is currently not natively supported by goseq, it will also be necessary to supply additional information regarding the genes length and/or the association between categories and genes.

A combination of bioconductor R packages such as GenomicFeatures and Rsamtools allow for the summarization of mapped reads into a table of counts, such as reads per gene. From there, several packages exist for performing differential expression analysis on summarized data

(eg. edgeR [Robinson and Smyth, 2007, 2008, Robinson et al., 2010]). goseq will work with any method for determining differential expression and as such differential expression analysis is outside the scope of this document, but in order to facilitate ease of use, we will make use of the edgeR package to calculate differentially expressed (DE) genes in all the case studies in this document.

2 Reading data

We assume that the user can use appropriate in-built R functions (such as read.table or scan) to obtain two vectors, one containing all genes assayed in the RNA-seq experiment, the other containing all genes which are DE. If we assume that the vector of genes being assayed is named assayed.genes and the vector of DE genes is named de.genes we can construct a named vector suitable for use with goseq using the following:

```
> gene.vector = as.integer(assayed.genes %in% de.genes)
> names(gene.vector) = assayed.genes
> head(gene.vector)
```

It may be that the user can already read in a vector in this format, in which case it can then be immediately used by goseq.

3 GO testing of RNA-seq data

To begin the analysis, goseq first needs to quantify the length bias present in the dataset under consideration. This is done by calculating a Probability Weighting Function or PWF which can be thought of as a function which gives the probability that a gene will be differentially expressed (DE), based on its length alone. The PWF is calculated by fitting a monotonic spline to the binary data series of differential expression (1=DE, 0=Not DE) as a function of gene length. The PWF is used to weight the chance of selecting each gene when forming a null distribution for GO category membership. The fact that the PWF is calculated directly from the dataset under consideration makes this approach robust, only correcting for the length bias present in the data. For example, if goseq is run on a microarray dataset, for which no length bias exists, the calculated PWF will be nearly flat and all genes will be weighted equally, resulting in no length bias correction.

In order to account for the length bias inherent to RNA-seq data when performing a GO analysis (or other category based tests), one cannot simply use the hypergeometric distribution as the null distribution for category membership, which is appropriate for data without DE length bias, such as microarray data. GO analysis of RNA-seq data requires the use of random sampling in order to generate a suitable null distribution for GO category membership and calculate each categories significance for over representation amongst DE genes.

However, this random sampling is computationally expensive. In most cases, the Wallenius distribution can be used to approximate the true null distribution, without any significant loss in

accuracy. The goseq package implements this approximation as its default option. The option to generate the null distribution using random sampling is also included as an option.

Having established a null distribution, each GO category is then tested for over and under representation amongst the set of differentially expressed genes and the null is used to calculate a p-value for under and over representation.

4 Natively supported Gene Identifiers and category tests

goseq needs to know the length of each gene, as well as what GO categories (or other categories of interest) each gene is associated with. goseq relies on the UCSC genome browser to provide the length information for each gene. However, because the process of fetching the length of every transcript is slow and bandwidth intensive, goseq relies on an offline copy of this information stored in the data package geneLenDataBase. To see which genome/gene identifier combinations are in the local database, simply run:

```
> supportedGenomes()
```

or

```
> supportedGeneIDs()
```

The rightmost column in the output of both these commands lists the identifiers (in the case of supportedGenomes) or genomes (in the case of supportedGeneIDs) for which length data exists in the local database. If your genome/ID combination is not in the local database, it will be downloaded from the UCSC genome browser. If your genome/ID combination is not in the UCSC database, you will have to manually specify the gene lengths.

In order to link GO categories to genes, goseq uses the organism packages from Bioconductor. These packages are named org.<Genome>.<ID>.db, where <Genome> is a short string identifying the genome and <ID> is a short string identifying the gene identifier. Currently, goseq will automatically retrieve the mapping between GO categories and genes from the relevant package (as long as it is installed) for commonly used genome/ID combinations. If GO mappings are not automatically available for your genome/ID combination, you will have to manually specify the relationship between genes and categories. Although the Genome/ID naming conventions used by the organism packages differ from the UCSC, goseq is able to convert between the two, so the user need only ever specify the UCSC genome/ID in most cases.

5 Non-native Gene Identifier or category test

If the organism, Gene Identifier or category test you wish to perform is not in the native goseq database, you will have to supply one or all of the following:

- `Length data`: the length of each gene in your gene identifier format.
- `Category mappings`: the mapping (usually many-to-many) between the categories you wish to test for over/under representation amongst DE genes and genes in your gene identifier format.

5.1 Length data format

The length data must be formatted as a numeric vector, of the same length as the main named vector specifying gene names/DE genes. Each entry should give the length of the corresponding gene in bp. If length data is unavailable for some genes, that entry should be set to NA.

5.2 Category mapping format

The mapping between category names and genes should be given as a data frame with two columns. One column should contain the gene IDs and the other the name of an associated category. As the mapping between categories and genes is usually many-to-many, this data frame will usually have multiple rows with the same gene name and category name.

Alternatively, mappings between genes and categories can be given as a list. The names of list entries should be gene IDs and the entries themselves should be a vector of category names to which the gene ID corresponds.

5.3 Some additional tips

Any organism for which there is an annotation on either Ensembl or the UCSC, can be easily turned into length data using the GenomicFeatures package. To do this, first create a TranscriptDb object using either makeTranscriptDbFromBiomart or makeTranscriptDbFromUCSC (see the help in the GenomicFeatures package on using these commands). Once you have a transcriptDb object, you can get a vector named by gene ID containing the median transcript length of each gene simply by using the command.

```
> txsByGene = transcriptsBy(txdb, "gene")
> lengthData = median(width(txsByGene))
```

6 Case study: Prostate cancer data

6.1 Introduction

This section provides an analysis of data from an RNA-seq experiment to illustrate the use of `goseq` for GO analysis.

