R News

The Newsletter of the R Project Volume 6/5, December 2006

Editorial

by Wolfgang Huber and Paul Murrell

Welcome to the fifth and final issue of R News for 2006, our third special issue of the year, with a focus on the use of R in Bioinformatics. Many thanks to guest editor Wolfgang Huber for doing a fantastic job in putting this issue together.

Paul Murrell
The University of Auckland, New Zealand
paul.murrell@R-project.org

Biology is going through a revolution. The genome sequencing projects that were initiated in the 1980's and undertaken in the 1990's have provided for the first time systematic inventories of the components of biological systems. Technological innovation is producing ever more detailed measurements on the functioning of these components and their interactions. The Internet has opened possibilities for the sharing of data and of computational resources that would have been unimaginable only 15 years ago.

The field of bioinformatics emerged in the 1990's to deal with the pressing questions that the massive amounts of new sequence data posed: sequence alignment, similarity and clustering, their phylogenetic interpretation, genome assembly, sequence annotation, protein structure. C and Perl were the languages of choice for the first generation of bioinformaticians. All of these question remain relevant, yet in addition we now have the big and colourful field of functional genomics, which employs all sorts of technologies to measure the abundances and activities of biomolecules under different conditions, map their interactions, monitor the effect of their perturbation on the phenotype of a cell or even a whole organism, and eventually to build predictive models of biological systems.

R is well suited to many of the scientific and computational challenges in functional genomics. Some of the efforts in this field have been pulled together since 2001 by the *Bioconductor* project, and many of the papers in this issue report on packages from the project. But Bioconductor is more than just a CRAN-style repository of biology-related packages. Motivated by the particular challenges of genomic data, the project has actively driven a number of technological innovations that have flown back into R, among these, package vignettes, an embracement of S4, the management of extensive package dependence hierarchies, and interfaces between R and

Contents of this issue:

other software systems. Biological metadata (for example, genome annotations) need to be tightly integrated with the analysis of primary data, and the Bioconductor project has invested a lot of effort in the provision of high quality metadata packages. The experimental data in functional genomics require more structured formats than the basic data types of R, and one of the main products of the Bioconductor core is the provision of common data structures that allow the efficient exchange of data and computational results between different packages. One example is the `ExpressionSet`, an S4 class for the storage of the essential data and information on a microarray experiment.

The articles in this issue span a wide range of topics. Common themes are *preprocessing* (data import, quality assessment, standardization, error modeling and summarization), *pattern discovery and detection*, and higher level statistical models with which we aim to gain insight into the underlying biological processes. Sometimes, the questions that we encounter in bioinformatics result in methods that have potentially a wider applicability; this is true in particular for the first three articles with which we start this issue.

Wolfgang Huber
European Bioinformatics Insitute (EBI)
European Molecular Biology Laboratory (EMBL) Cambridge, UK
huber@ebi.ac.uk

Graphs and Networks: Tools in Bioconductor

by Li Long and Vince Carey

Introduction

Network structures are fundamental components for scientific modeling in a number of substantive domains, including sociology, ecology, and computational biology. The mathematical theory of graphs comes immediately into play when the entities and processes being modeled are clearly organized into objects (modeled as graph nodes) and relationships (modeled as graph edges).

Graph theory addresses the taxonomy of graph structures, the measurement of general features of connectedness, complexity of traversals, and many other combinatorial and algebraic concepts. An important generalization of the basic concept of graph (traditionally defined as a set of nodes N and a binary relation E on N defining edges) is the hypergraph (in which edges are general subsets of the node set of cardinality at least 2).

The basic architecture of the Bioconductor toolkit for graphs and networks has the following structure:

- Representation infrastructure: packages **graph**, **hypergraph**;
- Algorithms for traversal and measurement: packages **RBGL**, **graphPart**
- Algorithms for layout and visualization: package **Rgraphviz**; **RBGL** also includes some layout algorithms;
- Packages for addressing substantive problems in bioinformatics: packages **pathRender**, **GraphAT**, **ScISI**, **GOstats**, **ontoTools**, and others.

In this article we survey aspects of working with network structures with some of these Bioconductor tools.

The basics: package graph

The **graph** package provides a variety of S4 classes representing graphs. A virtual class `graph` defines the basic structure:

```
> library(graph)
> getClass("graph")
Virtual Class

Slots:

Name:   edgemode  edgeData  nodeData
Class: character  attrData  attrData

Known Subclasses: "graphNEL", "graphAM",
   "graphH", "distGraph", "clusterGraph",
   "generalGraph"
```

A widely used concrete extension of this class is `graphNEL`, denoting the "node and edge list" representation:

```
> getClass("graphNEL")

Slots:

Name:      nodes     edgeL  edgemode  edgeData
Class:    vector      list character  attrData
```

```
            attrData
            nodeData

Extends: "graph"
```

An example graph is supplied, representing the "integrin mediated cell adhesion" pathway as defined in a previous version of KEGG (Kyoto Encyclopedia of Genes and Genomes, Kanehisa and Goto, 2000; the pathway is now called "focal adhesion").

```
> data(integrinMediatedCellAdhesion)
> IMCAGraph
A graphNEL graph with directed edges
Number of Nodes = 52
Number of Edges = 91
> nodes(IMCAGraph)[1:5]
[1] "ITGB" "ITGA" "ILK"  "CAV"  "SHC"
> edges(IMCAGraph)[1:5]
$ITGB
[1] "ITGA" "ILK"  "CAV"  "SHC"  "ACTN" "TLN"

$ITGA
[1] "ITGB"

$ILK
[1] "ITGB"

$CAV
[1] "ITGB"

$SHC
[1] "FYN"  "GRB2" "ITGB"
```

Methods that can be used to work with graph instances include:

- The `edgemode` method returns a character token indicating whether the graph is directed or undirected.
- The `nodes` method returns information on graph nodes; in the case of a `graphNEL` instance it returns a vector, intended to represent the (unordered) node set through character names of entities described by the graph. (Note that there are more general representations available, in which objects other than strings can constitute graph nodes, see remarks on `nodeData` below.)
- The `edges` method returns a named list. The *i*th element of the list bears as its name the name of the *i*th node, and contains a vector of node names to which the *i*th node is directly linked by a graph edge.

```
> all.equal(names(edges(IMCAGraph)),
     nodes(IMCAGraph))
[1] TRUE
```

The network may be visualized using the following code:

```
library(Rgraphviz)
plot(IMCAGraph,
    attrs = IMCAAttrs$defAttrs,
    nodeAttrs = IMCAAttrs$nodeAttrs,
    subGList = IMCAAttrs$subGList);
```

See Figure 1.

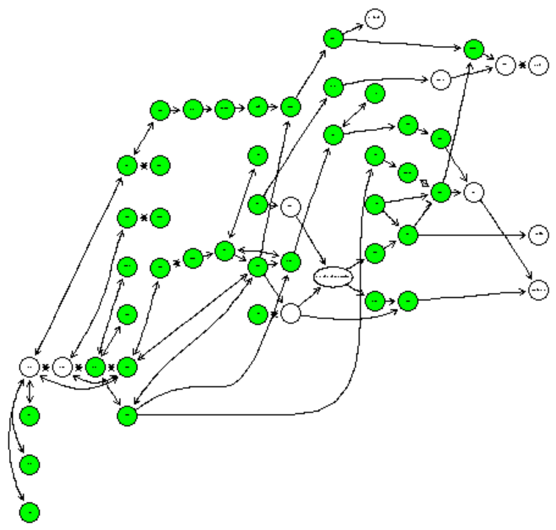


Figure 1: Rendering of the IMCA pathway.

We will illustrate aspects of other parts of the system using this example graph structure.

The Rgraphviz package

Rgraphviz provides an interface to graphviz[1], an open source library for graph visualization. Graphviz performs several types of graph layout:

- dot: draws directed graphs in a hierarchical way
- neato: draws graphs using Kamada-Kawai algorithm
- fdp: draws graphs using Fruchterman-Reingold heuristic
- twopi: draws radial layout
- circo: draws circular layout

Many aspects of graph rendering are covered, with general programmatic control over colors, fonts, shapes, and line styles employed for various aspects of the display and annotation of the graph.

The Bioconductor **Rgraphviz** interface package lets you choose the layout engine for your graph object among the options `dot` (the default), `neato` and `twopi`.

[1]http://www.graphviz.org

We generate a graph and plot it using different layout engines:

```
> library("Rgraphviz")
> set.seed(123)
> V <- letters[1:10]
> M <- 1:4
> g1 <- randomGraph(V, M, 0.2)
A graphNEL graph with undirected edges
Number of Nodes = 10
Number of Edges = 16
```

```
> plot(g1)
```

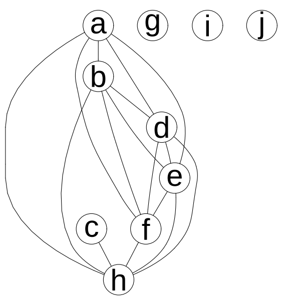


Figure 2: Rendering with dot.

```
> plot(g1, "neato")
```

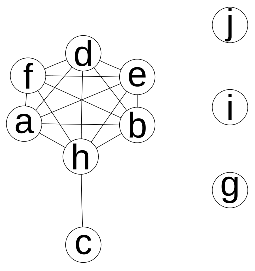


Figure 3: Rendering with neato.

```
> plot(g1, "twopi")
```

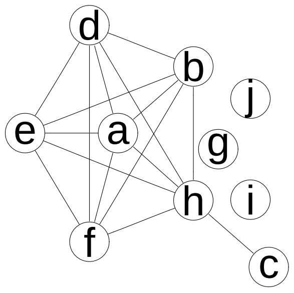


Figure 4: Rendering with twopi.

[2]http://www.boost.org

[3]http://www.boost.org/libs/graph

The RBGL package

Researchers study different kinds of biological networks: protein interaction networks, genetic regulatory systems, metabolic networks are a few examples. Signaling pathway identification could be modeled using shortest paths in a protein interaction network. The task of identification of protein complexes can be conducted by finding cohesive subgroups in protein interaction networks. The protein interaction network under consideration is built from the experimental data, which contains incomplete information, false positives and false negatives. Extensible representations that can manage such complications are at a premium in this process.

RBGL provides a set of graph algorithms to analyse the structures and properties of graphs and networks.

There are three categories of algorithms in this package:

- interfaces to graph algorithms from the boost[2] graph library (BGL[3]);
- algorithms built on BGL; and
- other algorithms.

The Boost C++ Library provides peer-reviewed C++ libraries, with the objective of creating highly portable open source resources for C++ programming analogous to the proprietary standard template library (STL).

The Boost Graph Library (BGL) is one of the Boost libraries. BGL provides a set of generic data structures to represent graphs, and a collection of C++ modules (all defined as templates) constituting algorithms on graphs.

Package **RBGL** provides a direct interface to many of the graph algorithms available in BGL. Table 1 lists the main functionalities currently provided.

In the second category, we implemented minimum-cut algorithm `minCut`, based on one of the maximum-flow algorithms from BGL.

In the third category, we implemented

- highly connected subgraphs (function `highlyConnSG`), a clustering algorithm proposed by Hartuv and Shamir (2000).
- a number of algorithms from social network analysis: k-cliques (function `kCliques`), k-cores (function `kCores`), maximum cliques (function `maxClique`), lambda-sets (function `LambdaSets`), and
- predicate functions: to decide if a graph is triangulated (function `is.triangulated`), to test if a subset of nodes separates two other subsets of nodes (function `separates`).

RBGL functions	Comments
Traversals	
`bfs`	breadth-first search
`dfs`	depth-first search
Shortest paths	
`dijkstra.sp`	Single-source, nonnegative weights
`bellman.ford.sp`	Single-source, general weights
`dag.sp`	Single-source, DAG
`floyd.warshall.-all.pairs.sp`	Returns distance matrix
`johnson.all.pairs.sp`	Returns distance matrix
Minimal spanning trees	
`mstree.kruskal`	Returns edge list and weights
`prim.minST`	As above
Connectivity	
`connectedComp`	Returns list of node-sets
`strongComp`	As above
`articulationPoints`	As above
`biConnComp`	As above
`edgeConnectivity`	Returns index and minimum disconnecting set
`init.incremental.-components`	Special processing for evolving graphs
`incremental.components`	
`same.component`	Boolean in the incremental setting
Maximum flow algorithms	
`edmunds.karp.max.flow`	List of max flow, and edge-specific flows
`push.relabel.max.flow`	
Vertex ordering	
`tsort`	Topological sort
`cuthill.mckee.ordering`	Reduces bandwidth
`sloan.ordering`	Reduces wavefront
`min.degree.ordering`	Heuristic
Other functions	
`transitive.closure`	Returns from-to matrix
`isomorphism`	Boolean
`sequential.vertex.coloring`	Returns color no for nodes
`brandes.betweenness.-centrality`	Indices and dominance measure
`circle.layout`	Returns vertex coordinates
`kamada.kawai.spring.layout`	Returns vertex coordinates

Table 1: Names of key functions in **RBGL**. Working examples for all functions are provided in the package manual pages.

To illustrate an application, we use the sp.between function to determine the shortest path in the network between the SRC gene and the cell proliferation process:

```
> sp.between(IMCAGraph, "SRC",
          "cell proliferation")
$`SRC:cell proliferation`
$`SRC:cell proliferation`$path
[1] "SRC"                "FAK"
[3] "MEK"                "ERK"
[5] "cell proliferation"

$`SRC:cell proliferation`$length
[1] 4

$`SRC:cell proliferation`$pweights
               SRC->FAK
                      1
               FAK->MEK
                      1
               MEK->ERK
                      1
ERK->cell proliferation
                      1
```

The resulting list includes a vector encoding the shortest path, its length, and the weights of individual steps through the graph.

Representations and general attributes

As noted in the class report for getClass("graph"), a variety of graph representations are available as S4 classes. It is particularly simple to work with graphAM on the basis of a binary adjacency matrix:

```
> am = diag(4)
> am = cbind(1,am)
> am = rbind(am,0)
> am[5,3] = am[3,5] = 1
> am[1,1] = 0
> dimnames(am) = list(letters[1:5],
                    letters[1:5])
> am
  a b c d e
a 0 1 0 0 0
b 1 0 1 0 0
c 1 0 0 1 1
d 1 0 0 0 1
e 0 0 1 0 0
> amg = new("graphAM", am,
          edgemode="directed")
> amg
A graphAM graph with directed edges
Number of Nodes = 5
Number of Edges = 9
> nodes(amg)
[1] "a" "b" "c" "d" "e"
> edges(amg)[2]
$b
[1] "a" "c"
```

Other classes can be investigated through examples in the manual pages.

To illustrate extensibility of graph component representations, we return to the IMCAGraph example. As noted above, the node set for a graphNEL instance is a character vector. We may wish to have additional attributes characterizing the nodes. In this example, we show how to allow a "long gene name" attribute to be defined on this instance.

First, we establish the name of the node attribute and give its default value for all nodes.

```
> nodeDataDefaults(IMCAGraph,
      attr="longname") <- as.character(NA)
```

Now we take a specific node and assign the new attribute value.

```
> nodeData(IMCAGraph, "SHC",
      attr="longname" ) =
+ paste("src homology 2 domain",
   "containing transforming protein")
```

To retrieve the attribute value, the nodeData accessor is used:

```
> nodeData(IMCAGraph, "SHC")
$SHC
$SHC$longname
[1] "src homology 2 domain containing
        transforming protein"
```

The hypergraph and graphPart packages

A hypergraph is a generalised graph, where a hyperedge is defined to represent a *subset* of nodes (in contrast to the edge in a basic graph, which is defined by a *pair* of nodes). This structure can be used to model one-to-many and many-to-many relationships. The package **hypergraph** provides an R class to represent hypergraphs.

A protein interaction network could be considered as a hypergraph: nodes are proteins, hyperedges are protein complexes, a node is connected to a hyperedge when the corresponding protein is a member of the corresponding protein complex.

A k-way partition of a hypergraph assigns the nodes to k disjoint non-empty sets so that a given cost function is minimum. A typical cost function is the weight sum of hyperedges that span over more than one such disjoint sets. Computing k-way partitions of hypergraphs is NP-hard.

There are several libraries available that provide approximate solutions to the above partition problem. The package **graphPart** provides interfaces to two such libraries:

- hMETIS[4]: a set of algorithms based on the multilevel hypergraph partitioning schemes developed in Karypis Lab at Univ. of Minnesota. These algorithms have shown to produce good results and are scalable.
- PaToH[5]: a multilevel hypergraph partitioning tool developed by U. Catalyurek in Ohio State Univ. It is fast and stable, and can handle partitioning with fixed cells and multi-contraints.

Note that these libraries are not open source, and must be obtained independently of the Bioconductor tools described here.

The pathRender package

The Cancer Molecular Analysis Project (cMAP) from NCICB[6] maintains a collection of genes organized by pathways and by ontology. The pathway interaction database is assembled from publicly available sources, represented in a formal, ontology-based manner. The pathways are modeled as directed graphs.

The data package **cMAP** contains annotated data from cMAP for use in bioconductor. For a given pathway, the package **pathRender** provides a tool to render this pathway as a whole or just a part of it.

pathRender does the following:

- it takes data from the cMAP database via package **cMAP**;
- it builds a graph where nodes represent molecules/proteins, edges represent interactions between them;
- it assigns different properties to the nodes and edges according to what they represent; then
- it uses package **Rgraphviz** to render such a graph on graphical display.

Outlook

For **RBGL**, we're planning to implement additional algorithms, such as algorithms for calculating clustering coefficient and transitivity of graphs, and computing *k*-cores of hypergraphs. Further experience is needed to establish more effective use of these algorithms in bioinformatic work flows.

Since bipartite graphs and hypergraphs are likely to play a substantial role, we will also introduce more specialized algorithms for these structures.

Acknowledgement

L.L. was supported by a grant from Intel Corp. to the Vital-IT Center.

Bibliography

E. Hartuv and R. Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(4–6):175–181, 2000. URL `citeseer.ist.psu.edu/hartuv99clustering.html`.

M. Kanehisa and S. Goto. KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Res*, 28: 27–30, 2000.

Li Long
Vital-IT Center
Swiss Institute of Bioinformatics
CH 1015 Lausanne
`Li.Long@isb-sib.ch`

Vincent J. Carey
Channing Laboratory
Brigham and Women's Hospital
Harvard Medical School
181 Longwood Ave.
Boston MA 02115, USA
`stvjc@channing.harvard.edu`

[4]`http://glaros.dtc.umn.edu/gkhome/views/metis`
[5]`http://bmi.osu.edu/~umit/software.html`
[6]`http://cmap.nci.nih.gov`

Modeling Package Dependencies Using Graphs

Introducing the pkgDepTools Package

by Seth Falcon

Introduction

Dealing with packages that have many dependencies, such as those in the Bioconductor repository, can be a frustrating experience for users. At present, there are no tools to list, recursively, all of a package's dependencies, nor is there a way to estimate the download size required to install a given package.