This experiment examined the effects of androgen stimulation on a human prostate cancer cell line, LNCaP. The data set includes more than 17 million short cDNA reads obtained for both the treated and untreated cell line and sequenced on Illumina's 1G genome analyzer.

For each sample we were provided with the raw 35 bp RNA-seq reads from the authors. For the untreated prostate cancer cells (LNCaP cell line) there were 4 lanes totaling 10 million, 35 bp reads. For the treated cells there were 3 lanes totaling 7 million, 35 bp reads. All replicates were technical replicates. Reads were mapped to NCBI version 36.3 of the human genome using bowtie. Any read with which mapped to multiple locations was discarded. Using the ENSEMBL 54 annotation from biomart, each mapped read was associated with an ENSEMBL gene. This was done by associating any read that overlapped with any part of the gene (not just the exons) with that gene. Reads that did not correspond to genes were discarded.

6.2 Source of the data

The data set used in this case study is taken from [Li et al., 2008] and was made available from the authors upon request.

6.3 Determining the DE genes using `edgeR`

To begin with, we load in the text data and convert it the appropriate `edgeR` DGEList object.

```
> library(edgeR)
> table.summary = read.table(system.file("extdata", "Li_sum.txt",
+     package = "goseq"), sep = "\t", header = TRUE, stringsAsFactors = FALSE)
> counts = table.summary[, -1]
> rownames(counts) = table.summary[, 1]
> grp = factor(rep(c("Control", "Treated"), times = c(4, 3)))
> summarized = DGEList(counts, lib.size = colSums(counts), group = grp)
```

Next, we use `edgeR` to estimate the biological dispersion and calculate differential expression using a negative binomial model.

```
> disp = estimateCommonDisp(summarized)
> disp$common.dispersion

[1] 0.05681933

> tested = exactTest(disp)

Comparison of groups:  Treated - Control

> topTags(tested)
```

```
Comparison of groups: Treated-Control
                  logConc     logFC P.Value FDR
ENSG00000132437 -14.14850 -5.896416       0   0
ENSG00000151503 -12.96052  5.404687       0   0
ENSG00000096060 -11.77715  4.899691       0   0
ENSG00000166451 -12.61713  4.567569       0   0
ENSG00000116285 -13.01524  4.112220       0   0
ENSG00000113594 -12.24687  4.053165       0   0
ENSG00000123983 -12.06487  3.715681       0   0
ENSG00000162772 -10.91685  3.325138       0   0
ENSG00000075790 -12.21819  3.171288       0   0
ENSG00000116133 -11.87990  3.037119       0   0
```

Finally, we Format the DE genes into a vector suitable for use with goseq

```
> genes = as.integer(p.adjust(tested$table$p.value[tested$table$logFC !=
+     0], method = "BH") < 0.05)
> names(genes) = row.names(tested$table[tested$table$logFC != 0, ])
> table(genes)

genes
    0     1
17916  4827
```

6.4 Determining Genome and Gene ID

In order to allow for automatic data retrieval, the user has to tell goseq what genome and gene ID format were used to summarize the data. In our case we used build 36.1 of the NCBI human genome, we check what code this corresponds to by running:

```
> head(supportedGenomes())

       db organism      date                                name
1    hg19    Human Feb. 2009 Genome Reference Consortium GRCh37
2    hg18    Human Mar. 2006                      NCBI Build 36.1
3    hg17    Human  May 2004                        NCBI Build 35
4    hg16    Human Jul. 2003                        NCBI Build 34
5 felCat4      Cat Dec. 2008                     NHGRI catChrV17e
6 felCat3      Cat Mar. 2006           Broad Institute Release 3

1                                                    ccdsGene,ensGene,exoniphy,geneSym
2   acembly,acescan,ccdsGene,ensGene,exoniphy,geneSymbol,geneid,genscan,knownGene,knownG
3 acembly,acescan,ccdsGene,ensGene,exoniphy,geneSymbol,geneid,genscan,knownGene,refGene,
```

```
4                                                                    acembly,ensGene,exoniphy,geneSymb
5
6                                                                        ensGene,geneSymbol,geneid,ge
```

Which lists the genome codes in the far left column, headed "db". As we are using NCBI build 36.1, we see that we should use "hg18". We also know that we used ENSEMBL Gene ID to summarize our read data, we check what code this corresponds to by running:

```
> head(supportedGeneIDs(), n = 12)
```

```
                       db          track           subtrack               GeneID
1               knownGene     UCSC Genes               <NA>       Entrez Gene ID
2           knownGeneOld3 Old UCSC Genes               <NA>
3  wgEncodeGencodeManualV3  Gencode Genes    Genecode Manual HAVANA Pseudogene ID
4    wgEncodeGencodeAutoV3  Gencode Genes      Genecode Auto HAVANA Pseudogene ID
5   wgEncodeGencodePolyaV3  Gencode Genes     Genecode PolyA HAVANA Pseudogene ID
6                ccdsGene           CCDS               <NA>
7                 refGene   RefSeq Genes               <NA>       Entrez Gene ID
8              xenoRefGene   Other RefSeq               <NA>
9                 vegaGene     Vega Genes Vega Protein Genes HAVANA Pseudogene ID
10          vegaPseudoGene     Vega Genes   Vega Pseudogenes HAVANA Pseudogene ID
11                ensGene  Ensembl Genes               <NA>      Ensembl gene ID
12                acembly  AceView Genes               <NA>
```

```
1
2
3
4
5
6
7
8  anoCar1,aplCal1,braFlo1,caeJap1,caePb1,caePb2,caeRem2,caeRem3,calJac1,canFam1,canFam2
9
10
11
12
```

Note that the head command is only used here to save on space in this guide. Again, the gene ID codes are listed in the far left "db" column. We find that our gene ID code is "ensGene". We will use these strings whenever we are asked for a genome or id.