In this article we present the **pkgDepTools** package which provides tools for inspecting the dependency relationships among packages, generating the complete list of dependencies of a given package, and estimating the total download size required to install a package and its dependencies. The tools are built on top of the **graph** (Gentleman et al., 2006) package which is used to model the dependencies among packages in the BioC and CRAN repositories. Aside from this particular application to package dependencies, the approach taken is instructive for those interested in modeling and analyzing relationship data using graphs and the **graph** package.

Deciding how to represent the data in a graph structure and transforming the available data into a graph object are first steps of any graph-based analysis. We describe in detail the function used to generate a graph representing the dependency relationships among R packages as the general approach can be adapted for other types of data.

We also demonstrate some of the methods available in **graph**, **RBGL** (Carey and Long, 2006), and **Rgraphviz** (Gentry, 2006) to analyze and visualize graphs.

Modeling Package Dependencies

An R package can make use of functions defined in another R package by listing the package in the `Depends` field of its 'DESCRIPTION' file. The `available.packages` function returns a matrix of meta data for the packages in a specified list of CRAN-style R package repositories. Among the data returned are the dependencies of each package.

To illustrate our method, we use data from Bioconductor's package repositories as well as the CRAN repository. The Bioconductor project strongly encourages package contributors to use the dependency mechanism and build on top of code in other packages. The success of this code reuse policy can be measured by examining the dependencies of Bioconductor packages. As we will see, packages in the Bioconductor software repository have, on average, much richer dependency relationships than the packages hosted on CRAN.

A graph consists of a set of nodes and a set of edges representing relationships between pairs of nodes. The relationships among the nodes of a graph are binary; either there is an edge between a pair of nodes or there is not. To model package dependencies using a graph, let the set of packages be the nodes of the graph with directed edges originating from a given package to each of its dependencies. Figure 2 shows a part of the Bioconductor dependency graph corresponding to the **Category** package. Since circular dependencies are not allowed, the resulting dependency graph will be a directed acyclic graph (DAG).

The 'DESCRIPTION' file of an R package also contains a `Suggests` field which can be used by package authors to specify packages that provide optional features. The interpretation and use of the `Suggests` field varies, and the graph resulting from using this relationship in the Bioconductor repository is not a DAG; cycles are created by packages suggesting each other.

Building a Dependency Graph

To carry out the analysis, we need the **pkgDepTools** package along with its dependencies: **graph** and **RBGL**. We will also make use of **Biobase**, **Rgraphviz**, and **RCurl**. You can install these packages on your system using `biocLite` as shown below.

```
> u <- "http://bioconductor.org/biocLite.R"
> source(u)
> biocLite("pkgDepTools", dependencies=TRUE)
```

```
> library("pkgDepTools")
> library("RCurl")
> library("Biobase")
> library("Rgraphviz")
```

We now describe the `makeDepGraph` function that retrieves the meta data for all packages of a specified type (source, win.binary, or mac.binary) from each repository in a list of repository URLs and builds a `graph` instance representing the packages and their dependency relationships.

The function takes four arguments: 1) `repList` a character vector of CRAN-style package repository URLs; 2) `suggests.only` a logical value indicating whether the resulting graph should represent relations from the Depends field (FALSE, default) or the Suggests field (TRUE); 3) type a string indicating the type of packages to search for, the default is "source"; 4) keep.builtin which will keep packages that come with a standard R install in the dependency graph (the default is FALSE).

The definition of makeDepGraph is shown in Figure 1. The function obtains a matrix of package meta data from each repository using available.packages. A new graphNEL instance is created using new. A node attribute with name "size" is added to the graph with default value NA. When keep.builtin is FALSE (the default), a list of packages that come with a standard R install is retrieved and stored in baseOrRecPkgs.

Iterating through each package's meta data, the appropriate field (either Depends and Imports or Suggests) is parsed using a helper function cleanPkgField. If the user has not set keep.builtin to TRUE, the packages that come with R are removed from deps, the list of the current package's dependencies. Then for each package in deps, addNode is used to add it to the graph if it is not already present. addEdge is then used to create edges from the package to its dependencies. The size in megabytes of the packages in the current repository is retrieved using getDownloadSizesBatched and is then stored as node attributes using nodeData. Finally, the resulting graphNEL, depG is returned. A downside of this iterative approach to the construction of the graph is that the addNode and addEdge methods create a new copy of the entire graph each time they are called. This will be inefficient for very large graphs.

Definitions for the helper functions cleanPkgField, makePkgUrl, and getDownloadSizesBatched are in the source code of the **pkgDepTools** package.

Here we use makeDepGraph to build dependency graphs of the BioC and CRAN packages. Each dependency graph is a graphNEL instance. The out-edges of a given node list its direct dependencies (as shown for package **annotate**). The node attribute "size" gives the size of the package in megabytes.

```
> biocUrl <- biocReposList()["bioc"]
> cranUrl <- biocReposList()["cran"]
> biocDeps <- makeDepGraph(biocUrl)
> cranDeps <- makeDepGraph(cranUrl)
```

```
> biocDeps

A graphNEL graph with directed edges
Number of Nodes = 256
Number of Edges = 389
```

```
> cranDeps

A graphNEL graph with directed edges
Number of Nodes = 908
Number of Edges = 403
```

```
> edges(biocDeps)["annotate"]

$annotate
[1] "Biobase"
```

```
> nodeData(biocDeps, n = "annotate",
      attr = "size")

$annotate
[1] 1.451348
```

The degree and connectedComp methods can be used to compare the BioC and CRAN dependency graphs. Here we observe that the mean number of direct dependencies (out degree of nodes) is larger in BioC than it is in CRAN.

```
> mean(degree(biocDeps)$outDegree)

[1] 1.519531
```

```
> mean(degree(cranDeps)$outDegree)

[1] 0.4438326
```

A subgraph is connected if there is a path between every pair of nodes. The **RBGL** package's connectedComp method returns a list of the connected subgraphs. Examining the distribution of the sizes (number of nodes) of the connected components in the two dependency graphs, we can see that the BioC graph has relatively fewer length-one components and that more of the graph is a part of the largest component (87% of packages for BioC vs 50% for CRAN). The two tables below give the size of the connected components (top row) and the number of connected components of that size found in the graph (bottom row).

```
> table(listLen(connectedComp(cranDeps)))

  1   2   3   4   5   7  14 245
521  34  10   2   3   1   1   1
```

```
> table(listLen(connectedComp(biocDeps)))

  1   2   3 195
 32  10   3   1
```

Both results demonstrate the higher level of interdependency of packages in the BioC repository.

```
makeDepGraph <- function(repList, suggests.only=FALSE,
                         type=getOption("pkgType"),
                         keep.builtin=FALSE, dosize=TRUE)
{
    pkgMatList <- lapply(repList, function(x) {
        available.packages(contrib.url(x, type=type))
    })
    if (!keep.builtin)
      baseOrRecPkgs <- rownames(installed.packages(priority="high"))
    allPkgs <- unlist(sapply(pkgMatList, function(x) rownames(x)))
    if (!length(allPkgs))
      stop("no packages in specified repositories")
    allPkgs <- unique(allPkgs)
    depG <- new("graphNEL", nodes=allPkgs, edgemode="directed")
    nodeDataDefaults(depG, attr="size") <- as.numeric(NA)
    for (pMat in pkgMatList) {
        for (p in rownames(pMat)) {
            if (!suggests.only) {
                deps <- cleanPkgField(pMat[p, "Depends"])
                deps <- c(deps, cleanPkgField(pMat[p, "Imports"]))
            } else {
                deps <- cleanPkgField(pMat[p, "Suggests"])
            }
            if (length(deps) && !keep.builtin)
              deps <- deps[!(deps %in% baseOrRecPkgs)]
            if (length(deps)) {
                notFound <- ! (deps %in% nodes(depG))
                if (any(notFound))
                  depG <- addNode(deps[notFound], depG)
                deps <- deps[!is.na(deps)]
                depG <- addEdge(from=p, to=deps, depG)
            }
        }
        if (dosize) {
            sizes <- getDownloadSizesBatched(makePkgUrl(pMat))
            nodeData(depG, n=rownames(pMat), attr="size") <- sizes
        }

    }
    depG
}
```

Figure 1: The definition of the makeDepGraph function.

Using the Dependency Graph

The dependencies of a given package can be visualized using the graph generated by `makeDepGraph` and the **Rgraphviz** package. The graph in Figure 2 was produced using the code shown below. The `acc` method from the **graph** package returns a vector of all nodes that are accessible from the given node. Here, it has been used to obtain the complete list of **Category**'s dependencies.

```
> categoryNodes <- c("Category",
      names(acc(biocDeps, "Category")[[1]]))
> categoryGraph <- subGraph(categoryNodes,
      biocDeps)
> nn <- makeNodeAttrs(categoryGraph,
      shape = "ellipse")
> plot(categoryGraph, nodeAttrs = nn)
```

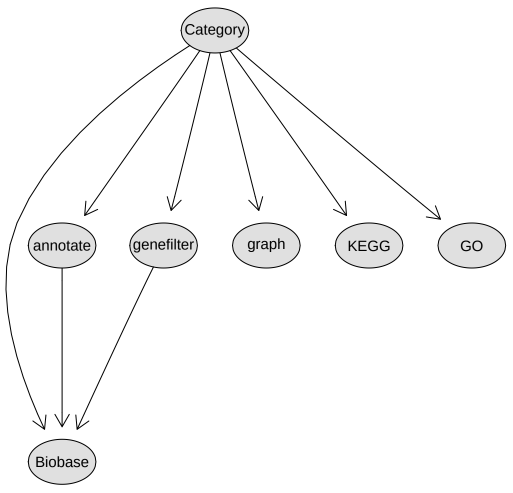


Figure 2: The dependency graph for the **Category** package.

In R , there is no easy to way to preview a given package's dependencies and estimate the amount of data that needs to be downloaded even though the `install.packages` function will search for and install package dependencies if you ask it to by specifying `dependencies=TRUE`. Next we show how to build a function that provides such a "preview" by making use of the dependency graph.

Given a plot of a dependency graph like the one for **Category** shown in Figure 2, one can devise a simple strategy to determine the install order. Namely, find the packages that have no dependencies, the leaves of the graph, and install those first. Then repeat that process on the graph that results from removing the leaves from the current graph. The download size is easily computed by retrieving the "size" node attribute for each package in the dependency list.

```
basicInstallOrder <- function(pkg, depG) {
    allPkgs <- c(pkg,
                 names(acc(depG, pkg)[[1]]))
    if (length(allPkgs) > 1) {
        pkgSub <- subGraph(allPkgs, depG)
        toInst <- tsort(pkgSub)
        if (!is.character(toInst))
          stop("depG is not a DAG")
        rev(toInst)
    } else {
        allPkgs
    }
}
```

Figure 3: Code listing for the `basicInstallOrder` function.

Figure 3 lists the definition of `basicInstallOrder`, a function that generates the complete dependencies for a given package using the strategy outlined above. The `tsort` function from **RBGL** performs a topological sort of the directed graph. A topological sort on a DAG gives an ordering of the nodes in which node *a* comes before *b* if there is an edge from *a* to *b*. Reversing the topological sort yields a valid install order.

The `basicInstallOrder` function can be used as the core of a "preview" function for package installation. The **pkgDepTools** package provides such a preview function called `getInstallOrder`. This function returns the *uninstalled* dependencies of a given package in proper install order along with the size, in megabytes, of each package. In addition, the function returns the total expected download size. Below we demonstrate the `getInstallOrder` function.

First, we create a single dependency graph for all CRAN and BioC packages.

```
> allDeps <- makeDepGraph(biocReposList())
```

Calling `getInstallOrder` for package **GOstats**, we see a listing of only those packages that need to be installed. Your results will be different based upon your installed packages.

```
> getInstallOrder("GOstats", allDeps)

$packages
    1.45MB     0.22MB      1.2MB
"annotate" "Category"  "GOstats"

$total.size
[1] 2.87551
```

When `needed.only=FALSE`, the complete dependency list is returned regardless of what packages are currently installed.

```
> getInstallOrder("GOstats", allDeps,
      needed.only = FALSE)
```

```
$packages
      0.31MB      17.16MB       1.64MB
     "graph"         "GO"    "Biobase"
      1.45MB       1.29MB       0.23MB
  "annotate"       "RBGL"       "KEGG"
      0.23MB       0.22MB        1.2MB
"genefilter"   "Category"    "GOstats"

$total.size
[1] 23.739
```

Wrap Up

We have shown how to generate package dependency graphs and preview package installation using the **pkgDepTools** package. We have described in detail how the underlying code is used and the process of modeling relationships with the **graph** package.

These tools can help identify and understand interdependencies in packages. A very similar approach can be applied to visualizing class hierarchies in R such as those implemented using the S4 (Chambers, 1998) class system or Bengtsson's **R.oo** (Bengtsson, 2006) package.

The **graph**, **RBGL**, and **Rgraphviz** suite of packages provides a very powerful means of manipulating, analyzing, and visualizing relationship data.

Bibliography

H. Bengtsson. *R.oo: R object-oriented programming with or without references*, 2006. URL http://www.braju.com/R/. R package version 1.2.3.

V. Carey and L. Long. *RBGL: Interface to boost C++ graph lib*, 2006. URL http://bioconductor.org. R package version 1.10.0.

J. M. Chambers. *Programming with Data: A Guide to the S Language*. Springer-Verlag New York, 1998.

R. Gentleman, E. Whalen, W. Huber, and S. Falcon. *graph: A package to handle graph data structures*, 2006. R package version 1.12.0.

J. Gentry. *Rgraphviz: Provides plotting capabilities for R graph objects*, 2006. R package version 1.12.0.

E. Hartuv and R. Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(4–6):175–181, 2000. URL citeseer.ist.psu.edu/hartuv99clustering.html.

M. Kanehisa and S. Goto. KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Res*, 28: 27–30, 2000.

Seth Falcon
Program in Computational Biology
Fred Hutchinson Cancer Research Center
Seattle, WA, USA
emailsfalcon@fhcrc.org

Image Analysis for Microscopy Screens

Image analysis and processing with EBImage

by Oleg Sklyar and Wolfgang Huber

The package **EBImage** provides functionality to perform *image processing* and *image analysis* on large sets of images in a programmatic fashion using the R language.

We use the term *image analysis* to describe the extraction of numeric features (*image descriptors*) from images and image collections. Image descriptors can then be used for statistical analysis, such as classification, clustering and hypothesis testing, using the resources of R and its contributed packages.

Image analysis is not an easy task, and the definition of image descriptors depends on the problem. Analysis algorithms need to be adapted correspondingly. We find it desirable to develop and optimize such algorithms in conjunction with the subsequent statistical analysis, rather than as separate tasks. This is one of our motivations for writing the package.

We use the term *image processing* for operations that turn images into images, with the goals of enhancing, manipulating, sharpening, denoising or similar (Russ, 2002). While some image processing is often needed as a preliminary step for image analysis, image processing is not the primary aim of the package. We focus on methods that do not require interactive user input, such as selecting image regions with a pointer etc. Whereas interactive methods can be extremely effective for small sets of images, they tend to have limited throughput and reproducibility.

EBImage uses the `Magick++` interface to the ImageMagick (2006) image processing library to implement much of its functionality in image processing and input/output operations.

Cell-based assays

Advances in automated microscopy have made it possible to conduct large scale cell-based assays with image-type phenotypic readouts. In such an assay, cells are grown in the wells of a microtitre plate (often a 96- or 384-well format is used) under a condition or stimulus of interest. Each well is treated with one of the reagents from the screening library and the response of the cells is monitored, for which in many cases certain proteins of interest are antibody-stained or labeled with a GFP-tag (Carpenter and Sabatini, 2004; Wiemann et al., 2004; Moffat and Sabatini, 2006; Neumann et al., 2006).

The resulting imaging data can be in the form of two-dimensional (2D) still images, three-dimensional (3D) image stacks or image-based time courses. Such assays can be used to screen compound libraries for the effect of potential drugs on the cellular system of interest. Similarly, RNA interference (RNAi) libraries can be used to screen a set of genes (in many cases the whole genome) for the effect of their loss of function in a certain biological process (Boutros et al., 2004).

Importing and handling images

Images are stored in objects of class `Image` which extends the `array` class. The colour mode is defined by the slot `rgb` in `Image`; the default mode is grayscale.

New images can be created with the standard constructor `new`, or using the wrapper function `Image`. The following example code produces a 100x100 pixel grayscale image of black and white vertical stripes:

```
> im <- Image(0, c(100,100))
> im[c(1:20, 40:60, 80:100),,] = 1
```

By using ImageMagick, the package supports reading and writing of more than 95 image formats including JPEG, TIFF and PNG. The package can process multi-frame images (image stacks, 3D images) or multiple files simultaneously. For example, the following code reads all colour PNG files in the working directory into a single object of class `Image`, converts them to grayscale and saves the output as a single multi-frame TIFF file:

```
> files <- dir(pattern=".png")
> im <- read.image(files, rgb=TRUE)
> img <- toGray(im)
> write.image(img, "single_multipage.tif")
```

Besides operations on local files, the package can read from anonymous HTTP and FTP sources, and it can write to anonymous FTP locations. These protocols are supported internally by ImageMagick and do not use R-connections.

The storage mode of grayscale images is `double`, and all R-functions that work with arrays can be directly applied to grayscale images. This includes the arithmetic functions, subsetting, histograms, Fourier transformation, (local) regression, etc. For example, the sharpened image in Figure 1**c** can be obtained by subtracting the slightly blurred, scaled in colour version of the original image (Figure 1**b**) from its source in Figure 1**a**. All pixels that become negative after subtraction are then re-set to background. The source image is a subset of the original microscopic image. Hereafter, variables in the code are given the same literal names as the corresponding image labels (e.g. data of variable `a` are shown in Figure 1 **a**, `b` – in **b**, and `C` – in **c**, etc).

```
> orig <- read.image("ch2.png")
> a <- orig[150:550, 120:520,]
> b <- blur(0.5 * a, 80, 5)
> C <- a - b
> C[C < 0] = 0
> C <- normalize(C)
```

One can think of this code as of a naive, but fast and effective, version of the *unsharp mask* filter; a more sophisticated implementation from the ImageMagick library is provided by the function `unsharpMask` in the package.

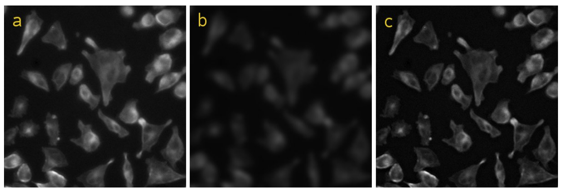


Figure 1: Implementation of a simple *unsharp mask* filter: (a) source image, (b) blurred colour-scaled image, (c) sharpened image after normalization

Some of the image analysis routines assume grayscale data in the interval $[0, 1]$, but formally there are no restrictions on the range.

The storage mode of RGB-images is `integer`, and we use the three lowest bytes to store red (R), green (G) and blue (B) values, each in the integer-based range of $[0, 255]$. Because of this, arithmetic and other functions are generally meaningless for RGB-images; although they can be useful in some special cases, as shown in the example code in the following section. Support for RGB-images is included to enhance the display of the analysis results. Most analysis routines require grayscale data though.