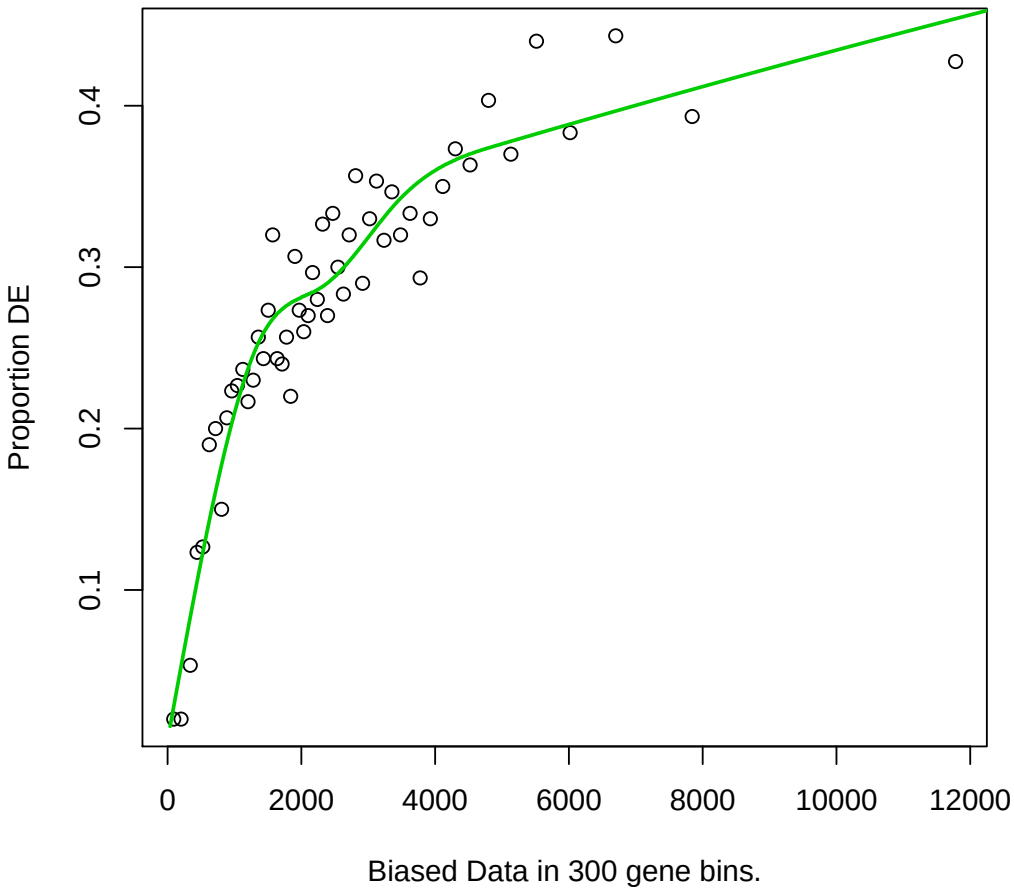
6.5 GO analysis

6.5.1 Fitting the Probability Weighting Function (PWF)

We first need to obtain a weighting for each gene, depending on its length, given by the PWF. As you may have noticed when running supportedGenomes or supportedGeneIDs, length data is available in the local database for our gene ID, "ensGene" and our genome, "hg18". We will let `goseq` automatically fetch this data from its databases.

```
> pwf = nullp(genes, "hg18", "ensGene")
> head(pwf)
```

```
                DEgenes bias.data       pwf
ENSG00000230758       0        NA        NA
ENSG00000182463       0      5455 0.3818773
ENSG00000124208       1      2135 0.2837877
ENSG00000230753       0        NA        NA
ENSG00000224628       0        NA        NA
ENSG00000125835       1       954 0.2023507
```



`nullp` plots the resulting fit, allowing verification of the goodness of fit before continuing the analysis. Further plotting of the pwf can be performed using the `plotPWF` function.

The output of nullp contains all the data used to create the PWF, as well as the PWF itself. It is a data frame with 3 columns, named "DEgenes", "bias.data" and "pwf" with the rownames set to the gene names. Each row corresponds to a gene with the DEgenes column specifying if the gene is DE (1 for DE, 0 for not DE), the bias.data column giving the numeric value of the DE bias being accounted for (usually the gene length or number of counts) and the pwf column giving the genes value on the probability weighting function.

6.5.2 Using the Wallenius approximation

To start with we will use the default method, to calculate the over and under expressed GO categories among DE genes. Again, we allow `goseq` to fetch data automatically, except this time

the data being fetched is the relationship between ENSEMBL gene IDs and GO categories.

```
> GO.wall = goseq(pwf, "hg18", "ensGene")
> head(GO.wall)
```

```
       category over_represented_pvalue under_represented_pvalue
1122 GO:0003674            0.000000e+00                         1
2110 GO:0005575            0.000000e+00                         1
2138 GO:0005622            0.000000e+00                         1
2139 GO:0005623            0.000000e+00                         1
9194 GO:0044464            0.000000e+00                         1
9161 GO:0044424           2.277643e-321                         1
```

The resulting object is ordered by GO category over representation amongst DE genes.

6.5.3 Using random sampling

It may sometimes be desirable to use random sampling to generate the null distribution for category membership. This is easily accomplished by using the `method` option to specify sampling and the `repcnt` option to specify the number of samples to generate:

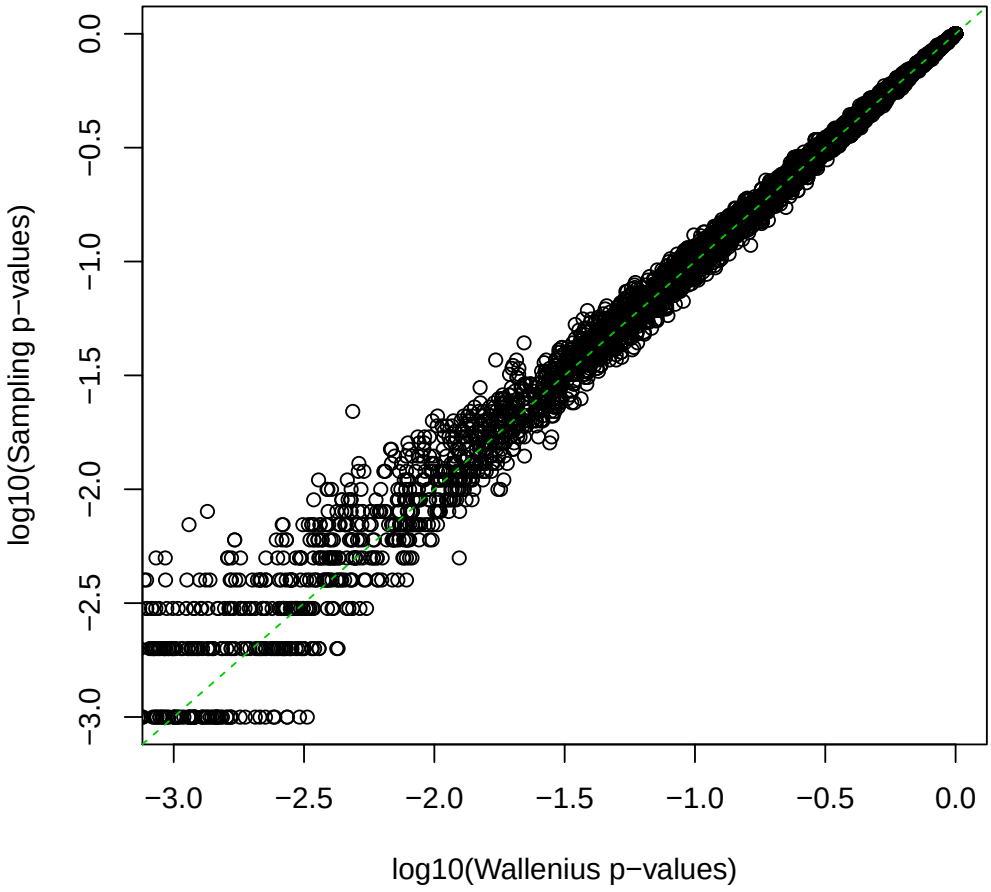
```
> GO.samp = goseq(pwf, "hg18", "ensGene", method = "Sampling", repcnt = 1000)
```

```
> head(GO.samp)
```

```
     category over_represented_pvalue under_represented_pvalue
2  GO:0000003             0.000999001                         1
36 GO:0000075             0.000999001                         1
38 GO:0000077             0.000999001                         1
41 GO:0000082             0.000999001                         1
45 GO:0000086             0.000999001                         1
46 GO:0000087             0.000999001                         1
```

You will notice that this takes far longer than the Wallenius approximation. Plotting the p-values against one another, we see that there is little difference between the two methods.

```
> plot(log10(GO.wall[, 2]), log10(GO.samp[match(GO.samp[, 1], GO.wall[,
+     1]), 2]), xlab = "log10(Wallenius p-values)", ylab = "log10(Sampling p-values)",
+     xlim = c(-3, 0))
> abline(0, 1, col = 3, lty = 2)
```



6.5.4 Ignoring length bias

`goseq` also allows for one to perform a GO analysis without correcting for RNA-seq length bias. In practice, this is only useful for assessing the effect of length bias on your results. You should NEVER use this option as your final analysis. If length bias is truly not present in your data, `goseq` will produce a nearly flat PWF and no length bias correction will be applied to your data and all methods will produce the same results.