Image processing

The ImageMagick library provides a number of image processing routines, so-called *filters*. Many of those are ported to R by the package. The missing ones may be added at a later stage. We have also implemented additional image processing routines that we found useful for work on cell-based assays.

Filters are implemented as functions acting on objects of class `Image` and returning a new `Image`-object of the same or appropriately modified size. One can divide them into four categories: image *enhancement*, *segmentation*, *transformation* and *colour correction*. Some examples are listed below.

sharpen, unsharpMask generate sharpened versions of the original image.

gaussFilter applies the Gaussian blur operator to the image, softening sharp edges and noise.

thresh segments a grayscale image into a binary black-and-white image by the adaptive threshold algorithm.

mOpen, mClose use erosion and dilation to enhance edges of objects in binary images and to reduce noise.

distMap performs a Euclidean distance transform of a binary image, also known as *distance map*. On a distance map, values of pixels indicate how far are they away from the nearest background. Our implementation is adapted from the Scilab image processing toolbox (SIP Toolbox, 2005) and is based on the algorithm by Lotufo and Zampirolli (2001).

normalize shifts and scales colours of grayscale images to a specified range, normally $[0, 1]$.

sample.image proportionally resizes images.

The following code demonstrates how grayscale images recorded using three different microscope filters (Figure 2 **a**, **b** and **c**) can be put together into a single *false-colour* representation (Figure 2 **d**), and conversely, how a single false-colour image can be decomposed into its individual channels.

```
> files <- c("ch1.png","ch2.png","ch3.png")
> orig <- read.image(files, rgb=FALSE)
> abc <- orig[150:550, 120:520,]
> a <- toGreen(abc[,,1])            # RGB
> b <- toRed(abc[,,2])              # RGB
> d <- a + b + toBlue(abc[,,3])
> C <- getBlue(d)                   # gray
```

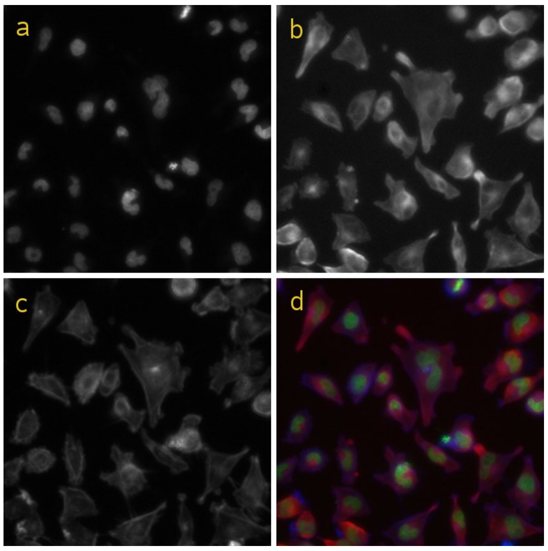


Figure 2: Composition of a false-colour image (d) from a set of grayscale microscopy images for three different luminescent compounds: (a) – DAPI, (b) – tubulin and (c) – phalloidin

Displaying images

The package defines the method `display` which shows images in an interactive X11 window, where image stacks can be animated and browsed through. This function does not use R graphics devices and cannot be redirected to any of those. To redirect display into an R graphics device, the method `plot.image` can be used, which is a wrapper for the `image` function from the **graphics** package. Since each pixel is drawn as a polygon, `plot.image` is much slower compared to `display`; also, it shows only the first image of a stack:

```
> display(abc)          # displays all 3
> plot.image(abc[,,2]) # can display just 1
```

Drawables

Pixel values can be set either by using the conventional subset assignment syntax for arrays (as in the third code example, `C[C < 0] = 0`) or by using *drawables*. **EBImage** defines the following instantiable classes for drawables (derived from the virtual `Drawable`): `DrawableCircle`, `DrawableLine`, `DrawableRect`, `DrawableEllipse` and `DrawableText`. The stroke and fill colours, the fill opacity and the stroke width can be set in the corresponding slots of `Drawable`. As the opportunity arises, we plan to provide drawables for text, poly-lines and polygons. Drawables can be drawn on `Image`s with the method `draw`; both grayscale and RGB images are supported with all colours automatically converted to gray levels on grayscale images.

The code below illustrates how drawables can be used to mark the positions and relative sizes of the nuclei detected from the image in Figure 2**a**. It assumes that x1 is the result of the function wsObjects, which uses a watershed-based image segmentation for object detection. x1 contains matrix objects with object coordinates (columns 1 and 2) and areas (column 3). The resulting image is shown in Figure 3**b**. This is just an illustration, we do not assume circular shapes of nuclei. For comparison, the actual segmentation boundaries are colour-marked in Figure 3**a** using the function wsPaint:

```
> src <- toRGB(abc[,,1])
> x <- x1$objects[,1]
> y <- x1$objects[,2]
> r <- sqrt(x1$objects[,3] / pi)
> cx <- DrawableCircle(x, y, r)
> b <- draw(src, cx)
> a <- wsPaint(x1, src)
```

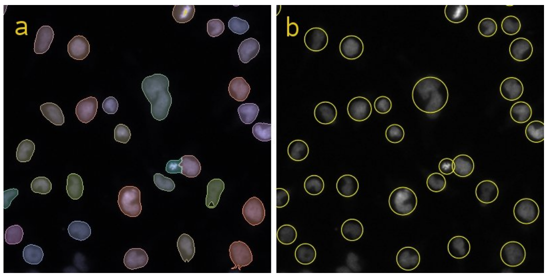


Figure 3: Colour-marked nuclei detected with function wsObjects: (a) – as detected, (b) – illustrated by DrawableCircle's.

Analysing an RNAi screen

Consider an experiment in which images like those in Figure 2 were recorded for each of ≈ 20,000 genes, using a whole-genome RNAi library to test the effect of gene knock-down on cell viability and appearance. Among the image descriptors of interest are the number, position, size, shape and the fluorescent intensities of cells and nuclei.

The package provides functionality to identify objects in images and to extract image descriptors in the function wsObjects. The function identifies different objects in parallel using a modified watershed-based segmentation algorithm and using image distance maps as input. The result is a list of three matrix elements objects, pixels (indices of pixels constituting the objects) and borders (indices of pixels constituting the object boundaries). If the supplied image is an image stack, the result is a list of such lists. Each row in the matrix objects corresponds to a detected object, with different object descriptors in the columns: x and y coordinates, size, perimeter, number of pixels on the image edge, acircularity, effective radius, perimeter to radius ratio, etc. Objects on the image edges can be automatically removed if the ratio of the detected edge pixels to the perimeter is larger than a given threshold. If the original image, from which the distance map was calculated, is specified in the argument ref, the overall intensity of the object region is calculated as well.

For every gene, the image analysis workflow looks, therefore, as follows: load and normalize images, segment, enhance the segmented images by morphological opening and closing, generate distance maps, identify cells and nuclei, extract image descriptors, and, finally, generate image previews with the identified objects marked.

Object descriptors can then be analysed statistically to cluster genes by their phenotypic effect, generate a list of genes that should be studied further in more detail (hit list), e.g., genes that have a specific phenotypic effect of interest, etc. The image previews can be used to verify and audit the performance of the algorithm through visual inspection.

A schematic implementation is illustrated in the following example code and in Figure 4. Here we omit the step of nuclei detection (object x1), from where the matrix of nuclei coordinates (object seeds) is retrieved to serve as starting points for the cell detection. The nuclei detection is done analogously to the cell detection without specifying starting points.

```
> for (X in genes) {
+  files <- dir(pattern=X)
+  orig <- read.image(files)
+  abc <- normalize(orig, independent=TRUE)
+  i1 <- abc[,,1]
+  i2 <- abc[,,2]
+  i3 <- abc[,,3]
+  a <- sqrt(normalize(i1 + i3))
+  b <- thresh(a, 300, 300, 0.0, TRUE)
+  C <-  mOpen(b, 1, mKernel(7))
+  C <- mClose(C, 1, mKernel(7))
+  d <- distMap(C)
+  # x1 <- wsObjects(...) - nuclei detection
+  seeds <- x1$objects[,1:2]
+  x2 <- wsObjects(d, 30, 10, .2, seeds, i3)
+  rgb <- toGreen(i1)+toRed(i2)+ toBlue(i3)
+  e <- wsPaint(x2, rgb, col="white",fill=F)
+  f <- wsPaint(x2, i3, opac = 0.15)
+  f <- wsPaint(x1, f, opac = 0.15)
+ }
```

Note that here we adopted the *record-at-a-time* approach: image data, which can be huge, are stored on a mass-storage device and are loaded into RAM in portions of just a few images at a time.

Summary

EBImage brings image processing and image analysis capabilities to R. Its focus is the programmatic (non-interactive) analysis of large sets of similar images, such as those that arise in cell-based assays for gene function via RNAi knock-down. Image descriptors can be analysed further using R's functionalities in machine learning (clustering, classification) and hypothesis testing.

Our current work on this package focuses on more accurate object detection and algorithms for feature/descriptor extraction. Image registration, alignment and object tracking are of foreseeable interest. In addition, one can imagine many other useful features, for example, support for more ImageMagick functions, better display options (e.g., using GTK) or interactivity. Contributions or collaborations on these or other topics are welcome.

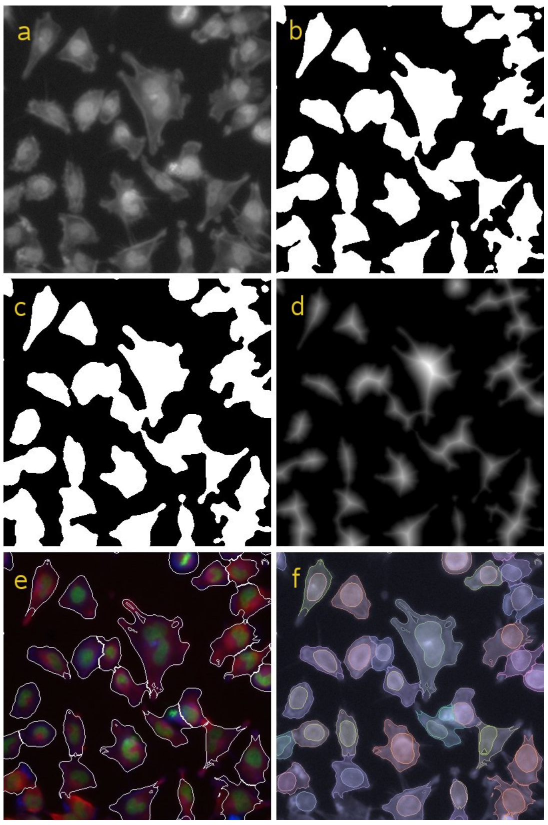


Figure 4: Illustration of the object detection algorithm: (a) – *sqrt*-brightened combined image of nuclei (DAPI from Figure 2**a**) and cells (phalloidin from Figure 2**c**); (b) – image **a** after *blur* and *adaptive thresholding*; (c) – image **b** after morphological *opening* followed by *closing*; (d) – normalized *distance map* generated from image **c**; (e) – outlines of cells detected using `wsObjects` drawn on top of the RGB image from Figure 2**d**; (f) – colour-mapped cells and nuclei as detected with `wsObjects` (one unique colour per object)

Installation

The package depends on ImageMagick, which needs to be present on the system to install the package. Please refer to the 'INSTALL' file.

Acknowledgements

We thank F. Fuchs and M. Boutros for providing their miscroscopy data and for many stimulating discussions about the technology and the biology, R. Gottardo and F. Swidan for testing the package on MacOS X and the European Bioinformatics Institute (EBI), Cambridge, UK, for financial support.

Bibliography

M. Boutros, A. Kiger, S. Armknecht, *et al.* Genome-wide RNAi analysis of cell growth and viability in Drosophila. *Science*, 303:832–835, 2004.

A. E. Carpenter and D.M. Sabatini. Systematic genome-wide screens of gene function. *Nature Reviews Genetics*, 5:11–22, 2004.

ImageMagick: software to convert, edit, and compose images. *Copyright: ImageMagick Studio LLC*, 1999-2006. URL http://www.imagemagick.org/

R. Lotufo and F. Zampirolli. Fast multidimensional parallel Euclidean distance transform based on mathematical morphology. *SIBGRAPI-2001/Brazil*, 100–105, 2001.

J. Moffat and D.M. Sabatini. Building mammalian signalling pathways with RNAi screens. *Nature Reviews Mol. Cell Biol.*, 7:177–187, 2006.

B. Neumann, M. Held, U. Liebel, *et al.* High-throughput RNAi screening by time-lapse imaging of live human cells. *Nature Mathods*, 3(5):385–390, 2006.

J. C. Russ. The image processing handbook – 4th ed. CRC Press, *Boca Raton*. 732 p., 2002

SIP Toolbox: Scilab image processing toolbox. *Sourceforge*, 2005. URL http://siptoolbox.sourceforge.net/

S. Wiemann, D. Arlt, W. Huber, *et al.* From ORFeome to biology: a functional genomics pipeline. *Genome Res.* 14(10B):2136–2144, 2004.

Oleg Sklyar and Wolfgang Huber
European Bioinformatics Institute
European Molecular Biology Laboratory
Wellcome Trust Genome Campus
Hinxton, Cambridge
CB10 1SD
United Kingdom
osklyar@ebi.ac.uk; huber@ebi.ac.uk

beadarray: An R Package to Analyse Illumina BeadArrays

by Mark Dunning, Mike Smith*, Natalie Thorne, and Simon Tavaré*

Introduction

Illumina have created an alternative microarray technology based on randomly arranged beads, each of which carries copies of a gene-specific probe (Kuhn et al., 2004). Random sampling from an initial pool of beads produces an array containing, on average, 30 randomly positioned replicates of each probe type. A series of decoding hybridisations is used to identify each bead on an array (Gunderson et al., 2004). The high degree of replication makes the gene expression levels obtained using BeadArrays more robust. Spatial effects do not have such a detrimental effect as they do for conventional arrays that provide little or no replication of probes over an array. Illumina produce two distinct BeadArray technologies; the SAM (Sentrix Array Matrix) and the BeadChip. A SAM is a 96-well plate arrangement containing 96 uniquely prepared hexagonal BeadArrays, each containing around 1500 bead types. The BeadChip technology comprises a series of rectangular strips on a slide with each strip containing around 24,000 bead types. The most commonly used BeadChip is the HumanRef-6 BeadArray. These have 6 pairs of strips, each pair having 24,000 RefSeq genes and 24,000 supplemental genes.

Until now, analysis of BeadArray data has been carried out by using Illumina's own software package (BeadStudio) and therefore does not utilise the wide range of bioinformatic tools already available via Bioconductor (Gentleman et al., 2004), such as **limma** (Smyth, 2005) and **affy** (Gautier et al., 2004). Also, the data output from BeadStudio only gives a single average for each bead type on an array. Thus potentially interesting information about the replicates is lost. The intention of this project is to create an R package, **beadarray**, for the analysis of BeadArray data incorporating ideas from existing Bioconductor packages. We aim to provide a flexible and extendable means of analysing BeadArray data both for our own research purposes and for the benefit of other users of BeadArrays. The **beadarray** package is able to read the full bead level data for both SAM and BeadChip technologies as well as bead summary data processed by BeadStudio. The package includes an implementation of Illumina's algorithms for image processing, although local background correction and sharpening are optional.

At present, **beadarray** is a package for the analysis of Illumina expression data only. For the analysis of Illumina SNP data, see the **beadarraySNP** package in Bioconductor.

^{1}These authors contributed equally to this work.

Data Formats

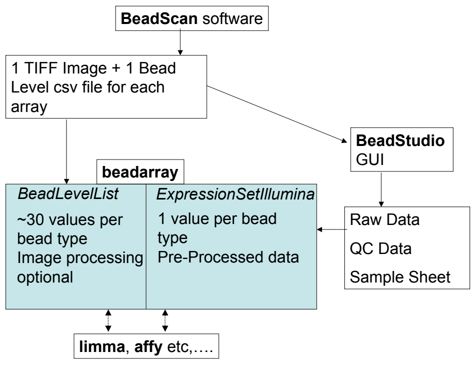


Figure 1: Diagram showing the interaction between **beadarray** and Illumina software. Various functions from Bioconductor packages (eg., **limma** and **affy**) may be used for the pre-processing or downstream analysis of `BeadLevelList` or `ExpressionSetIllumina` objects.

After hybridisation and washing, each SAM or BeadChip is scanned by the BeadScan software to produce a TIFF image for each array in an experiment. The size of each TIFF image is around 6Mb and 80Mb for SAM and BeadChip arrays respectively. BeadScan will also provide a text file describing the identity and position of each bead on the array if requested (version 3.1 and above of BeadScan only). We call this the bead level data for an array. Note that the bead level text file has to be produced when the arrays are scanned and cannot be generated later on. BeadStudio is able to read these raw data and produce a single intensity value for each bead type after outliers are excluded. These values are referred to as bead summary data. The images are processed using a local background correction and sharpening transformation (Dunning et al., 2005). These steps are not optional within the BeadStudio software. The bead summary values calculated by BeadStudio can be output directly with or without normalisation applied or used for further analysis within the application. The output from BeadStudio also contains information about the standard error of each bead type, the number of beads and a detection score that measures the probablity that the bead type is expressed; this uses a model assumed by Illumina. All analysis within BeadStudio is done on the unlogged scale and using the bead summary values rather than full replicate information for each bead type. Figure 1 gives an overview of the interaction between the file formats and software involved in the analysis of BeadArray data.

Bead Level Analysis

The `readBeadLevelData` function is used to read bead level data. A targets object is used in a similar way to **limma** to define the location of the TIFF images and text files. Both SAM and BeadChip data can be read by `readBeadLevelData`. The default options for the function use the sharpening transformation recommended by Illumina and calculate a local background value for each bead. The usage of `readBeadLevelData` is:

```
> BLData = readBeadLevelData(targets,
+ imageManipulation="sharpen")

> slotNames(BLData)

[1] "G"      "R"      "Rb"      "Gb"
[5] "GrnY"   "GrnX"   "ProbeID" "targets"
```

Even though the size of the TIFF images is rather large, `readBeadLevelData` uses C code to read these images quickly, taking around 2 seconds to process each SAM array and 1 minute for each BeadChip array on a 3Ghz Pentium©IV machine with 3Gb of RAM. An object of type `BeadLevelList` is returned by `readBeadLevelData`. The design of this class is similar to the `RGList` object used in **limma**. Note that we store the foreground (`G`) and background (`Gb`) intensities of each bead separately so that local background correction is optional using the `backgroundCorrect` function. Each bead on the array has a ProbeID that defines the bead type for that bead. We order the rows in the matrix according to the ProbeID of beads, making searching for beads with a particular ProbeID more efficient. The `GrnX` and `GrnY` matrices give the coordinates of each bead centre on the original TIFF image. Note that the random nature of BeadArrays means that the number of replicates of a particular bead type will vary between arrays and rows in the matrices will not always correspond to the same bead type, unlike data objects for other microarray technologies.