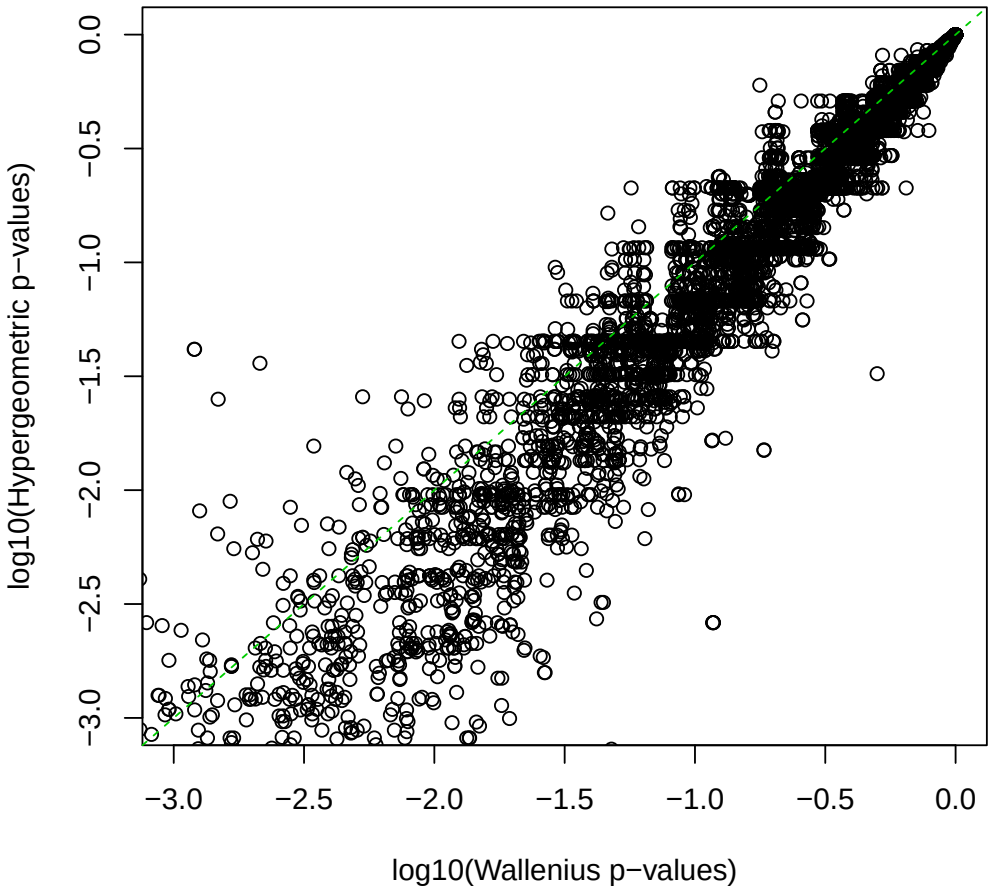
However, if you still wish to ignore length bias in calculating GO category enrichment, this is again accomplished using the `method` option.

```
> GO.nobias = goseq(pwf, "hg18", "ensGene", method = "Hypergeometric")
> head(GO.nobias)
```

```
       category over_represented_pvalue under_represented_pvalue
1122 GO:0003674                       0                        1
2074 GO:0005488                       0                        1
2110 GO:0005575                       0                        1
2138 GO:0005622                       0                        1
2139 GO:0005623                       0                        1
3340 GO:0008150                       0                        1
```

Ignoring length bias gives very different results from a length bias corrected analysis.

```
> plot(log10(GO.wall[, 2]), log10(GO.nobias[match(GO.nobias[, 1],
+     GO.wall[, 1]), 2]), xlab = "log10(Wallenius p-values)", ylab = "log10(Hypergeome
+     xlim = c(-3, 0), ylim = c(-3, 0))
> abline(0, 1, col = 3, lty = 2)
```



6.5.5 Limiting GO categories and other category based tests

By default, goseq tests all three major Gene Ontology branches; Cellular Components, Biological Processes and Molecular Functions. However, it is possible to limit testing to any combination of the major branches by using the test.cats argument to the goseq function. This is done by specifying a vector consisting of some combination of the strings "GO:CC", "GO:BP" and "GO:MF". For example, to test for only Molecular Function GO categories:

```
> GO.MF = goseq(pwf, "hg18", "ensGene", test.cats = c("GO:MF"))
> head(GO.MF)
```

```
       category over_represented_pvalue under_represented_pvalue
156  GO:0003674            0.000000e+00                         1
1108 GO:0005488           1.069226e-265                         1
1118 GO:0005515           8.164167e-198                         1
208  GO:0003824            4.972261e-85                         1
19   GO:0000166            1.390289e-57                         1
157  GO:0003676            3.070543e-47                         1
```

Native support for other category tests, such as KEGG pathway analysis are also made available via this argument. See the man goseq function man page for up to date information on what category tests are natively supported.

6.5.6 Making sense of the results

Having performed the GO analysis, you may now wish to interpret the results. If you wish to identify categories significantly enriched/unenriched below some p-value cutoff, it is necessary to first apply some kind of multiple hypothesis testing correction. For example, GO categories over enriched using a .05 FDR cutoff [Benjamini and Hochberg, 1995] are:

```
> enriched.GO = GO.wall$category[p.adjust(GO.wall$over_represented_pvalue,
+     method = "BH") < 0.05]
> head(enriched.GO)
```

```
[1] "GO:0003674" "GO:0005575" "GO:0005622" "GO:0005623" "GO:0044464" "GO:0044424"
```

Unless you are a machine, GO accession identifiers are probably not very meaningful to you. Information about each term can be obtained from the Gene Ontology website, http://www.geneontology.org/, or using the R package GO.db.

```
> library(GO.db)
> for (go in enriched.GO[1:10]) {
+     print(GOTERM[[go]])
+     cat("--------------------------------------\n")
+ }
```

```
GOID: GO:0003674
Term: molecular_function
Ontology: MF
Definition: Elemental activities, such as catalysis or binding, describing
    the actions of a gene product at the molecular level. A given gene
    product may exhibit one or more molecular functions.
Synonym: molecular function
Synonym: molecular function unknown
Synonym: GO:0005554
Secondary: GO:0005554
---------------------------------------
GOID: GO:0005575
Term: cellular_component
Ontology: CC
Definition: The part of a cell or its extracellular environment in which a
    gene product is located. A gene product may be located in one or more
    parts of a cell and its location may be as specific as a particular
    macromolecular complex, that is, a stable, persistent association of
    macromolecules that function together.
Synonym: cellular component
Synonym: cellular component unknown
Synonym: GO:0008372
Secondary: GO:0008372
---------------------------------------
GOID: GO:0005622
Term: intracellular
Ontology: CC
Definition: The living contents of a cell; the matter contained within
    (but not including) the plasma membrane, usually taken to exclude
    large vacuoles and masses of secretory or ingested material. In
    eukaryotes it includes the nucleus and cytoplasm.
Synonym: internal to cell
Synonym: nucleocytoplasm
Synonym: protoplasm
Synonym: protoplast
---------------------------------------
GOID: GO:0005623
Term: cell
Ontology: CC
Definition: The basic structural and functional unit of all organisms.
    Includes the plasma membrane and any external encapsulating structures
    such as the cell wall and cell envelope.
```

```
--------------------------------------
GOID: GO:0044464
Term: cell part
Ontology: CC
Definition: Any constituent part of a cell, the basic structural and
    functional unit of all organisms.
Synonym: protoplast
--------------------------------------
GOID: GO:0044424
Term: intracellular part
Ontology: CC
Definition: Any constituent part of the living contents of a cell; the
    matter contained within (but not including) the plasma membrane,
    usually taken to exclude large vacuoles and masses of secretory or
    ingested material. In eukaryotes it includes the nucleus and
    cytoplasm.
--------------------------------------
GOID: GO:0008150
Term: biological_process
Ontology: BP
Definition: Any process specifically pertinent to the functioning of
    integrated living units: cells, tissues, organs, and organisms. A
    process is a collection of molecular events with a defined beginning
    and end.
Synonym: biological process
Synonym: biological process unknown
Synonym: physiological process
Synonym: GO:0000004
Synonym: GO:0007582
Secondary: GO:0000004
Secondary: GO:0007582
--------------------------------------
GOID: GO:0009987
Term: cellular process
Ontology: BP
Definition: Any process that is carried out at the cellular level, but not
    necessarily restricted to a single cell. For example, cell
    communication occurs among more than one cell, but occurs at the
    cellular level.
Synonym: cell growth and/or maintenance
Synonym: cell physiology
Synonym: cellular physiological process
```

```
Synonym: GO:0008151
Synonym: GO:0050875
Secondary: GO:0008151
Secondary: GO:0050875
--------------------------------------
GOID: GO:0005488
Term: binding
Ontology: MF
Definition: The selective, non-covalent, often stoichiometric, interaction
    of a molecule with one or more specific sites on another molecule.
Synonym: ligand
--------------------------------------
GOID: GO:0043229
Term: intracellular organelle
Ontology: CC
Definition: Organized structure of distinctive morphology and function,
    occurring within the cell. Includes the nucleus, mitochondria,
    plastids, vacuoles, vesicles, ribosomes and the cytoskeleton. Excludes
    the plasma membrane.
--------------------------------------
```

6.5.7 Understanding goseq internals

The situation may arise where it is necessary for the user to perform some of the data processing steps usually performed automatically by `goseq` themselves. With this in mind, it will be useful to step through the preprocessing steps performed automatically by `goseq` to understand what is happening.

To start with, when `nullp` is called, `goseq` uses the genome and gene identifiers supplied to try and retrieve length information for all genes given to the `genes` argument. To do this, it retrieves the data from the database of gene lengths maintained in the package `geneLenDataBase`. This is performed by the `getlength` function in the following way:

```
> len = getlength(names(genes), "hg18", "ensGene")
> length(len)

[1] 22743

> length(genes)

[1] 22743

> head(len)
```

```
[1]   NA 5455 2135   NA   NA  954
```

After some data cleanup, the length data and the DE data is then passed to the `makespline` function to produce the PWF. The `nullp` returns a data frame which has 3 columns, the original DEgenes vector, the length bias data (in a column called bias.data) and the PWF itself (in a column named pwf). The names of the genes are also kept in this data frame as the names of the rows. If length data could not be obtained for a certain gene the corresponding entries in the "bias.data" and "pwf" columns are set to NA.

Next we call the `goseq` function to determine over/under representation of GO categories amongst DE genes. When we do this, `goseq` looks for the appropriate organism package and tries to obtain the mapping from genes to GO categories from it. This is done using the `getgo` function as follows:

```
> go = getgo(names(genes), "hg18", "ensGene")
> length(go)

[1] 22743

> length(genes)

[1] 22743

> head(go)

$<NA>
NULL

$ENSG00000182463
 [1] "GO:0006139" "GO:0006350" "GO:0006351" "GO:0006355" "GO:0006807" "GO:0007275"
 [7] "GO:0008150" "GO:0008152" "GO:0009058" "GO:0009059" "GO:0009889" "GO:0009987"
[13] "GO:0010467" "GO:0010468" "GO:0010556" "GO:0016070" "GO:0019219" "GO:0019222"
[19] "GO:0031323" "GO:0031326" "GO:0032501" "GO:0032502" "GO:0032774" "GO:0034641"
[25] "GO:0034645" "GO:0043170" "GO:0044237" "GO:0044238" "GO:0044249" "GO:0044260"
[31] "GO:0045449" "GO:0050789" "GO:0050794" "GO:0051171" "GO:0051252" "GO:0060255"
[37] "GO:0065007" "GO:0080090" "GO:0090304" "GO:2000112" "GO:0005575" "GO:0005622"
[43] "GO:0005623" "GO:0005634" "GO:0043226" "GO:0043227" "GO:0043229" "GO:0043231"
[49] "GO:0044424" "GO:0044464" "GO:0001071" "GO:0003674" "GO:0003676" "GO:0003677"
[55] "GO:0003700" "GO:0005488" "GO:0008270" "GO:0043167" "GO:0043169" "GO:0043565"
[61] "GO:0046872" "GO:0046914"

$<NA>
NULL
```

```
$<NA>
NULL

$<NA>
NULL

$ENSG00000125835
 [1] "GO:0000375" "GO:0000377" "GO:0000387" "GO:0000398" "GO:0006139" "GO:0006350"
 [7] "GO:0006351" "GO:0006353" "GO:0006366" "GO:0006369" "GO:0006396" "GO:0006397"
[13] "GO:0006807" "GO:0008150" "GO:0008152" "GO:0008334" "GO:0008380" "GO:0009058"
[19] "GO:0009059" "GO:0009987" "GO:0010467" "GO:0016043" "GO:0016070" "GO:0016071"
[25] "GO:0022607" "GO:0022613" "GO:0022618" "GO:0032774" "GO:0034621" "GO:0034622"
[31] "GO:0034641" "GO:0034645" "GO:0034660" "GO:0043170" "GO:0043933" "GO:0044085"
[37] "GO:0044237" "GO:0044238" "GO:0044249" "GO:0044260" "GO:0065003" "GO:0071826"
[43] "GO:0071840" "GO:0071841" "GO:0071842" "GO:0071843" "GO:0071844" "GO:0090304"
[49] "GO:0005575" "GO:0005622" "GO:0005623" "GO:0005634" "GO:0005654" "GO:0005681"
[55] "GO:0005683" "GO:0005689" "GO:0005737" "GO:0005829" "GO:0030529" "GO:0030532"
[61] "GO:0031974" "GO:0031981" "GO:0032991" "GO:0043226" "GO:0043227" "GO:0043229"
[67] "GO:0043231" "GO:0043233" "GO:0044422" "GO:0044424" "GO:0044428" "GO:0044444"
[73] "GO:0044446" "GO:0044464" "GO:0070013" "GO:0071013" "GO:0003674" "GO:0003676"
[79] "GO:0003723" "GO:0005488" "GO:0005515"
```