Typical quality control steps for conventional microarrays involve looking for systematic differences between arrays in the same experiment and also for large spatial effects on each array. The `boxplot` function can be easily applied to the foreground or background values to reveal possible defective arrays. We also provide the function `imageplot` for investigating the variation in foreground and background intensities over an array. This function is different from the functionality available in **limma** because BeadArrays do not have print-tip groups as found on conventional spotted microarrays. Therefore we define a grid of M × N rectangles, average over the $\log_2$ bead intensities found inside each rectangle and call the `image` function. In Figure 3 we can clearly see that the top-right corner of an array has a marked difference in intensity compared to the rest of the array, where the variation in intensity appears to be random. Such a spatial effect might be a cause for concern in conventional microarrays where probes representing the same gene appear in the same location on all arrays. In BeadArrays however, the arrangement of beads is random and hence spatial trends tend to have less impact. Whilst BeadStudio provides functionality to view the TIFF images, the resolution of the images is too high to be able to identify overall trends across an array. Also, the intensity levels between arrays cannot be easily compared. The `displayTIFFImage` function in **beadarray** allows users to view sections of TIFF images in a similar manner to BeadStudio but with the advantage of being able to see which beads are outliers. See Figure 3 of Dunning et al. (2005).

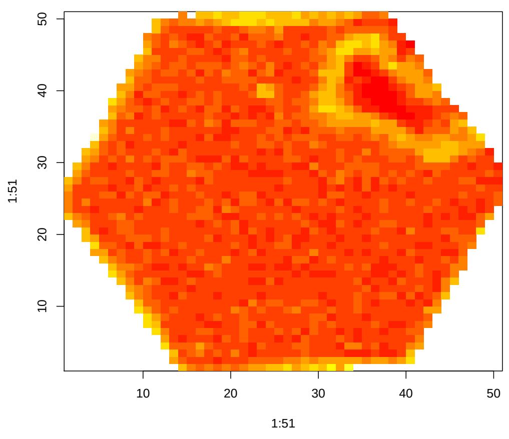


Figure 3: Plot showing the variation in $\log_2$ foreground intensity for a BeadArray from a SAM.

The random nature of BeadArrays and high replication rates allow for low level analysis that is not possible for other microarray technologies. For example, we can see where the replicates of a particular bead type are located on an array (`plotBeadLocations`) and view the intensities of the replicates (`plotBeadIntensities`). Due to the large number of bead types, in-depth analysis of each bead type is impractical. However, we might be interested in a subset of bead types, such as the controls used in the experiment. The `plotBeadIntensities` function produces a series of boxplots of $\log_2$ bead intensity for a supplied vector of ProbeIDs and can therefore

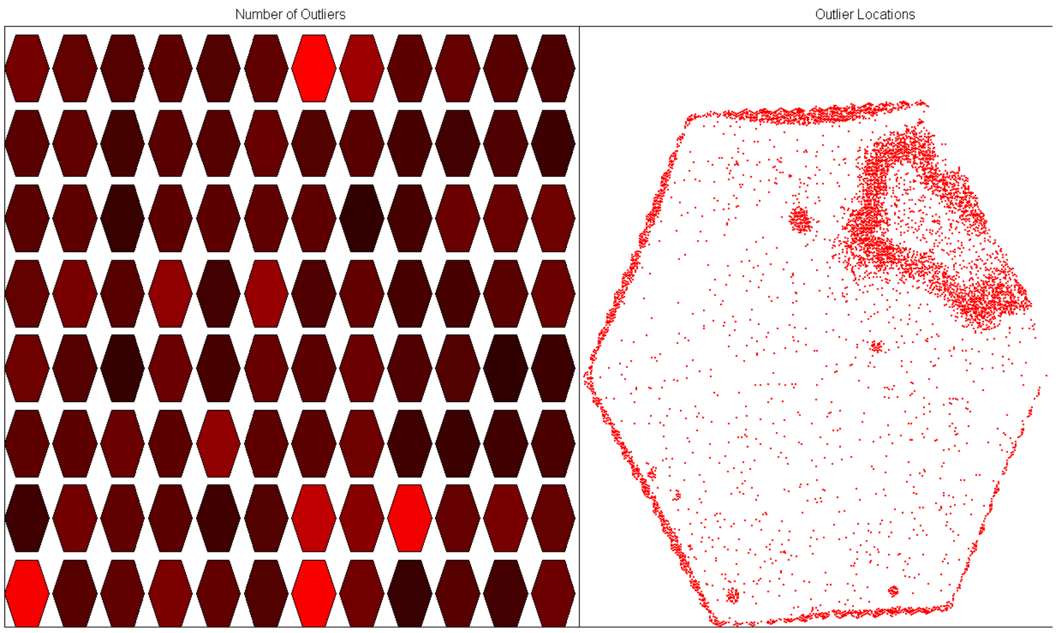


Figure 2: **beadarray** can be used to find spatial effects on arrays. On the left is a representation of the number of outliers for each array (bright red indicates more outliers) and on the right is the location of outliers for a particular array. Clicking on a hexagon on the left will change which array is displayed on the right. For this figure, the array in the 7th column of the first row of the SAM was chosen.

be a useful diagnostic tool.

Another set of beads that are of interest are outliers. Illumina exclude outliers for a particular bead type using a cut-off of 3 MADs from the unlogged mean. This analysis can be repeated for all bead types on an array using the `findAllOutliers` function. Users may specify a different cut-off as a multiple of the MAD or use the $\log_2$ intensity of beads in this function. Note that the outliers are not removed from the analysis at this point. To find all the outliers for every bead type on the first array we would use:

```
> o = findAllOutliers(BLData, array=1)
```

The result is a vector that indexes the rows of the first array in `BLData`. The `plotBeadLocations` function can then be used to plot the location of the outliers on the array.

```
> plotBeadLocations(BLData, array=1,
                    BeadIDs=o)
```

Typically, we find that 5% of beads on arrays are outliers and this can be used as an ad-hoc criterion for quality control. Figure 2 shows one of the interactive features available within **beadarray**. The left side of the screen gives an overview of all arrays on a SAM or BeadChip. In this example, each array is coloured according to the number of outliers and immediately we can see which arrays on the SAM have a larger number of outliers (shown in bright red). By clicking on a particular array in the graphic display, the location of all outliers on that array will be displayed on the right of the screen. This example shows the same array seen in Figure 3. As one might expect, many of the outliers in Figure 2 correspond to areas of higher foreground intensity visible in Figure 3.

Alternatively, the foreground or background intensities of arrays may be used to colour the arrays in the left screen and imageplots such as Figure 3 can be displayed when individual arrays are clicked. This interactive functionality is available for both SAM and BeadChip bead level data and can be started by the `SAMSummary` and `BeadChipSummary` functions respectively with the `BeadLevelList` created by `readBeadLevelData` passed as a parameter.

The `createBeadSummaryData` function creates bead summary data for each array by removing outliers for a particular bead type and then taking an average of the remaining beads. The `findAllOutliers` function is used by default and the result is an `ExpressionSetIllumina` object which can be analysed using different functionality within **beadarray**.

Reading Pre-Processed Bead Summary Data

Bead Summary data processed by BeadStudio can be read into **beadarray** using the function `readBeadSummaryData`. Three separate input files are required by the function, the location of which can be specified by a targets text file. The first two files are automatically created by BeadStudio by selecting the Gene Analysis option whereas the third file must be created by the user.

- Raw Data file - This contains the raw, non-normalised bead summary values as output by BeadStudio. Inside the file are several lines of header information followed by a data matrix. Each row is a different probe in the experiment and the columns give different measurements for the probes. For each array, we record the summarised expression level (AVG_Signal), standard error of the bead replicates (BEAD_ST_DEV), Number of beads used (Avg_NBEADS) and a Detection score which estimates the probability of a gene being detected above the background. Note that whilst this data has not been normalised, it has been subjected to local background correction at the bead level prior to summarising.
- QC Info - Gives the summarised expression values for each of the controls that Illumina place on arrays and hence is useful for diagnostic purposes. Each row in the file is a different array and the columns give average expression, standard error and detection for various controls on the array. See Illumina documentation for descriptions of control types.
- Sample Sheet - Gives a unique identifier to each array and defines which samples were hybridised to each array.

An example Bead Summary data set is included with the **beadarray** package and can be found as a zipped folder in the **beadarray** download. These data were obtained as part of a pilot study into BeadArray technology and comprises 3 HumanRef-6 BeadChips with various tumour and normal samples hybridised. The following code can be used to read the example data into R.

```
> targets <-
    readBeadSummaryTargets("targets.txt")
> BSData <- readBeadSummaryData(targets)

> BSData

Instance of ExpressionSetIllumina

assayData
  Storage mode: list
  Dimensions:
         BeadStDev Detection exprs NoBeads
Features     47293     47293 47293   47293
Samples         18        18    18      18

phenoData
  rowNames: I.1, IC.1,..., Norm.2, (18 total)
  varLabels and descriptions:

featureData
  featureNames: GI_10047089-S,...(47293 total)
  varLabels and descriptions:

Experiment data
  Experimenter name:
  Laboratory:
  Contact information:
  Title:
  URL:
  PMIDs:
  No abstract available.

Annotation [1] "Illumina"
QC Information
 Available Slots:  Signal StDev Detection
  featureNames: 1475542110_F,...1475542113_F
  sampleNames: Biotin, ..negative
```

The output of `readBeadSumamryData` is an object of type `ExpressionSetIllumina` which is an extension of the ExpressionSet class developed by the Biocore team used as a container for high-throughput assays. The data from the raw data file has been written to the assayData slot of the object, whereas the phenoData slot contains information from sample sheet and the QC slot contains the quality control information. For consistency with the definition of other `ExpressionSet` objects, we use `exprs` and `se.exprs` to access the expression and standard error slots.

Analysis of Bead Summary Data

The quality control information read as part of `readBeadSummaryData` can be retrieved using `QCInfo` and plotted using `plotQC`, which gives an overview of each control type. Plots of particular controls (e.g., negative controls) can be produced using `singleQCPlot` with the usual R plotting arguments available.

Scatter and M (difference) vs. A (average) plots can be generated for multiple arrays using the `plotMAXY` function (Figure 4). These can give a visual indicator of the variability between arrays in an experiment. For replicate arrays, we would expect to see the majority of points lying on the horizontal for the MA plots and along the diagonal for the scatter plots. Systematic deviation from these lines may indicate a bias between the arrays which requires normalising. An MA or scatter plot can also be produced for just two arrays at a time (`plotMA` and `plotXY` respectively). An attractive feature of these plots is that the location of particular genes (e.g., controls) can be highlighted using the `genesToLabel` argument.

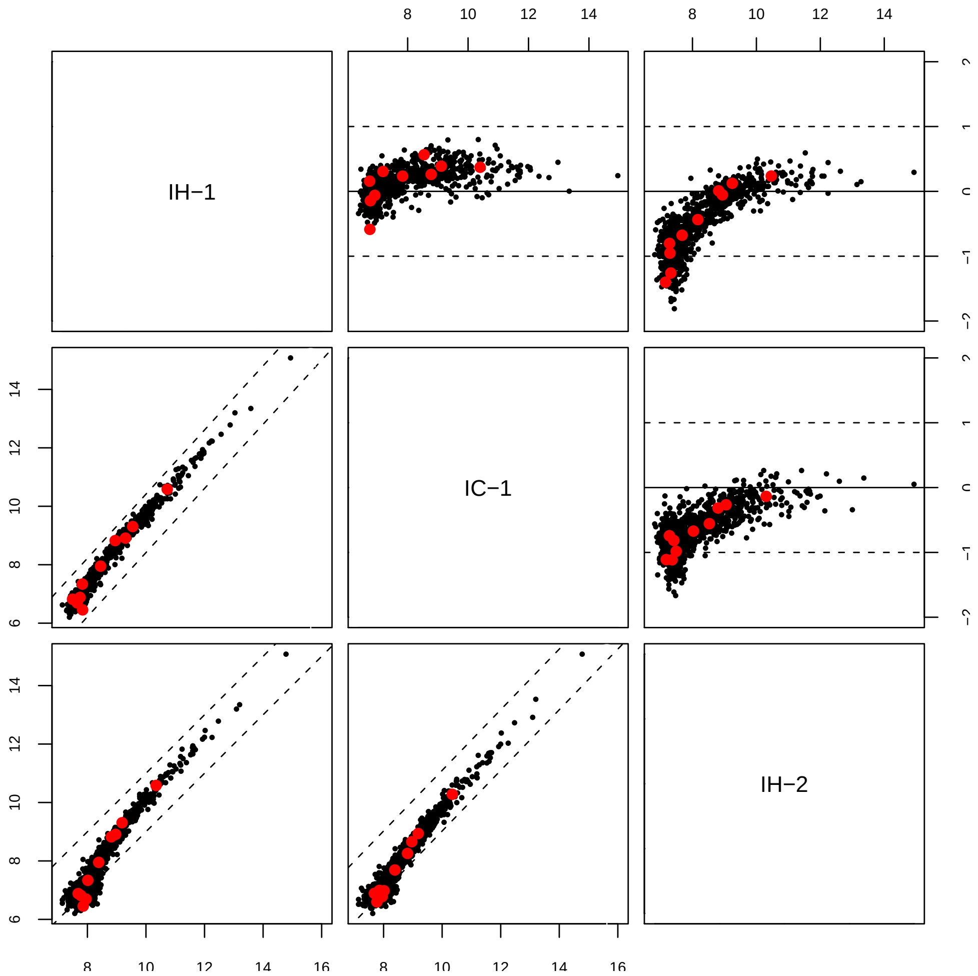


Figure 4: The `plotMAXY` function can be used to compare bead summary data from multiple arrays in the same experiment. In this figure we compare three replicates from the example bead summary data provided with **beadarray**

Since the `ExpressionSetIllumina` class includes a matrix of expression values, it can be analysed in a similar manner to data obtained from other technologies. In particular, this enables normalisation of BeadArray data to be carried out using the existing methods available in other Bioconductor packages (such as those available within the **affy** package, using `assayDataElementReplace` to replace the `exprs` slot). Illumina recommend normalising data by subtracting the average value of negative controls from each array. This method is implemented in the `backgroundNormalise` function and has quite a dramatic effect at lower intensities, often producing a lot of negative values. Typically, we find that the variability between arrays is low and a quantile or median normalisation (the function `medianNormalise` in **beadarray**) is sufficient.

The linear modelling tools within **limma** (Smyth, 2004) can be applied to the log-transformed expression `exprs` matrix in order to detect genes which are differentially expressed between arrays. The Illumina custom method is implemented by the function `DiffScore` but at present is only able to make pairwise comparisons between arrays.

Computational Issues and Future Plans

The vast amounts of data that can be generated from BeadArray experiments present a number of challenges for researchers, especially for analyses based on the bead level data. One has to consider that there are 12 80MB TIFF images for each BeadChip and 96 6MB TIFF Images for a SAM. In the case of a BeadChip experiment, simply reading the data into R for arrays from more than one BeadChip is problematic. We find that at least 1 Gb of RAM is required to run the `readBeadLevelData` and `createBeadSummaryData` functions on a `BeadLevelList` object representing one BeadChip. We hope to implement methods for normalisation which take the full bead level information into account but anticipate that this is going to be a computationally expensive task and may require the package to take advantage of parallel computing tools for R. Future versions of the package will also have better methods for creating bead averages which take information from replicate arrays into account.

Another major addition planned for **beadarray** is to allow sequence annotation information to be imported so that Illumina expression data can be combined with other microarray technologies such as arrayCGH, SNP and DNA methylation arrays. We plan to include functionality to read Illumina SNP and methylation data into the package.

Useful Illumina Resources

The vignettes supplied with the package give more detailed examples of how to analyse both bead level and bead summary data. Our previous paper (Dunning et al., 2005) provides descriptions of the image processing steps used by Illumina and some examples of bead level analysis. Readers interested in a comparison between Illumina and Affymetrix technologies are referred to Barnes et al. (2005).

We are keen to hear comments and feedback from users of **beadarray**.

Acknowledgements

We thank Brenda Kahl and Semyon Kruglyak (Illumina), Barbara Stranger, Matthew Forrest and Manolis Dermitzakis (Wellcome Trust Sanger Institute), and Andrew Lynch and John Marioni (University of Cambridge) for many helpful discussions during the devlopment of this work. We would also like to thank Isabelle Camilier (Ecole Polytechnique) for implementing the Illumina image processing algorithms and Roman Sasik (University of California, San Diego) for providing C code for reading TIFF images. The authors were supported by grants from Cancer Research UK (MS, NT & ST) and the Medical Research Council (MD). Simon Tavaré is a Royal Society / Wolfson Research Merit Award holder.

Bibliography

MJ Dunning, NP Thorne, I Camilier, *et al.* Quality control and low-level statistical analysis of Illumina BeadArrays. *Revstat*, 4:1–30, 2006.

RC Gentleman, VJ Carey, DM Bates, *et al.* Bioconductor: open software development for computational biology and bioinformatics. *Genome Biol*, 5:R80, 2004.

KL Gunderson, S Kruglyak, MS Graige, *et al.* Decoding randomly ordered DNA arrays. *Genome Res*, 14:870–877, 2004.

K Kuhn, SC Baker, E Chudin, *et al.* A novel, high-performance random array platform for quantitative gene expression profiling. *Genome Res*, 14:2347–2356, 2004.

L Gautier, L Cope, BM Bolstad, *et al.* **affy**–analysis of Affymetrix GeneChip data at the probe level. *Bioinformatics*, 20(3):307–15, 2004.

GK Smyth. Linear models and empirical Bayes methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology*, 3:113–136, 2004.

GK Smyth. Limma: linear models for microarray data. In R Gentleman, V Carey, W Huber, *et al. Bioinformatics and Computational Biology Solutions using R and Bioconductor*, pages 397–420. Springer, New York, 2005.

M Barnes, J Freudenberg, S Thompson, *et al.* Experimental comparison and cross-validation of the Affymetrix and Illumina gene expression analysis platforms. *Nucleic Acids Res*, 33:5914–5923, 2005.

Mark Dunning, Mike Smith, Natalie Thorne and Simon Tavaré
Computational Biology Group
Hutchison / MRC Research Centre
Department of Oncology
University of Cambridge
Hills Rd, Cambridge CB2 2XZ
United Kingdom
md392@cam.ac.uk
mls40@cam.ac.uk
npt22@cam.ac.uk
s.tavare@damtp.cam.ac.uk

Transcript Mapping with High-Density Tiling Arrays

by Matthew Ritchie and Wolfgang Huber

Introduction

Oligonucleotide tiling arrays allow the measurement of transcriptional activity and DNA binding events at a much higher resolution than traditional microarrays. Compared to the spotted technology, tiling arrays typically contain between 10 and 1000 times as many probes, which may be ordered or 'tiled' along entire chromosomes, or within specific regions of interest, such as promoters.

For RNA analysis, tiling arrays can be used to identify novel transcripts, splice variants, and antisense transcription (Bertone et al., 2004; Stolc et al., 2005). In DNA analysis, this technology can identify DNA binding sites through chromatin immunoprecipitation (ChIP) on chip analysis (Sun et al., 2003; Carroll et al., 2005) or genetic polymorphisms and chromosomal rearrangements via comparative genome hybridization (arrayCGH).

Due to the wide range of applications of this technology and the custom nature of the probe layout, the analysis of these data is different to that of regular microarrays. In this article, the **tilingArray** package, which extends the existing Bioconductor toolset to the problem of measuring transcriptional activity using Affymetrix high-density tiling arrays, is presented.

Background

The initial processing steps of quality assessment and normalization which are routinely applied to lower density arrays are also important when analyzing tiling array data. Diagnostic plots of the raw probe intensity data can highlight systematic biases or artefacts which may warrant the need for individual arrays or batches of arrays to be repeated. Normalization between arrays is necessary when data from multiple hybridizations is to be combined in an analysis. In the **tilingArray** package, a normalization method which uses the probe intensities from a DNA hybridization as a reference is implemented (Huber et al., 2006). The next step in the analysis is to detect the transcript boundaries. A simple change-point model, which segments the ordered chromosomal intensity data into discrete units has proven quite useful for whole genome tiling array data (David et al., 2006). Other approaches which use Hidden Markov Models (Toyoda and Shinozaki, 2005; Munch et al., 2006) or moving averages (Schadt et al., 2004) have also been proposed. Displaying the data with reference to the position along the chromosome allows visualization of the segmentation results. These capabilities will be demonstrated in the following sections.

The custom Affymetrix arrays used in this article were produced for the Stanford Genome Technology Center and tile the complete genome of *Saccharomyces cerevisiae* with 25mer probes arranged in steps of 8 bases along both strands of each chromosome. The two tiles per chromosome are offset by 4 bases (see Figure 1). Both perfect match (PM) and mismatch (MM) probes were available. The experimental data we analyze comes from David et al. (2006), and includes 5 RNA hybridizations from yeast cells undergoing exponential growth and 3 DNA hybridizations of labelled genomic fragments. This data is publicly available in the **davidTiling** package or from ArrayExpress (accession number E-TABM-14). A cell cycle experiment made up of RNA hybridizations from 24 time-points sampled at 10 minute intervals and 3 DNA hybridizations will also be used.

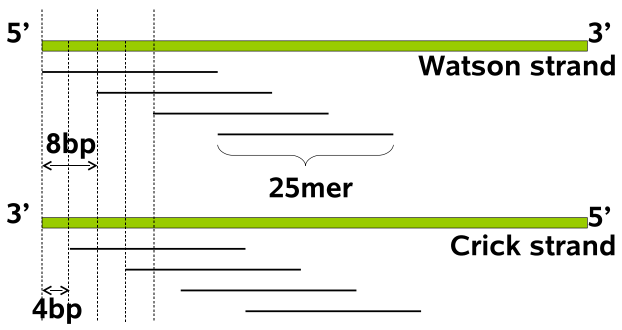


Figure 1: Probe spacing for *Saccharomyces cerevisiae* tiling arrays. The 25 mer probes are offset by 8 bases and tile each strand of DNA.

Reading data

We assume that the .CEL files from the **davidTiling** package are unzipped and available in the R working directory. To read in these data, the following commands can be used.

```
> library("tilingArray")
> cels = dir(pattern = ".cel")
> e = readCel2eSet(cels, rotated = TRUE)
```

The `readCel2eSet` function is is a wrapper for `ReadAffy` from the **affy** package. Rotating the .CEL file data through 90 degrees clock-wise by setting `rotated=TRUE` was necessary for these arrays due to old scanner settings. The data can also be loaded with

```
> library("davidTiling")
> data("davidTiling")
> dim(davidTiling)
```

```
Features  Samples
 6553600        8
```

The 8 arrays each contain 6,553,600 probes arranged in a grid with 2560 rows x 2560 columns. A special data structure is necessary for the mapping between the probes on the array and their target regions in the genome. For the yeast tiling array, we use an environment, named `probeAnno`, which organizes this information in two ways. Firstly, for each chromosomal strand, a vector of probe identifiers, in linear genomic order, and the coordinates and type of match is stored. In addition, the type of match and identifier of its hit region (e.g. YBR275C) are stored in the order that the probes appear on the array. To load, and find out further details about this environment, type

```
> data("probeAnno")
> ? probeAnno
```

To create this environment, the probes were mapped to the yeast genome using the MUMmer program (Delcher et al., 2002). In order to screen out ambiguous probes from the analysis, probes with multiple matches to the yeast genome where flagged and discarded from the analysis.

Users who wish to apply the **tilingArray** package to other types of tiling arrays, for example, from other species, need to produce their own `probeAnno`-like environment. Currently, there is no support for doing this. An objective of future work is to make this process and the genome mapping data structures more generic. We aim to use the infrastructure that will be provided by the **oligo** package for this purpose.

Once the data has been imported, some simple diagnostic plots can be generated with

```
> qcPlots(davidTiling, probeAnno = probeAnno)
```

This command generates an HTML report with image plots, box plots and density plots of log base 2 intensity data in the current working directory.

Normalization

The **tilingArray** package implements a DNA based normalization strategy in the `normalizeByReference` function. Normalization of the `davidTiling` data is carried out with the following commands

```
> isDNA = davidTiling$nucleicAcid ==
+     "genomic DNA"
> isRNA = davidTiling$nucleicAcid ==
+     "poly(A) RNA"
> pm = PMindex(probeAnno)
> bg = BGindex(probeAnno)
> yn = normalizeByReference(davidTiling[,
+     isRNA], reference = davidTiling[,
+     isDNA], pm = pm, background = bg)
```

The logical vectors `isDNA` and `isRNA` indicate which arrays are DNA and RNA hybridizations respectively. The intensities measured on the DNA hybridizations are used for two purposes. First, the probes indexed by the `bg` vector are used to estimate the background signal. Second, the PM intensities (indexed by the `pm` vector) provide a reference signal which is used to correct for probe specific responses due to base content. Once the PM intensities from the RNA hybridizations have been adjusted for background signal and probe effects, variance-stabilizing normalization between arrays using the `vsn` function (Huber et al., 2002) is applied to the data. For details of the method, see Huber et al. (2006).

Segmentation

Transcript boundaries are estimated from the data using a structural change model (SCM). Assume we have normalized intensity values z_{ki} from arrays $i = 1, \ldots, I$ and probes $k = 1, \ldots, n$ where the probe indexes (k) order the data by increasing position along the chromosome. We fit the SCM model

$$z_{ki} = \mu_s + \varepsilon_{ki} \qquad \text{for } t_s \le k < t_{s+1} \tag{1}$$

which has mean signal μ_s for the s-th segment, and residuals ε_{ki}. The change-points, $t_1, \ldots, t_S$ are the coordinates of the segment boundaries. This model was applied to arrayCGH data in Picard et al. (2005). The model is fitted separately for each strand of each chromosome. The algorithm is implemented in the function `segment`, which can be used directly on a matrix of data ordered along the chromosome.

To standardize some of the common data preprocessing steps, such as extracting the data for the chromosome of interest from `yn`, we use the wrapper function `segChrom`. To segment the data from chromosome 1, use

```
> seg1 = segChrom(yn, probeAnno = probeAnno,
+     chr = 1)
```

The `segment` algorithm is both time and memory intensive. On the example data, it uses a maximum of around 8 GB of RAM on larger chromosomes and takes several hours to complete. The computations for different chromosomes are trivially parallelizable, and the function `segChrom` offers a primitive mechanism for parallelization by synchronization through lock files.

The key parameter which the user must specify to the `segChrom` function is the maximum number of segments (S) to fit. The dynamic programming algorithm will then fit models with $1, 2, \ldots, S$ segments. S is specified via the parameter `nrBasesPerSegment`, the average number of bases per segment, which is used to set S by dividing the chromosome length by this number. For the data in David et al. (2006), the value `nrBasesPerSegment=1500` was chosen based on biological expectations and by manually inspecting the results obtained by varying this parameter. This value is obviously specific for these particular data. For other chip types or species, the value of `nrBasesPerSegment` needs to be adapted to the data.

The change-points t_s indicate transcript boundaries. Confidence intervals for the boundary locations are calculated using the **strucchange** package (Zeileis et al., 2002) when the `confint` argument of `segChrom` is set to `TRUE`.

Following the identification of transcript boundaries and levels through the segmentation algorithm, expression level changes for a given segment can be measured between different experimental conditions. Another approach to look for expression changes is to calculate a statistic for the condition-dependence for each probe and run the segmentation on this statistic.

Chromosome plots

Raw or normalized probe intensities can be plotted in along the chromosome order using the `plotAlongChrom` function. This function assumes that data is available for both strands of DNA and requires a `probeAnno` environment. Annotated genomic features can be included in these displays where a `GFF` (General Feature Format, see `http://www.sanger.ac.uk/Software/formats/GFF`) data frame is available.

An along the chromosome plot of the **davidTiling** data can be created with

```
> data("gff")
> plotAlongChrom(segObj = seg1,
+     probeAnno = probeAnno, gff = gff,
+     what = "dots", chr = 1, coord = c(156500,
+         160000))
```

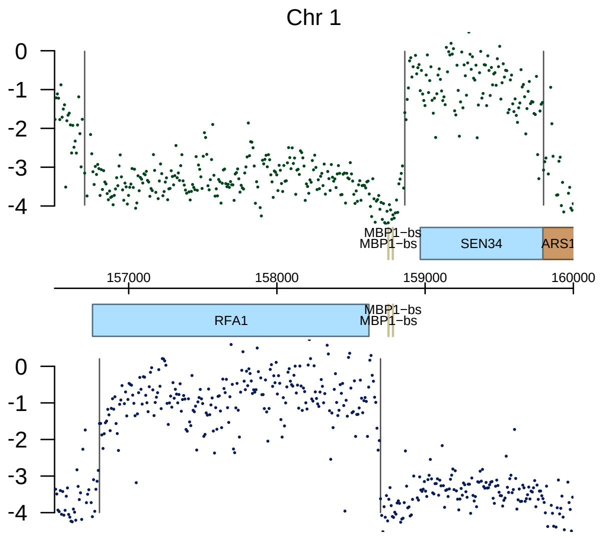


Figure 2: Plot of normalized intensity data (averaged between arrays) versus position along the chromosome (in bases) for a region of chromosome 1. The top and bottom panels display the probes from the Watson (+) and Crick (-) strands respectively. The gray lines indicate the segment boundaries (t_s) estimated from the data using `segChrom`. These boundaries correspond very closely with the known coordinates of the SEN34 and RFA1 genes, marked by blue boxes.

This plot (Figure 2) shows the result of the segmentation from the previous section; gray vertical lines indicate the transcript boundaries (t_s) and each point represents a probe. The `chr` and `coord` arguments specify the region plotted. The light blue boxes show the open reading frames of protein coding genes. Binding sites and other features from the gff data frame are also marked on the figure in the relevant locations. The absolute level of expression for different transcripts can be visualized in this display.

The following commands can be used to generate a heatmap display of the normalized intensities from the cell cycle data set (available in the object cycle), for the same region on chromosome 1 as in Figure 2.

```
> plotAlongChrom(y=cycle,
 + probeAnno=probeAnno, gff=gff,
 + what="heatmap", chr=1,
 + coord=c(156500,160000))
```

Figure 3 shows the resulting output. The periodic expression patterns for the cell cycle regulated SEN34 and RFA1 transcripts are clearly visible in this figure.

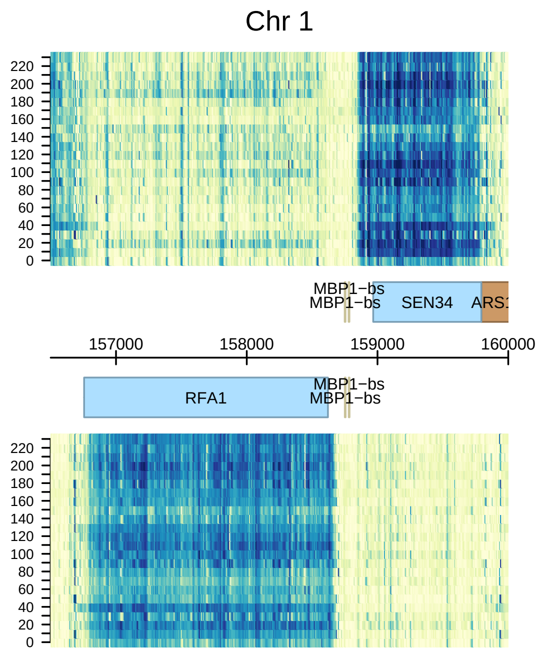


Figure 3: Heatmap plot of probe intensities from a cell cycle experiment for a region of chromosome 1. Each row in the y-direction displays the normalized intensities from a different time-point (0 minutes through to 230 minutes). The x-axis shows the position along the chromosome (in bases). The colorscheme indicates the intensity level, ranging from light yellow for lower intensities, through to dark blue for higher intensities.

Summary

The **tilingArray** package provides tools for reading, normalizing, segmenting and visualizing Affymetrix tiling array data. The core functions are normalizeByReference and segment, and their underlying methodology is described in Huber et al. (2006). These functions should be widely applicable to tiling array data.

There are also some functions and objects that have been customized to our specific *Saccharomyces cerevisiae* data sets, such as segChrom, plotAlongChrom and probeAnno. These functions can be used as templates for transferring the methods to other types of arrays and species, but this will require some work by the user. In future releases, we hope to use the infrastructure for tiling array data that will be offered by the **oligo** package to make these tools more generic.

Many exciting, open research questions still remain including a data-driven approach for selecting an optimal number of segments (S) for each chromosome (Huber et al., 2006), applying SCMs beyond piece-wise constant curves and the segmentation of condition-dependent transcription patterns.

Acknowledgements

We thank Jörn Tödling and Zhenyu Xu for contributions to the development of the **tilingArray** package, Lior David, Marina Granovskaia, Sandra Clauder-Münster and Lars Steinmetz for providing the data used in this article, Eugenio Mancera for Figure 1 and Rachel Uren for proof-reading the article.

Bibliography

P. Bertone, V. Stolc, T. E. Royce *et al.* Global identification of human transcribed sequences with genome tiling arrays. *Science*, 306(5705):2242–2246, 2004.

J. S. Carroll, X. S. Liu, A. S. Brodsky *et al.* Chromosome-wide mapping of estrogen receptor binding reveals long-range regulation requiring the forkhead protein FoxA1. *Cell*, 122:33–43, 2005.

L. David, W. Huber, M. Granovskaia *et al.* A high-resolution map of transcription in the yeast genome. *Proc Natl Acad Sci USA*, 103(14):5320–5325, 2006.

A. L. Delcher, A. Phillippy, J. Carlton and S. L. Salzberg. Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Res*, 30 (11):2478–2483, 2002.

W. Huber, A. V. Heydebreck, H. Sültmann *et al.* Variance stabilization applied to microarray data calibration and to the quantification of differential expression. *Bioinformatics*, 18(Suppl 1):S96–S104, 2002.

W. Huber, J. Toedling and L. M. Steinmetz. Transcript mapping with oligonucleotide high-density tiling arrays. *Bioinformatics*, 22(16):1963–1970, 2006.

K. Munch, P. Gardner, P. Arctander and A. Krogh. A hidden Markov model approach for determining expression from genomic tiling micro arrays. *BMC Bioinformatics*, 7:239, 2006.

F. Picard, S. Robin, M. Lavielle *et al.* A statistical approach for array CGH data analysis. *BMC Bioinformatics*, 6(1):27, 2005.

E. E. Schadt, S. W. Edwards, D. GuhaThakurta *et al.* A comprehensive transcript index of the human genome generated using microarrays and computational approaches. *Genome Biol*, 5(10):R73, 2004.

V. Stolc, M. Samanta, W. Tongprasit *et al.* Identification of transcribed sequences in Arabidopsis thaliana by using high-resolution genome tiling arrays. *Proc Natl Acad Sci USA*, 102(12):4453–4458, 2005.

L. V. Sun, L. Chen, F. Greil *et al.* Protein-DNA interaction mapping using genomic tiling path microarrays in Drosophila. *Proc Natl Acad Sci USA*, 100:9428–9433, 2003.

T. Toyoda and K. Shinozaki. Tiling array-driven elucidation of transcriptional structures based on maximum-likelihood and Markov models. *Plant J*, 43(4):611–621, 2005.

A. Zeileis, F. Leisch, K. Hornik and C. Kleiber. strucchange: An R package for testing for structural change in linear regression models. *Journal of Statistical Software*, 7:1–38, 2002.

Matt Ritchie
European Bioinformatics Insitute (EBI)
European Molecular Biology Laboratory (EMBL) Cambridge, UK
ritchie@ebi.ac.uk

Wolfgang Huber
European Bioinformatics Insitute (EBI)
European Molecular Biology Laboratory (EMBL) Cambridge, UK
huber@ebi.ac.uk

Analyzing Flow Cytometry Data with Bioconductor

by Nolwenn Le Meur and Florian Hahne

Introduction

In the recent past, flow cytometry (FCM) has become a high-throughput technique used in both basic and clinical research. Applications range from studies focusing on the immunological status of patients, therapeutic approaches involving stem cells up to functional screens used to identify specific phenotypes. The technology is capable of measuring multiple fluorescence as well as some morphological properties of individual cells in a cell population on the basis of light emission. FCM experiments can be extremely complex to analyze due to the large volume of data that is typically created in several processing steps. As an example, flow cytometry high content screening (FC-HCS) can process at a single workstation up to a thousand samples per day each containing thousands of cells, monitoring up to eighteen parameters per sample. Thus, the amount of information generated by these technologies must be stored and managed and finally needs to be summarized in order to make it accessible to the researcher.

Instrument manufacturers have developed software to drive the data acquisition process of their cytometers, but these tools are primarily designed for their proprietary instrument interface and offer few or no high level data processing functions. The packages **rflowcyt** and **prada** provide facilities for importing, storing, assessing and preprocessing data from FCM experiments. In this article we demonstrate the use of these packages for some common tasks in flow cytometry data analysis.

FCS format

In order to facilitate data exchange across different platforms, a data standard has been developed which is now widely accepted by the flow cytometry community and also by most instrument manufacturers. Flow Cytometry Standard (FCS) binary files contain both raw data and accompanying meta data of individual cytometry measurements and optionally the results of prior analyses carried out on the raw data (Seamer et al., 1997). The current version of the FCS standard is 3.0, but both packages can also deal with the old 2.0 standard which is still widely used. We can import FCS files into R using the function `read.fcs`.