Note that the first thing the getgo function does is to convert the UCSC genome/ID namings into the naming convention used by the organism packages. This is done using two hard coded conversion vectors that are included in the `goseq` package but usually hiden from the user.

```
> goseq:::.ID_MAP

 knownGene    refGene    ensGene geneSymbol
      "eg"       "eg"  "ENSEMBL"   "SYMBOL"

> goseq:::.ORG_PACKAGES

          anoGam      Arabidopsis           bosTau               ce
     "org.Ag.eg"    "org.At.tair"      "org.Bt.eg"      "org.Ce.eg"
          canFam               dm           danRer      E. coli K12
     "org.Cf.eg"      "org.Dm.eg"      "org.Dr.eg"   "org.EcK12.eg"
   E. coli Sakai           galGal               hg               mm
"org.EcSakai.eg"      "org.Gg.eg"      "org.Hs.eg"      "org.Mm.eg"
          rheMac          Malaria           panTro               rn
    "org.Mmu.eg"  "org.Pf.plasmo"      "org.Pt.eg"      "org.Rn.eg"
          sacCer              Pig           xenTro
    "org.Sc.sgd"      "org.Ss.eg"      "org.Xl.eg"
```

It is just as valid to run the length and GO category fetching as separate steps and then pass the result to the `nullp` and `goseq` functions using the `bias.data` and `gene2cat` arguments. Thus the following two blocks of code are equivalent:

```
> pwf = nullp(genes, "hg18", "ensGene")
> go = goseq(pwf, "hg18", "ensGene")
```

and

```
> gene_lengths = getlength(names(genes), "hg18", "ensGene")
> pwf = nullp(genes, bias.data = gene_lengths)
> go_map = getgo(names(genes), "hg18", "ensGene")
> go = goseq(pwf, "hg18", "ensGene", gene2cat = go_map)
```

6.6 KEGG pathway analysis

In order to illustrate performing a category test not present in the `goseq` database, we perform a KEGG pathway analysis. For human, the mapping from KEGG pathways to genes are stored in the package org.Hs.eg.db, in the object org.Hs.egPATH. In order to test for KEGG pathway over representation amongst DE genes, we need to extract this information and put it in a format that `goseq` understands. Unfortunately, the org.Hs.eg.db package does not contain direct mappings between ENSEMBL gene ID and KEGG pathway. Therefore, we have to construct this map by combining the ENSEMBL <-> Entrez and Entrez <-> KEGG mappings. This can be done using the following code:

```
> en2eg = as.list(org.Hs.egENSEMBL2EG)
> eg2kegg = as.list(org.Hs.egPATH)
> grepKEGG = function(id, mapkeys) {
+     unique(unlist(mapkeys[id], use.names = FALSE))
+ }
> kegg = lapply(en2eg, grepKEGG, eg2kegg)
> head(kegg)
```

Note that this step is quite time consuming. The code written here is not the most efficient way of producing this result, but the logic is much clearer than faster algorithms. The source code for getgo contains a more efficient routine.

We produce the PWF as before. Then, to perform a KEGG analysis, we simply make use of the `gene2cat` option in `goseq`:

```
> pwf = nullp(genes, "hg18", "ensGene")
> KEGG = goseq(pwf, gene2cat = kegg)
> head(KEGG)
```

Note that we do not have to tell the goseq function what organism and gene ID we are using as we are manually supplying the mapping between genes and categories.

KEGG analysis is shown as an illustration of how to supply your own mapping between gene ID and category, KEGG analysis is actually natively supported by GOseq and we could have performed it with the following code.

```
> pwf = nullp(genes, "hg18", "ensGene")
> KEGG = goseq(pwf, "hg18", "ensGene", test.cats = "KEGG")
> head(KEGG)

    category over_represented_pvalue under_represented_pvalue
85     01100            1.791722e-26                        1
88     03010            6.978547e-25                        1
186    05012            3.017921e-14                        1
118    04141            3.574455e-13                        1
89     03013            5.049368e-13                        1
185    05010            1.561099e-12                        1
```

Noting that this time it was necessary to tell the goseq function that we are using HG18 and ENSEMBL gene ID, as the function needs this information to automatically construct the mapping from geneid to KEGG pathway.