Data models

Both **rflowcyt** and **prada** use their own object models to deal with FCM data. While the focus of **rflowcyt** is more on single cytometry measurements, **prada** offers the possibility to combine several individual measurements in the confines of a single experiment and to include all the necessary metadata. Its object model tries to stay close to the familiar micro-array data structures (`expressionSet`) making use of already defined generic functions. Both models store the data corresponding to the different immunofluorescence measurements or variables and the metadata included in the FCS files. The 2 main slots provide:

- a data frame with rows corresponding to the biological unit (i.e. cells) and columns corresponding to the measured variables
- the experimental metadata as a list (**rflowcyt**) or a named vector (**prada**)

The argument `objectModel` to `read.fcs` can be used to chose between the two models when importing the data. In addition, the package **rflowcyt** provides functions for the conversion between objects of both classes.

prada data model

Objects of class `cytoFrame` are the containers for storing individual cytometry measurements in **prada**. The data slot can be accessed using the function `exprs`, the metadata slot via the function `description`. Subsetting of the data is possible using the usual syntax for data frames and matrices.

```
> library(prada)
> data(cframe)
> cframe
```

```
cytoFrame object with 2115 cells and 8 observables:
FSC-H SSC-H FL1-H FL2-H FL3-H FL2-A FL4-H Time
slot 'description' has 148 elements
```

```
> subset <- cframe[1:3, c(1, 2,
+     3, 7, 8)]
> exprs(subset)
```

```
     FSC-H SSC-H FL1-H FL4-H Time
[1,]   467   532    87   449    2
[2,]   437   431    28   478    2
[3,]   410   214     0   358    2
```

```
> description(cframe)[4:6]
```

```
                                $SYS
"Macintosh System Software 9.2.2"
                             CREATOR
           "CellQuest Pro  4.0.2"
                                $TOT
                              "2115"
```

Collections of several cytometry measurements (whole experiments) can be stored in objects of class `cytoSet`. The phenoData slot of these objects contains all the relevant experiment-wide meta data. Multiple FCS files can be imported along with their metadata when the first argument to `read.fcs` is a vector of filenames or an object of class `phenoData` (see the documentation to `read.fcs` for more details). Subsetting of `cytoSets` is similar to subsetting of list, i.e., individual `cytoFrame` objects are returned when subsetting is done with double brackets.

```
> data(cset)
> cset
```

```
cytoSet object with 5 cytoFrames and colnames
 FSC-H SSC-H FL1-H FL2-H FL3-H FL2-A FL4-H Time
```

```
> subset <- cset[1:2]
> pData(subset)
```

```
                              name  ORF
2 fas-Bcl2-plate323-04-04.A02 MOCK
3 fas-Bcl2-plate323-04-04.A03  YFP
  batch
2     1
3     1
```

```
> class(cset[[3]])
```

```
[1] "cytoFrame"
attr(,"package")
[1] "prada"
```

`csApply` can be used to apply a function on all items of a `cytoSet`. In a simple case this could for instance be a preprocessing step or a statistical inference on the data from each well. In a more complex application, the function could summarize different features of the data and even produce diagnostic plots for visualization and quality assesment. Here, we apply a preprocessing function which removes artefactual measurements from our dataset based on the morphological properties of a cell and computes the number of cells in each of the wells on the plate.

```
> myFun <- function(xraw) {
+     fn <- fitNorm2(xraw[, c("FSC-H",
+         "SSC-H")], scale = 2)
+     x <- xraw[fn$sel, ]
+     return(nrow(x))
+ }
> cellCounts <- csApply(cset,
+     myFun)
```

Many of the common R methods like `plot` or `length` are also available for objects of class `cytoFrame` and `cytoSet`.

rflowcyt data model

Objects of class `FCS` are the containers for storing individual cytometry measurements in **rflowcyt**. The data slot can be accessed using the function `fluors`, the metadata slot via the function `metaData`. Subsetting of the data is possible using:

- [i,j] to extract or subset information from the data (a matrix object) of the FCS R-object
- [[i]] to extract metadata (which is of S4 class FCSmetadata) of the FCS R-object

```
> library(rflowcyt)
> data(VRCmin)
> st.DRT

Original Object of class FCS from:
DRT_GAG.fcs
Object name: st.DRT
Dimensions 206149 by 8

> subset <- st.DRT[1:3,1:3]
> metaData(subset)

 FACSmetadata for non-original FCS object:
st.DRT from original file DRT_GAG.fcs
 with  3 cells and 3 parameters.

> fluors(subset)
   FSC-Height Side Scatter CD8 FITC
1         640          458      298
2         136          294      102
3         588          539      265
```

Multiple cytometry measurements can be imported when the `filename` argument to `read.fcs` is a vector of file names and are stored as a list of individualFCS objects. These lists may be further processed using the familiar basic R functions, however, no experiment-wide metadata is provided.

Besides the FCS class, **rflowcyt** include `FCSmetadata`, `FCSsummary`, and `FCSgate` classes. `FCSmetadata` is the class of the metadata slot of an FCS R-object. The `FCSsummary` class is the class of the output of the summary method implemented on a FCS R-object. The `FCSgate` class contains the FCS class and extends it to include gating information (for more details, see the following section).

Gating

A common task in the analysis of flow cytometry data is to perform interactive selections of subpopulations of cells with respect to one or several measurement parameters, a process known as gating. In this respect, a gate is a set of rules that uniquely identifies a cell to be part of a given subpopulation. In the easiest case this can be a sharp cutoff, e.g., all cells with values in one parameter that are larger than a given threshold. But often much more complex selections are necessary like rectangular or elliptic areas in two dimensions or even polygonal boundaries. It is sometimes desirable to define a gate on a data set and later on apply this gate to a number of additional data sets, hence gates should be independent from the actual raw data. In addition, there may be several different combinations of gates that can be combined in a logical manner (i.e., "AND" and "OR") and in a defined order, thus the concept of the gate can be extended to collections of multiple gates.

The package **prada** offers the infrastructure to apply gating on cytometry data. Objects of class `gate` and `gateSet` model the necessary features of individual gates and of collections of multiple gates and can be assigned to the `gate` slot of objects of class `cytoFrame`. Gates can either be created from scratch by specifying the necessary selection rules or, much more conveniently, the function `drawGate` can be used to interactively set gates based on two-dimensional scatter plots of the raw data (Fig. 1). Please see the vignette of package **prada** for a more thorough discussion on gating.

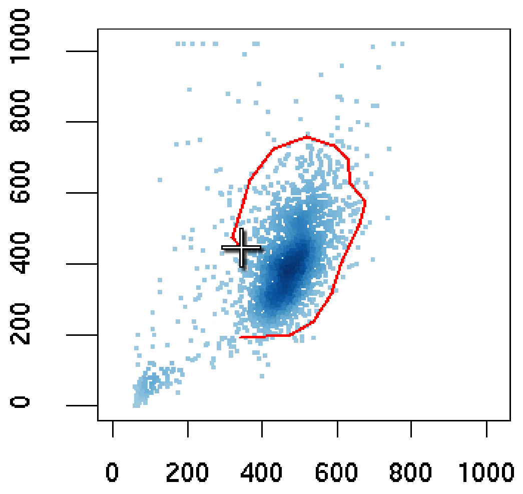


Figure 1: Interactive drawing of a polygonal gate based on a scatterplot of two cytometry parameters using function `drawGate`.

Quality control and quality assessment of cytometry data

Data quality control and quality assessment are crucial steps in processing high throughput FCM data. Quality control efforts have been made in clinical cell analysis by flow cytometry. For example, guidelines were defined to monitor the fluorescence measurements by computing calibration plots for each fluorescent parameter. However such procedures are not yet systematically applied in high throughput FCM and quality assessment of the raw data is often needed to overcome the lack of data quality control. The aim of data quality assessment is to detect systematic and stochastic effects that are not likely to be biologically motivated. The rationale is that systematic errors often indicate the need for adjustments in sample handling or processing. Further, the aberrant samples should be identified and potentially removed from any downstream analysis in order to avoid spurious results.

rflowcyt proposes a variety of graphical exploratory data analytic (EDA) tools to explore and summarize ungated FCM data.

- `plotECDF.FCS` creates Empirical Cumulative Distribution (ECDF) plots that reveal differences in distributions (Fig. 2);
- `boxplot.FCS` draws boxplots that display location and variation of the distributions and facilitate the comparison of these features between samples as they are aligned horizontally;
- `plotQA.FCS` summarizes the distribution of one or two parameters by their means, medians, modes or IQR for the diferent samples and displays the values in a scatterplot (Fig. 4). The dots in the resulting scatterplot can be colored according to the samples position in a 96-well plate to reveal potential plate effects;
- `plotdensity.FCS` displays density curves that reveal the shape of the distributions, especially multi-modality and asymmetry;

We illustrate the usefulness of those visualization tools to assess FCM data quality through examination of a collection of weekly peripheral blood samples obtained from a patient following allogeneic blood and marrow transplant. Samples were taken at various time points before and after transplantation. At each time point, every blood sample was divided into eight aliquots. Values for the forward light scatter (FSC) which measures a cell's size and for the sideward light scatter (SSC, a measure for a cell's granularity) of aliquots from the same sample should therefore be comparable.

The `plotECDF.FCS` function can be used to visualize several variables for several samples in the same graph.

```
> data(flowcyt.data)
> subset <- flowcyt.data[c(1:24,
+     41:48, 57:72)]
> stain <- paste("A", 1:8, sep = "")
> timePoint <- c(-8, 0, 5, 27,
+     39, 46)

> plotECDF.FCS(subset,
+             varpos = c(1),
+            var.list = c(paste("Day ",
+            timePoint)),
+            group.list = stain,
+            type = "l", xlab = "FSC",
+            lwd = 2, cex = 1.5)
```

For example, Figure 2 shows the FSC parameter for the 8 aliquots of a sample per time point. Each panel corresponds to a particular time point, in days before or after transplantation. In Figure 2 we expected to see the density curves superimpose. One aliquot significantly deviates from the other. This aliquot should be investigated in more detail and potentially be removed from further analysis.

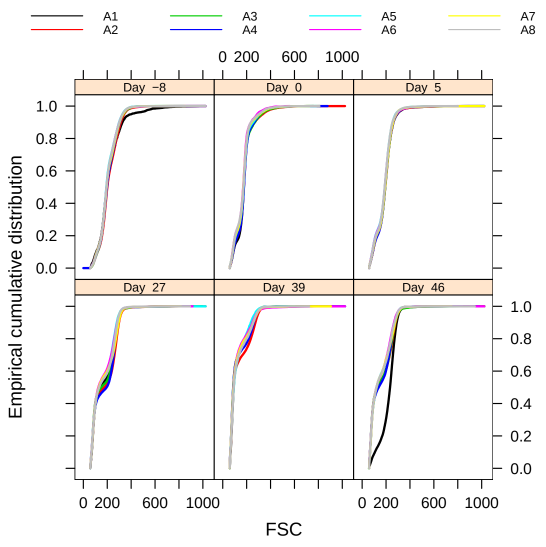


Figure 2: ECDF plots of the FSC parameter for 8 aliquots of a sample at different time point. Each panel corresponds to a particular time point, in days before or after transplantation. In each panel, each intensity curves represents one of the 8 aliquots.

ECDF plots are not good for visualizing the shape of the distributions. Instead, you can use the function `plotdensity.FCS` (Fig. 3).

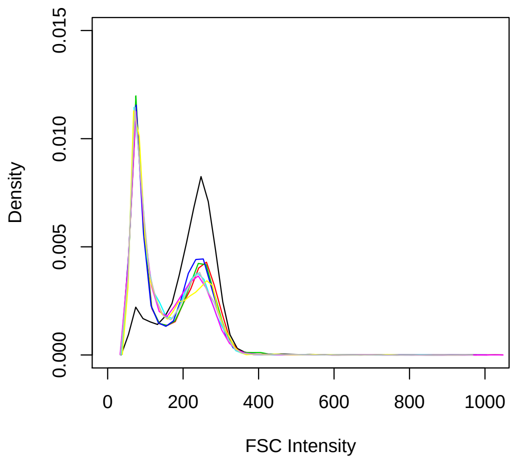


Figure 3: Density plots of the FSC parameter for 8 aliquots of the same sample.

```
> plotdensity.FCS(subset[41:48],
+     varpos = c(1), ylab = "Density",
+     xlab = "FSC Intensity",
+     col = c(1:8), ylim = c(0,
+         0.015))
```

Finally the `plotQA.FCS` function creates "summary" scatterplots to visualize samples relationship within plates. This representation allows to identify biological outlier and/or plate biases, such as edge effect or within-plate spatial effect. Figure 4 shows the SSC *vs* FSC median intensities for all aliquots stored in one 96-well plate and colored by their column position in the plate. In this figure, if all samples were identical, we expect to see a single cluster of data points. One has to be careful when interpreting such plots as each column correspond to different samples collected at different time points. However, we note that some columns have widely spread values (light blue and brown) and that one aliquot is an outlier as it is far away from the rest of its group. This aliquot appears to be the same as in Figures 2 and 3.

```
> idx <- order(names(flowcyt.data))
> flowcyt.data <- flowcyt.data[idx]
> plotQA.FCS(flowcyt.data, varpos = c(1,
+     2), col = "col", median,
+     labeling = TRUE, xlab = "SSC median",
+     ylab = "FSC median", xlim = c(0,
+         200), ylim = c(75, 275),
+     pch = "*", asp = 1, cex = 1.5,
+     main = "")
```

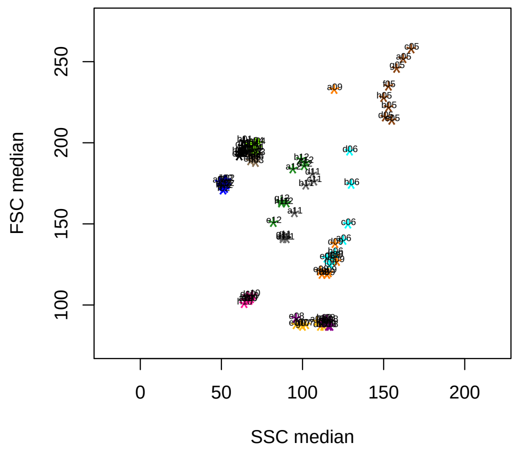


Figure 4: Scatterplot of SSC *vs* FSC medians intensities for one plate.

prada offers another visualization tool which can be used to inspect the data from whole experiments. Using the function `plotPlate` we can display quantitative as well as qualitative values or even complex graphs for each well of a microtiter plate retaining its array format 2. This allows for the identification of spatial effects and for a consise presentation of important features of an experiment. `plotPlate` is implemented using grid graphic and users are able to define their own plotting functions, so conceptually anything can be plotted in a mircrotiter plate format (see Figure 5).

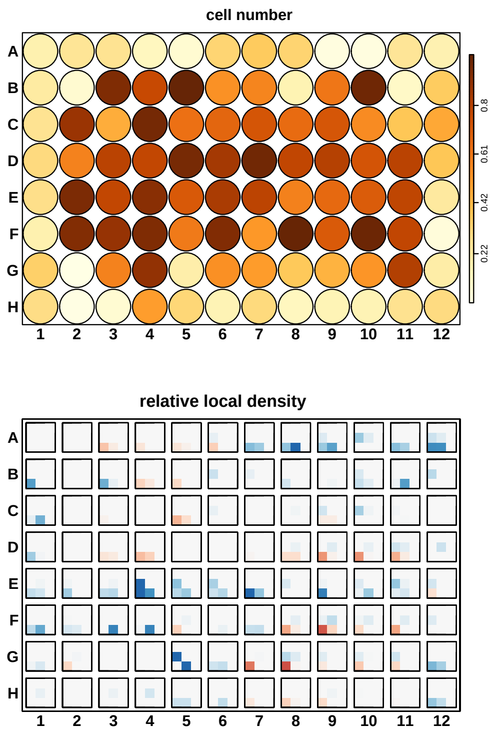


Figure 5: Two different variations of plate plots for 96 well microtiter plates. Top: Quantitative values. The consistently low number of cells around the edges of the plate indicates a technical problem. Bottom: Complex graph. Image maps of two-dimensional local densities of FSC vs SSC values for each well relative to a standard. Blue areas indicate low, red areas indicate high local densities. These plots help detect morphological changes in a cell population.

Discussion and Conclusion

The application of flow cytometry in modern cell biology is diverse and so are the demands on data analysis. The multitude of packages within R and Bioconductor already provides for many tools that are also useful in the analysis of FCM data. The packages **rflowcyt** and **prada** try to close the gap between data acquisition and data analysis by enabling the researches to take their data into the powerful R environment and to make use of the statistical and graphical solutions already available there. In addition, they provide for tools that are commonly used in early steps of data analysis which in principle are the same for all FCM applications.

Currently, in a collaboration of several groups involved in high-throughput FCM together with instrument manufacturers and members of the flow cytometry standards initiative a **flowCore** package and a number of additional FCM utility packages are developed. The aim is to merge both **prada** and **rflowcyt** into one core package which is copmpliant with the data exchange standards that are currently developed in the community. Visualization as well as quality control will than be part of the utility packages that depend on the data structures defined in the **flowCore** package.

Bibliography

L. C. Seamer, C. B. Bagwell, L. Barden *et al.* Proposed new data file standard for flow cytometry, version fcs 3.0. *Cytometry*, 28(2):118–122, Jun 1997.

Nolwenn Le Meur
Computational Biology
Fred Hutchinson Cancer Research Center
Seattle, WA, USA
nlemeur@fhcrc.org

Florian Hahne
Molecular Genome Analysis
German Cancer Research Center
Heidelberg, Germany
f.hahne@dkfz.de

Protein Complex Membership Estimation using apComplex

by Denise Scholtens

Graphs of protein-protein interactions, so called 'interactomes', are rapidly surfacing in the systems biology literature. In these graphs, nodes represent cellular proteins and edges represent interactions between them. Global interactome analyses are often undertaken to explore topological features such as network diameter, clustering coefficients, and node degree distribution. Local interactome modeling, particularly at the protein complex level, is also important for identifying distinct functional components of the cell and studying their interactivity (Hartwell et al., 1999). The **apComplex** package contains functions to locally estimate protein complex membership as described in Scholtens and Gentleman (2004) and Scholtens et al. (2005).

Two technologies are generally used to query protein-protein relationships. Affinity purification-mass spectrometry (AP-MS) technologies detect protein complex co-membership. In these experiments a set of proteins are used as baits, and in separate purifications, each bait identifies all hits with which it shares protein complex membership. AP-MS baits and their hits may physically bind to each other, or they may be joined together in a complex through an intermediary protein or set of proteins. If a bait protein is a member of more than one complex, all of its hits may not necessarily themselves be complex co-members. These biological realities become essential components of complex membership estimation. Publicly available AP-MS data sets for *Saccharomyces cerevisiae* include those reported by Gavin et al. (2002, 2006), Ho et al. (2002), and Krogan et al. (2004, 2006).

Yeast-two-hybrid (Y2H) technology is another bait-hit system that measures direct physical interactions. The distinction between AP-MS and Y2H data is subtle, but crucial. Two proteins that are part of the same complex may not physically interact with each other. Thus an interaction detected by AP-MS may not be detected by Y2H. On the other hand, two proteins that do physically interact by definition form a complex so any interaction detected by Y2H should also be detected by AP-MS. Under the same experimental conditions, Y2H technology should in fact consist of a subset of the interactions detected by AP-MS technology, the subset consisting of complex co-members that are physically bound to each other. Ito et al. (2001) and Uetz et al. (2000) both offer publicly available Y2H data sets for *Saccharomyces cerevisiae*.

apComplex deals strictly with data resulting from AP-MS experiments. The joint analysis of Y2H and AP-MS data is an interesting and important problem and is in fact an obvious next step after complex membership estimation, but is not currently dealt with in **apComplex**.