6.7 Extracting mappings from organism packages

If you know that the information mapping gene ID to your categories of interest is contained in the organism packages, but `goseq` fails to fetch it automatically, you may want to extract it yourself and then pass it to the `goseq` function using the `gene2cat` argument. This is done in exactly the same way as extracting the KEGG to ENSEMBL mappings in the section "KEGG pathway analysis" above. This example is actually the worst case, where it is necessary to combine two mappings to get the desired list. If we had instead wanted the association between Entrez gene IDs and KEGG pathways, the following code would have been sufficient:

```
> kegg = as.list(org.Hs.egPATH)
> head(kegg)

$`1`
[1] NA

$`10`
[1] "00232" "00983" "01100"

$`100`
```

```
[1] "00230" "01100" "05340"

$`1000`
[1] "04514" "05412"

$`10000`
 [1] "04010" "04012" "04062" "04150" "04210" "04370" "04380" "04510" "04530" "04620"
[11] "04630" "04660" "04662" "04664" "04666" "04722" "04910" "04914" "04920" "04973"
[21] "05142" "05145" "05160" "05200" "05210" "05211" "05212" "05213" "05214" "05215"
[31] "05218" "05220" "05221" "05222" "05223"

$`100008586`
[1] NA
```

A note on fetching GO mappings from the organism. The data structure of GO is a directed acyclic graph. This means that in addition to each GO category being associated with a set of genes, it may also have children that are associated to other genes. It is important to use the org.Hs.egGO2ALLEGS and NOT the org.Hs.egGO object to create the mapping between GO categories and gene identifiers, as the latter does not include the links to genes arising from "child" GO categories. Thank you to Christopher Fjell for pointing this out.

6.8 Correcting for other biases

It is possible that in some circumstances you will wish to correct not just for length bias, but for the total number of counts. This can make sense because power to detect DE depends on the total number of counts a gene receives, which is the product of gene length and gene expression. So correcting for read count bias will compensate for all biases, known and unknown, in power to detect DE. On the other hand, it will also remove bias resulting from differences in expression level, which may not be desirable.

Correcting for count bias will produce a different PWF. Therefore, we need to tell `goseq` about the data on which the fraction DE depends when calculating the PWF using the `nullp` function. We then simply pass the result to `goseq` as usual.

So, in order to tell `goseq` to correct for read count bias instead of length bias, all you need to do is supply a numeric vector, containing the number of counts for each gene to `nullp`.

```
> countbias = rowSums(counts)[rowSums(counts) != 0]
> length(countbias)

[1] 22743

> length(genes)
```

```
[1] 22743
```

To use the count bias when doing GO analysis, simply pass this vector to `nullp` using the `bias.data` option. Note that we have to supply "hg18" and "ensGene" to `goseq` as it is not used by nullp and hence not in the pwf.counts object.

```
> pwf.counts = nullp(genes, bias.data = countbias)
> GO.counts = goseq(pwf.counts, "hg18", "ensGene")
> head(GO.counts)

       category over_represented_pvalue under_represented_pvalue
2110 GO:0005575           1.176220e-41                         1
3340 GO:0008150           3.832750e-39                         1
2139 GO:0005623           7.379123e-36                         1
9194 GO:0044464           7.379123e-36                         1
1122 GO:0003674           2.893818e-34                         1
4029 GO:0009987           3.074095e-33                         1
```

Note that if you want to correct for length bias, but your organism/gene identifier is not natively supported, then you need to follow the same procedure as above, only the numeric vector supplied will contain each gene's length instead of its number of reads.

7 Setup

This vignette was built on:

```
> sessionInfo()

R version 2.13.0 (2011-04-13)
Platform: i386-pc-mingw32/i386 (32-bit)

locale:
[1] LC_COLLATE=C                           LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252 LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] GO.db_2.5.0            org.Hs.eg.db_2.5.0     RSQLite_0.9-4
 [4] DBI_0.2-5              AnnotationDbi_1.14.0   Biobase_2.12.0
```

```
 [7] edgeR_2.2.1            goseq_1.4.0            geneLenDataBase_0.99.7
[10] BiasedUrn_1.03

loaded via a namespace (and not attached):
 [1] BSgenome_1.20.0       Biostrings_2.20.0     GenomicFeatures_1.4.0
 [4] GenomicRanges_1.4.0   IRanges_1.10.0        Matrix_0.999375-50
 [7] RCurl_1.5-0.1         XML_3.2-0.2           biomaRt_2.8.0
[10] grid_2.13.0           lattice_0.19-23       limma_3.8.0
[13] mgcv_1.7-5            nlme_3.1-100          rtracklayer_1.12.0
[16] tools_2.13.0
```

8 Acknowledgments

Christopher Fjell for a series of bug fixes and pointing out the difference between the egGO and egGO2ALLEGS objects in the organism packages.

References

Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B*, 57:289–300, 1995.

Hairi Li, Michael T Lovci, Young-Soo Kwon, Michael G Rosenfeld, Xiang-Dong Fu, and Gene W Yeo. Determination of tag density required for digital transcriptome analysis: application to an androgen-sensitive prostate cancer model. *Proc Natl Acad Sci USA*, 105(51):20179–84, Dec 2008. doi: 10.1073/pnas.0807121105.

Alicia Oshlack and Matthew J Wakefield. Transcript length bias in RNA-seq data confounds systems biology. *Biol Direct*, 4:14, Jan 2009. doi: 10.1186/1745-6150-4-14.

M. D. Robinson and G. K. Smyth. Moderated statistical tests for assessing differences in tag abundance. *Bioinformatics*, 23(21):2881–2887, 2007.

M. D. Robinson and G. K. Smyth. Small-sample estimation of negative binomial dispersion, with applications to sage data. *Biostatistics*, 9(2):321–332, 2008.

M. D. Robinson, D. J. McCarthy, and G. K. Smyth. edgeR: a bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, 26(1):139–40, Jan 2010. doi: 10.1093/bioinformatics/btp616.

M. D. Young, M. J. Wakefield, G. K. Smyth, and A. Oshlack. Gene ontology analysis for RNA-seq: accounting for selection bias. *Genome Biology*, 11:R14, Feb 2010.