Protein Complex Membership and *Co*-Membership

apComplex estimates protein complex membership given a set of AP-MS co-membership data. The distinction between the complex membership and co-membership ties back to affiliation relationships in social networks analyses (Wasserman and Faust, 1999). As a simple example to be discussed throughout this article, suppose proteins P_1, P_2, P_4, and P_6 compose complex C_1 and proteins P_3, P_4, and P_5 compose complex C_2. Then their affiliation matrix, A, is as follows.

$$A = \begin{array}{c|cc} & C_1 & C_2 \\ \hline P_1 & 1 & 0 \\ P_2 & 1 & 0 \\ P_3 & 0 & 1 \\ P_4 & 1 & 1 \\ P_5 & 0 & 1 \\ P_6 & 1 & 0 \end{array}$$

A can also be represented as a bipartite graph in which one set of nodes represent proteins, another set represents complexes, and edges from proteins to complexes denote complex membership.

Instead of A, AP-MS technology assays $Y = A \otimes A'$ where $\otimes$ represents matrix multiplication under the Boolean algebra $0+0 = 0 \times 0 = 1 \times 0 = 0 \times 1 = 0$ and $1+0 = 0+1 = 1+1 = 1 \times 1 = 1$. Entries of 1 in Y represent co-membership of two proteins in a complex. In this simple example, we have Y as follows.

$$Y = \begin{array}{c|cccccc} & P_1 & P_2 & P_3 & P_4 & P_5 & P_6 \\ \hline P_1 & 1 & 1 & 0 & 1 & 0 & 1 \\ P_2 & 1 & 1 & 0 & 1 & 0 & 1 \\ P_3 & 0 & 0 & 1 & 1 & 1 & 0 \\ P_4 & 1 & 1 & 1 & 1 & 1 & 1 \\ P_5 & 0 & 0 & 1 & 1 & 1 & 0 \\ P_6 & 1 & 1 & 0 & 1 & 0 & 1 \end{array}$$

apComplex estimates A using assays of Y.

Observed Data

The entire matrix Y is not tested in AP-MS experiments. For this to happen, all cellular proteins would need to be used as baits. Even in small model organisms such as *Saccharomyces cerevisiae* with approximately 6000 cellular proteins, genome-wide testing is logistically prohibitive. Instead, only a subset of the rows of Y are tested (letting rows represent baits and columns represent hits). In a graph of Y, this is best described by neighborhood sampling of all edges extending from baits to all other proteins. Recognition of this sampling scheme is crucial as it draws a very important distinction between two types of edges that are absent from a graph of AP-MS data. Figure 1 shows ideal results from a neighborhood sampling of our simple interactome using P_1, P_2, and P_3 as baits. The edge between P_1 and P_3 is tested and observed to be absent, but the edge between P_4 and P_6 is absent because it is never tested. Scholtens and Gentleman (2004) discuss why inference should only be based on the tested edges, leaving the untested edges to be estimated or tested in further experiments.

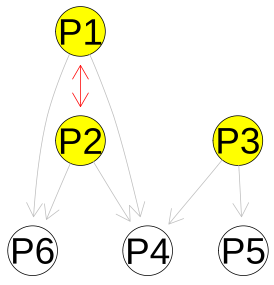


Figure 1: Co-memberships detected using neighborhood sampling scheme with P_1, P_2, and P_3 as baits.

In addition to being incomplete AP-MS data are also imperfect, including both false positive (FP) and false negative (FN) observations. Figure 2 demonstrates hypothetical FP observations from P_8 to P_3 and P_3 to P_7 and a FN observation from P_2 to P_4.

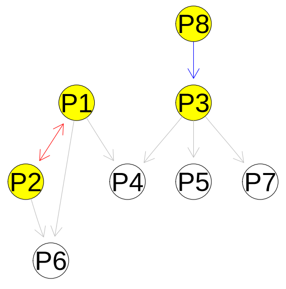


Figure 2: Observed data including FPs and FNs.

Penalized Likelihood Approach

Since edges in an AP-MS graph represent complex comembership, if all proteins were used as baits, then maximal complete subgraphs (or cliques) in the AP-MS graph would contain entire collections of proteins that compose a complex. The maximal complete subgraphs could then be used to estimate A (see Scholtens and Gentleman, 2004 and Scholtens et al., 2005 for a more thorough discussion).

Since all proteins are not used as baits, **apComplex** instead searches for *maximal BH-complete subgraphs* in the observed AP-MS data. A *BH-complete subgraph* is defined to be a collection of baits and hits for which all bait-bait edges and all bait-hit-only edges exist; a *maximal BH-complete subgraph* is a BH-complete subgraph that is not contained in any other

BH-complete subgraph. In other words, all edges observed in the neighborhood sampling scheme must exist for a subgraph to be *BH-complete*. The function `bhmaxSubgraph` can be used to find maximal BH-complete subgraphs in the observed AP-MS data.

In the event of unreciprocated observations between pairs of baits, the edges are estimated to exist when the sensitivity of the AP-MS technology is less than the specificity. Under a logistic regression model where the parameters represent sensitivity and specificity, this treatment of unreciprocated bait-bait edges maximizes the likelihood L for the data (Scholtens and Gentleman, 2004).

For our example, `bhmaxSubgraph` reports four maximal BH-complete subgraphs as shown below. Figure 3 depicts the bipartite graph of the results contained in `BP1`.

```
> apEX
   P1 P2 P3 P8 P4 P5 P6 P7
P1  1  1  0  0  1  0  1  0
P2  1  1  0  0  0  0  1  0
P3  0  0  1  0  1  1  0  1
P8  0  0  1  1  0  0  0  0
> BP1 <- bhmaxSubgraph(apEX)
   bhmax1 bhmax2 bhmax3 bhmax4
P1      0      1      1      0
P2      0      1      0      0
P3      1      0      0      1
P8      1      0      0      0
P4      0      0      1      1
P5      0      0      0      1
P6      0      1      1      0
P7      0      0      0      1
```

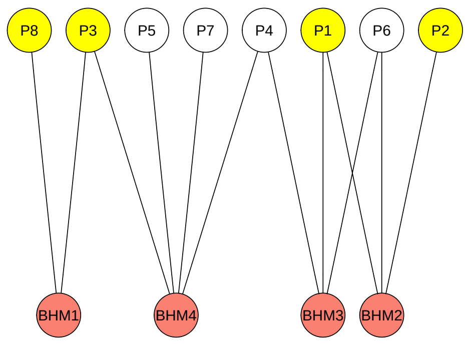


Figure 3: Bipartite graph for the initial estimate of A determined by locating maximal BH-complete subgraphs in the graph of observed AP -MS data.

The initial maximal BH-complete subgraph estimate of A does not allow missing edges between bait and hit-only proteins; since AP-MS technology is not perfectly sensitive, it is reasonable to expect a number of missing edges in the subgraph for each complex estimate. `mergeComplexes` accommodates this by employing a penalty term with the likelihood. For a complex c_k, let $C(c_k)$ represent the product of 1) the binomial probability for the number of observed edges in c_k given the number of tested edges and the supposed sensitivity of the technology, and 2) a two-sided p-value from Fisher's exact test for the distribution of missing incoming edges for complex estimate c_k. Then let C equal the product of $C(c_k)$ over all complexes $c_1, \ldots, c_K$. The penalized likelihood P is the product of L and C, or $P = L \times C$. L is maximized with the initial maximal BH-complete subgraphs – the algorithm in `mergeComplexes` looks to increase C in favor of small decreases in L.

After the initial estimate of A is made using `bhmaxSubgraph`, `mergeComplexes` proposes pairwise unions of individual complex estimates. If P increases when the complexes are treated as one, then the combination is accepted. If more than one union increases P, then the union with the largest increase is accepted. This is a greedy algorithm and `mergeComplexes` can be sensitive to the order in which the columns in the input matrix are specified. Users may want to order the columns putting the initial complex estimates with more bait proteins first since these contain proportionately more tested data.

```
> BP2 <- mergeComplexes(BP1, apEX,
sensitivity = 0.7, specificity = 0.75)
> BP2
   Complex1 Complex2 Complex3
P1        0        1        0
P2        0        1        0
P3        1        0        1
P8        1        0        0
P4        0        1        1
P5        0        0        1
P6        0        1        0
P7        0        0        1
```

Figure 4 shows the corresponding bipartite graph for the estimated A.

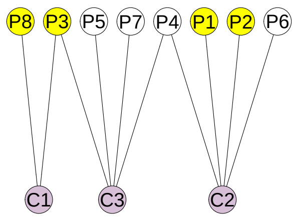


Figure 4: Bipartite graph for new complex estimates after using `mergeComplexes`.

The function `findComplexes` can be used to run both steps together. It will automatically reorder the input to `mergeComplexes` as suggested previously. Note that the user must specify the sensitivity and specificity of the AP-MS technology. Specificity should be considered in light of the dimension of the data under consideration. In our small example, a fairly high FP rate (i.e. low specificity) creates a reasonable number of suspected FP interactions. In large dimensional data sets, the number of true negative interactions is quite large. In these cases, very low FP probabilities (e.g. 0.001 or specificity=0.999) are usually appropriate.

Algorithm Output

apComplex makes three types of complex estimates: multi-bait-mult-edge (MBME) complexes that contain multiple baits and multiple edges, single-bait-multi-hit (SBMH) complexes that contain a single bait and a collection of hit-only proteins, and unreciprocated bait-bait (UnRBB) complexes that only contain two bait proteins connected by one unreciprocated edge. MBME complexes are the most reliable since they contain the most tested data. SBMH complexes are useful for proposing future experiments since the topology among the hit-only proteins is unknown. UnRBB complexes may result from FP observations since the edges are tested twice, observed once, and not confirmed by other subgraph edges. On the other hand, the unreciprocated edge may also result from a FN observation between the two baits. The complex estimates resulting from `mergeComplexes` or `findComplexes` can be sorted into the MBME, SBMH, and UnRBB components using the function `sortComplexes`.

Results for three publicly available data sets are included in **apComplex**. `TAP` is an adjacency matrix of the AP-MS data (called 'TAP') reported by Gavin, et al. (2002). There were 3420 comemberships reported using 455 baits and 909 hit-only proteins. The TAP data were originally compiled into 232 yTAP complexes, available in Supplementary Table 1 of Gavin et al. (2002) at `http://www.nature.com` and at `http://yeast.cellzome.com`. These yTAP complex estimates, along with the annotations given by Gavin, et al. are available in `yTAP`.

`HMSPCI` is an adjacency matrix of the AP-MS data (called 'HMS-PCI') reported by Ho et al. (2002). There were 3687 comemberships reported using 493 baits and 1085 hit-only proteins.

`Krogan` is an adjacency matrix of the AP-MS data reported by Krogan et al. (2004). There were 1132 comemberships reported using 153 baits and 332 hit-only proteins.

These data were analyzed using **apComplex**, and the results for the TAP and HMS-PCI data sets are described in Scholtens et al. (2005). Complex estimates are available for all three data sets - `MBMEcTAP`, `SBMHcTAP`, and `UnRBBcTAP` for the TAP data, `MBMEcHMSPCI`, `SBMHcHMSPCI`, and `UnRBBcHMSPCI` for the HMS-PCI data, and `MBMEcKrogan` for the Krogan data.

One example of the detail with which the **apComplex** algorithm can estimate complex membership involves the PP2A proteins Tpd3, Cdc55, Rts1, Pph21, and Pph22. These five proteins compose four heterotrimers (Jiang and Broach, 1999). Using the TAP data, **apComplex** accurately predicts these trimers as distinct complexes and furthermore notes the exclusive association of Zds1 and Zds2 with the Cdc55/Pph22 trimer. Confirmation of this prediction in the lab may help clarify the cellular function of this particular trimer and the reason for its joint activity with Zds1 and Zds2.

Related Packages

Several other packages based on the **apComplex** algorithm are currently being developed. The **ScISI** package contains an *in silico* interactome including **apComplex** estimates of publicly available AP-MS data. Given an interactome, **simulatorAPMS** can be used to simulate the neighborhood sampling scheme and both stochastic and systematic errors characteristic of AP-MS experiments for testing the performance of complex estimation algorithms, among other things. To complement the AP-MS data analysis, **y2hStat** contains algorithms for Y2H data analysis. This effort to better model Y2H observations will facilitate improved joint modeling of **apComplex** outputs and Y2H data.

Summary

In summary, **apComplex** can be used to predict complex membership using data from AP-MS experiments. An accurate catalog of complex membership is a fundamental requirement for understanding functional modules in the cell. Integration of **apComplex** analyses with other high-throughput data, including Y2H physical interactions, gene expression data, and binding domain data are promising avenues for further systems biology research.

Bibliography

A. C. Gavin *et al.* Functional organization of the yeast proteome by systematic analysis of protein complexes. *Nature*, 415:141–147, 2002.

A. C. Gavin *et al.* Proteome survey reveals modularity of the yeast cell machinery. *Nature*, 440:631–636, 2006.

L. Hartwell, J. Hopfield, S. Leibler *et al.* From molecular to modular cell biology. *Nature*, 402:C47, 1999.

Y. Ho *et al.* Systematic identification of protein complexes in *Saccharomyces cerevisiae* by mass spectrometry. *Nature*, 415:180–183, 2002.

T. Ito, T. Chiba, R. Ozawa, *et al.* A comprehensive two-hybrid analysis to explore the yeast protein interactome. *Proc. Nat. Acad. Sci. U.S.A.*, 98:4569–4574, 2001.

Y. Jiang and J. Broach. Tor proteins and protein phosphatase 2A reciprocally regulate Tap42 in controlling cell growth in yeast. *EMBO J.*, 18:2782–2792, 1999.

N. Krogan *et al.* High-definition macromolecular composition of yeast RNA-processing complexes. *Molecular Cell*, 13(2):225–239, 2004.

N. Krogan *et al.* Global landscape of protein complexes in the yeast *Saccharomyces cerevisiae*. *Nature*, 440:637–643, 2006.

D. Scholtens and R. Gentleman. Making sense of high-throughput protein-protein interaction data. *Statistical Applications in Genetics and Molecular Biology*, 3(1):Article 39, 2004.

D. Scholtens, M. Vidal, and R. Gentleman. Local modeling of global interactome networks. *Bioinformatics*, 21:3548–3557, 2005.

P. Uetz, L. Giot, G. Cagney, *et al.* A comprehensive analysis of protein-protein interactions in *Saccharomyces cerevisiae*. *Nature*, 403:623–627, 2000.

S. Wasserman and K. Faust. *Social Network Analysis*. Cambridge University Press, New York, 1999.

Denise Scholtens
Northwestern University Medical School
Chicago, IL, USA
dscholtens@northwestern.edu

SNP Metadata Access and Use with Bioconductor

by Vince Carey

Introduction

"Single nucleotide polymorphisms (or SNPs) ... are DNA sequence variations that occur when a single nucleotide in genomic sequence is altered"[1]. Conventionally, a given variation must be present in at least one percent of the population in order for the variant to be regarded as a SNP.

There are many uses of data on SNPs in bioinformatics. Two recent contributions which lay out aspects of the concept of "genetical genomics" are Li and Burmeister (2005) and Cheung et al. (2005). In this short contribution I review some functionality provided by Bioconductor for investigating analyses related to the Cheung *et al.* paper.

The *RSNPper* package

The SNPper[2] web service of the Children's Hospital (Boston) Informatics Program provides interactive access to a curated database of metadata on SNPs. Details of the system are provided in Riva and Kohane (2005). In addition to the browser-based interface, SNPper has an XML-RPC query resolution system. The *RSNPper* package provides an interface to this XML-RPC-based service. The objective of *RSNPper* is to provide a convenient high-level interface to the SNPper database contents, by providing a small number of high-level query functions with simple calling sequence, and by translating XML responses to convenient R-language objects for further use.

Getting gene-level information

A `geneInfo` function takes a string argument with a HUGO gene symbol and returns an object of class `SNPperGeneMeta`:

```
> cpm = geneInfo("CPNE1")
> cpm
SNPper Gene metadata:
There are  8 entries.
Basic information:
  GENEID  NAME CHROM STRAND  PRODUCT NSNPS
1  12431 CPNE1 chr20      - copine I   160
  TX.START   TX.END CODSEQ.START CODSEQ.END
1 33677382 33705245     33677577   33684259
  LOCUSLINK   OMIM  UNIGENE SWISSPROT
1      8904 604205 Hs.166887    Q9NTZ6
    MRNAACC   PROTACC REFSEQACC
1 NM_003915 NP_003906      NULL
SNPper info:
     SOURCE           VERSION
[1,] "*RPCSERV-NAME*" "$Revision: 1.38 $"
```

[1] http://www.ornl.gov/sci/techresources/Human_Genome/faq/snps.shtml
[2] snpper.chip.org

```
     GENOME DBSNP
[1,] "hg17" "123"
```

The notion of multiple "entries" mentioned in the `show` result concerns the multiplicity of mRNA and protein accession numbers referenced by annotation of the chosen gene. The `allGeneMeta` method provides access to such details.

```
> allGeneMeta(cpm)[,15:16]
    MRNAACC   PROTACC
1 NM_003915 NP_003906
2 NM_152925 NP_690902
3 NM_152926 NP_690903
4 NM_152927 NP_690904
5 NM_152928 NP_690905
6 NM_152929 NP_690906
7 NM_152930 NP_690907
8 NM_152931 NP_690908
```

Note that the `show` result gives a `GENEID` field, which is an internal SNPper-based index, which must be used for further gene-level queries. A `geneLayout` function provides information on the extents of the coding region and exons in a gene.

Getting SNP-level information

The `SNPinfo` function takes standard dbSNP[3] identifiers (deleting the `rs` prefix) and returns curated metadata:

```
> mysnp = SNPinfo("rs6060535")
> mysnp
SNPper SNP metadata:
     DBSNPID     CHROMOSOME POSITION
[1,] "rs6060535" "chr20"    "33698936"
     ALLELES VALIDATED
[1,] "C/T"   "Y"
There are details on 4 populations
and 10 connections to gene features
SNPper info:
     SOURCE          VERSION
[1,] "*RPCSERV-NAME*" "$Revision: 1.38 $"
     GENOME DBSNP
[1,] "hg17" "123"
```

Information on populations in which allele frequencies were analyzed is obtained with the `popDetails` method:

```
> popDetails(mysnp)
            PANEL   SIZE MAJOR.ALLELE
1        Japanese sanger            C
2     Han_Chinese sanger            C
3 Yoruba-30-trios sanger            C
4   CEPH-30-trios sanger            C
  MINOR.ALLELE    majorf    minorf
1            T 0.918605 0.0813954
2            T  0.94186 0.0581395
3            T    0.925     0.075
4            T      0.9       0.1
```

[3]www.ncbi.nlm.nih.gov/SNP

The genes near this SNP are described using the `geneDetails` method:

```
>  geneDetails(mysnp)[8:9,]
   HUGO LOCUSLINK
8 CPNE1      8904
9 RBM12     10137
                               NAME      MRNA
8                          copine I NM_152931
9 RNA binding motif protein 12 NM_006047
    ROLE RELPOS AMINO AMINOPOS
8   Exon -14677  <NA>     <NA>
9 3' UTR   7722  <NA>     <NA>
```

Broad queries can also be handled by this system. The *itemsInRange* function allows tabulation of SNPs in specific chromosomal regions:

```
> itemsInRange("countsnps", "chr20", "36000000",
    "37000000")
 total exonic nonsyn
  3679    145     48
```

If `"genes"` is supplied as the first argument, a list of genes and counts of SNPs related to those genes is returned.

The *RSNPper* interface package also includes `useSNPper`, permitting direct communication with the XML-RPC facility, returning XML to be parsed by the R user.

Exploring a genome-wide association study

Data representation

A marked benefit of Bioconductor architecture for analysis of datasets arising in high-throughput biology is the capacity for unifying diverse experimental result structures in S4 objects. For this illustration of inference in genetical genomics, we made an extension of the eSet class in Biobase to house expression and allele counts along with phenotype data. This extension is the `racExSet` class (`rac` connoting rare allele count), and an exemplar, `chr20GGdem`, is supplied with the package:

```
> chr20GGdem
racExSet (SNP rare allele count + expression)
rare allele count assayData:
  Storage mode: environment
  featureNames: rs4814683, ..., rs6062370,
    rs6090120 (117417 total)
  Dimensions:
             racs
Features 117417
Samples      58
```

```
expression assayData
  Storage mode: environment
  featureNames: 1007_s_at, ... (8793 total)
  Dimensions:
         exprs
Features  8793
Samples     58

phenoData
  rowNames: NA06985, ..., NA12892 (58 total)
  varLabels and varMetadata:
    sample: arbitrary numbering
...
```

Information on high-density SNP genotyping (here restricted to SNPs resident on chromosome 20) is accessible with the snps method:

```
> snps(chr20GGdem)[1:5,1:5]
           NA06985 NA06993 NA06994
rs4814683        2       0       0
rs6076506        0       0       0
rs6139074        2       0       0
rs1418258        2       0       0
rs7274499        0       0       0
           NA07000 NA07022
rs4814683        2       1
rs6076506        0      NA
rs6139074        2       1
rs1418258        2       1
rs7274499        0      NA
```

Entries count the number of copies of the rare allele in each subject's genotype.

The data noted here were provided by Vivian Cheung and Richard Spielman in conjunction with a summer course at Cold Spring Harbor Lab. This data will be provided in a Bioconductor experimental data package in the near future.

An association test

Figure 1 illustrates the test for association between a specific SNP (rs6060535) and expression measured in a probe set annotated to gene CPNE1. The p value reported by Cheung and Spielman for this test was 8.35×10^{-13}, in good agreement with the finding noted here. Comprehensive computation of such tests over a chromosome or in a specific region could be conducted with a simple iteration. Some optimizations of note include the elimination of SNPs for which all subjects sampled have identical genotype, and memoization of computations that depend only on the frequency distribution of genotypes, and not on their specific connection to outcomes.

Conclusions

Management of high-quality metadata on SNPs is a complex task. The XML document for dbSNP's data on chromosome 20 alone decompresses to 3GB. The Informatics Program at Children's Hospital Boston provides an extremely useful resource that can be queried interactively and programatically; *RSNPper* makes use of the Omegahat[4] XML interface of Duncan Temple Lang to simplify use of SNPper by the R community. More work on efficient data representation and algorithm design for genome-wide association studies is underway.

Bibliography

V. G. Cheung, R. S. Spielman, K. G. Ewens *et al.* Mapping determinants of human gene expression by regional and genome-wide association. *Nature*, 437 (7063):1365–9, 2005.

J. Li and M. Burmeister. Genetical genomics: combining genetics with gene expression analysis. *Human Molecular Genetics*, 14(R2):R163–R169, 2005.

A. Riva and I. Kohane. A SNP-centric database for the investigation of the human genome. *BMC Bioinformatics*, 5(33), 2005.

Vincent J. Carey
Channing Laboratory
Brigham and Women's Hospital
Harvard Medical School
181 Longwood Ave.
Boston MA 02115, USA
stvjc@channing.harvard.edu

[4]www.omegahat.org

```
Call:
lm(formula = exprs(chr20GGdem)["206918_s_at", ] ~ snps(chr20GGdem)["rs6060535",
    ])

Residuals:
     Min       1Q   Median       3Q      Max
-0.54749 -0.17590  0.02143  0.17102  0.64717

Coefficients:
                                Estimate Std. Error t value Pr(>|t|)
(Intercept)                      7.63381    0.04027  189.57  < 2e-16
snps(chr20GGdem)["rs6060535", ] -0.84324    0.08197  -10.29 1.62e-14

(Intercept)                     ***
snps(chr20GGdem)["rs6060535", ] ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2782 on 56 degrees of freedom
Multiple R-Squared: 0.654,      Adjusted R-squared: 0.6478
F-statistic: 105.8 on 1 and 56 DF,  p-value: 1.619e-14
```

Figure 1: Call and report on a specific fit.

Integrating Biological Data Resources into R with biomaRt

by Steffen Durinck

Abstract

Comprehensive analysis of data generated from high-throughput biological experiments, involves integration of a variety of information that can be retrieved from public databases. A simple example is to annotate a set of features that are found differentially expressed in a microarray experiment with corresponding gene symbols and genomic locations. Most public databases provide access to their data via web browsers. However, a major remaining bioinformatics challenge is how to efficiently have access to this biological data from within a data analysis environment. BioMart is a generic, query oriented data management system, capable of integrating distributed data resources. It is developed at the European Bioinformatics Institute (EBI) and Cold Spring Harbour Laboratory (CSHL). We first describe a number of important public biological databases, such as Ensembl and Uniprot, that implement the BioMart data management system. And then describe biomaRt, a software package aimed at integrating data from BioMart systems into R, enabling biological data mining.

Biological databases

In recent years, biological databases have become repositories containing large amounts of heterogeneous data. In the next paragraphs we discuss some of these public databases which have a BioMart implementation.

The completion of the human genome sequencing project and other sequencing efforts resulted in a surge of available sequence data. Dedicated databases and genome browsers have been set up that make these fully or partially sequenced genomes available to the community together with annotation information. Ensembl is a software system that produces and maintains automatic annotation on selected eukaryotic genomes (Birney et al., 2006). At the time of writing, Ensembl (http://www.ensembl.org) covers 25 annotated genomes from a wide range of organisms such as human, mouse, and dog. There are many ways to use meta-data from Ensembl in biological data analysis. Examples are to annotate a variety of identifiers with the corresponding gene symbol, a description of the gene and the gene's chromosomal coordinates. Alternatively one can extract a subset of identifiers that fulfill a certain requirement set, such as being located on human chromosome 19 and having the Gene Ontology (GO) term for *imprinting* associated with it. The ability to calculate and display integrated comparative genomics resources is another important feature of Ensembl (Birney et al., 2006). This allows one for example to easily go from a set of mouse identifiers to identifiers of corresponding homologs in human.

Genomes display natural polymorphisms, which are variations in the genome sequence. Two human genomes differ in about 1 base pair per 1000 bases. Most of these variations are Single Nucleotide Polymorphisms (SNPs), where one nucleotide is changed into another. Other sources of the variation are attributable to deletions and insertions, and copy number polymorphisms.

dbSNP (Sherry et al., 2001) is a database located at the NCBI, and stores this variation data. This database is also mirrored by Ensembl. A second resource of variation data has been delivered by the HapMap project of which Phase I was completed in November 2005 (The international hapmap consortium, 2005). The HapMap project aims to determine common patterns in human genome variation by characterizing sequence variants and their frequencies in human populations. The HapMap database (http://www.hapmap.org) implements the BioMart data management system and contains more than one million SNPs. This database provides a unique data collection to for example find associations between the reported SNPs and differences in gene expression.

The Vertebrate Genome Annotation database (VEGA[1]) stores high quality manual annotations of finished vertebrate genomes (Ashurst et al., 2005) and thus differs from Ensembl, which stores computationally derived gene predictions on finished and unfinished genomes. At the time of writing, the VEGA database contained five species: human, mouse, zebrafish, pig and dog. Manually annotated information that is available from VEGA includes gene symbol, gene description, location, OMIM[2] identifiers, HUGO[3] identifiers, and InterPro[4] protein domains.

The UniProt database (http://www.ebi.uniprot.org, Wu et al., 2006) is the most comprehensive catalog of information on proteins. The database

[1]http://vega.sanger.ac.uk
[2]Online Mendelian Inheritance in Men
[3]Human Genome Organization
[4]InterPro is a database of protein domains.

provides a resource for protein sequences as well as functional annotation. The annotations provided by UniProt include protein name and function, protein domains, taxonomy, and post-translational modifications.

Wormbase (http://www.wormbase.org) is a database dedicated to the model species *Caenorhabditis elegans* (Schwarz et al., 2006). In addition to gene-centric queries, Wormbase supports querying expression patterns, RNAi phenotypes, mutant phenotypes, genome variations, and literature citations.

The Gramene database (http://www.gramene.org) is a resource for comparative genomics of grasses (Ware et al., 2002) and currently contains data on *Zea Mays* and *Oryza Sativa*. Additional to the grasses, Gramene also stores information on the plant model species *Arabidopsis*. Examples of data present in Gramene is Gene Ontology and Plant Ontology associations, genome sequences, Trait Ontology terms (an ontology describing associated traits of Quantitative Trait Loci), and phenotypic descriptions.

BioMart

BioMart (http://www.biomart.org) is a joint project between the European Bioinformatics Institute (EBI) and Cold Spring Harbor Laboratory (CSHL) and aims to develop a generic, query oriented data management system, capable of integrating distributed data resources. BioMart systems have a three-tier architecture. The first tier consists of one or multiple relational databases. Central to these BioMart databases is the concept of the star and the reverse-star schemas, of which the former consist of a single main table linked to different dimension tables and the latter is a variant (Kasprzyk et al., 2004). The overall simplicity of these schemas avoids complex joins and enables fast data retrieval.

The second tier consists of two Application Programming Interfaces (APIs), one written in Java and the other in Perl. The third tier consists of query interfaces. BioMart comes with web-based query interfaces (MartView), a stand-alone query interface (MartExplorer) and a command-line interface (MartShell). In addition to this, the BioMart system comes with the MartEditor tool to edit the database configuration. This configuration is stored as XML within the database making meta-data (a description of the database, including the tables and fields) available to third-party applications. Ensembl was one of the first databases to deploy a full-featured BioMart implementation in spring 2005 (Birney et al., 2006; Kasprzyk et al., 2004). Since then, more databases have implemented a BioMart data management system and by now all of the databases described in the previous section have a BioMart implementation.

biomaRt

With the development of the biomaRt package (Durink et al., 2005) we wanted to take advantage of the fast query capabilities in BioMart systems and the growing number of major databases present in this uniform system. This way a large collection of biological data becomes available in R, making it an ideal environment for biological data mining. Queries to the BioMart databases are either performed via web services or by using MySQL.

The biomaRt package depends on the R packages **RCurl** and **XML**, and it is being used on Windows, Linux and OSX. In the sections below we will highlight the functions that are available in the biomaRt package.

Selecting a BioMart database and dataset

A first step when using biomaRt, is to check which BioMart web services are available. The function `listMarts` will display all available BioMart web services. Next we need to select a BioMart database to use, which can be done with the `useMart` function. In the example we will choose to use the Ensembl BioMart web service.

```
> library(biomaRt)
> listMarts()

      name
1    dicty
2  ensembl
3      snp
4     vega
5  uniprot
6      msd
7 wormbase
                         version
1     DICTYBASE (NORTHWESTERN)
2          ENSEMBL 41 (SANGER)
3              SNP 41 (SANGER)
4             VEGA 41 (SANGER)
5 UNIPROT PROTOTYPE 4-5 (EBI)
6       MSD PROTOTYPE 4 (EBI)
7      WORMBASE CURRENT (CSHL)

> ensmart = useMart("ensembl")
```

BioMart databases can contain several datasets. In a next step we look at which datasets are available in the selected BioMart by using the function `listDatasets`.

```
> datasets = listDatasets(ensmart)
> datasets[8:12, ]

                         dataset
8       hsapiens_gene_ensembl
9        ggallus_gene_ensembl
10 tnigroviridis_gene_ensembl
11       mmulatta_gene_ensembl
12       olatipes_gene_ensembl
      version
8      NCBI36
9     WASHUC1
```

```
10 TETRAODON7
11     MMUL_1
12    MEDAKA1
```

```
> ensmart <- useMart("ensembl",
+     dataset = "hsapiens_gene_ensembl")
```

Simple biomaRt functions

In this section we will discuss a set of simple biomaRt functions which are tailored to Ensembl.

Gene annotation

The function `getGene` uses a vector of query identifiers to look up the symbol, description and chromosomal information of the corresponding genes. When using `getGene` with Affymetrix identifiers, we have to specify the chip name by using the `array` argument. When using any other type of identifier the type should be specified with the `type` argument (for example: entrezgene, refseq, unigene,...). The `mart` argument should be used to specify which `Mart` object (which we generated above) to use.

```
> affyids = c("202763_at", "209310_s_at",
+     "207500_at")
> gene = getGene(id = affyids,
+     array = "affy_hg_u133_plus_2",
+     mart = ensmart)
> gene[, -3]
```

```
           ID symbol chromosome
1   202763_at  CASP3          4
2   207500_at  CASP5         11
3 209310_s_at  CASP4         11
   band strand chromosome_start
1 q35.1     -1        185785845
2 q22.3     -1        104370180
3 q22.3     -1        104318810
  chromosome_end ensembl_gene_id
1      185807623 ENSG00000164305
2      104384909 ENSG00000137757
3      104345373 ENSG00000196954
  ensembl_transcript_id
1       ENST00000308394
2       ENST00000260315
3       ENST00000355546
```

```
> entrez = c("673", "7157", "837")
> gene2 = getGene(id = entrez,
+     type = "entrezgene", mart = ensmart)
> gene2[, -3]
```

```
    ID symbol chromosome  band
1  673   BRAF          7   q34
2 7157   TP53         17 p13.1
3  837  CASP4         11 q22.3
  strand chromosome_start
1     -1        140080754
2     -1          7512464
3     -1        104318810
  chromosome_end ensembl_gene_id
1      140271033 ENSG00000157764
2        7531642 ENSG00000141510
3      104345373 ENSG00000196954
  ensembl_transcript_id
1       ENST00000288602
2       ENST00000269305
3       ENST00000355546
```

Note that Ensembl maps all annotation to the transcript level. As such a query might result in multiple apparantly redundant results which can be distinguished by the `ensembl_transcript_id`. Similarly as the `getGene` function, there are simple biomaRt functions to retrieve GO and InterPro protein domain information from Ensembl.

Retrieving sequences

Sequences can be retrieved using the `getSequence` function either starting from chromosomal coordinates or identifiers. The chromosome name can be specified using the `chromosome` argument. The `start` and `end` arguments are used to specify `start` and `end` positions on the chromosome. The `seqType` argument enables one to specify which sequence type should be retrieved (cdna, 5utr, 3utr or protein).

In the example below, we retrieve the 5'UTR[5] sequences of all genes on chromosome 3 between a given start and end position

```
> utr5 = getSequence(chromosome=3,
                     start=185514033,
                     end=185535839,
                     seqType="5utr",
                     mart=ensmart)
> utr5
```

```
              V1 V2            V3
1 ENSG00000114867  3 protein_coding
1 CCGGCTGCGCCTGCGGAGAAGCGGTGGCCGCCGAGCGGGATCTGTGCGGGGAGCCGG
  AAATGGTTGTGGACTACGTCTGTGCGGCTGCGTGGGGCTCGGCCGCGCGGACT....
```

Selecting a set of identifiers with a certain location or GO term association

The `getFeature` function enables us to select a set of features based on chromosomal coordinates or GO identifiers. We can for example select all Affymetrix identifiers on the hgu133plus2 chip for genes located on chromosome 16 between basepair 1100000 and 1250000. `getFeature` takes the `array` or `type` arguments if one wants to retrieve Affymetrix identifiers or other identifiers respectively.

```
> ids = getFeature(array = "affy_hg_u133_plus_2",
+     chromosome = "4", start = "600000",
+     end = "700000", mart = ensmart)
> ids
```

```
  ensembl_transcript_id
1       ENST00000255622
2       ENST00000389796
3       ENST00000389795
4       ENST00000383023
5       ENST00000383023
```

[5]UnTranslated Region

```
6        ENST00000304312
7        ENST00000304312
8        ENST00000360686
9        ENST00000304351
10       ENST00000347950
11       ENST00000322224
12       ENST00000362003
   chromosome_name start_position
1                4         609373
2                4         609373
3                4         609373
4                4         656227
5                4         656227
6                4         656227
7                4         656227
8                4         657369
9                4         665618
10               4         665618
11               4         665618
12               4         689573
   end_position affy_hg_u133_plus_2
1        653896           210304_at
2        653896           210304_at
3        653896           210304_at
4        658127         207335_x_at
5        658127         209492_x_at
6        658127         207335_x_at
7        658127         209492_x_at
8        665816         205145_s_at
9        673230           214269_at
10       673230           214269_at
11       673230           214269_at
12       754428
```

Human variation data

This section briefly shows how to retrieve SNP data. After selecting the SNP BioMart database, the getSNP function can be used to retrieve SNPs that are present in a given genomic region.

```
> snpm = useMart("snp", dataset = "hsapiens_snp")

Checking attributes and filters ... ok

> snp = getSNP(chromosome = 5,
+     start = 327200, end = 327350,
+     mart = snpm)
> snp

      tscid refsnp_id allele
1 TSC1701737 rs2864968    C/G
2 TSC1701738 rs2864969    G/A
3 TSC1790560 rs2902896    A/G
4 TSC1701739 rs2864970    G/A
5 TSC1790561 rs2902897    T/C
6 TSC1790562 rs2902898    A/G
7 TSC1701740 rs2864971    A/G
8 TSC1701741 rs2864972    T/C
  chrom_start chrom_strand
1      327271            1
2      327274            1
3      327283            1
4      327292            1
5      327317            1
6      327326            1
7      327348            1
8      327349            1
```

Homology mapping

BioMart takes advantage of the many species present in Ensembl to do homology mappings. By using two datasets (i.e. two species), we can apply the getHomolog function to map identifiers from one species to the other. In the example below we start from an Affymetrix identifier of a human chip and we want to retrieve the identifiers of the corresponding homolog on a mouse chip.

```
> hs = useMart("ensembl",
                dataset = "hsapiens_gene_ensembl")
> mm = useMart("ensembl",
                dataset = "mmusculus_gene_ensembl")
> hom = getHomolog( id = "1939_at",
                    to.array = "affy_mouse430_2",
                    from.array = "affy_hg_u95av2",
                    from.mart = hs,
                    to.mart = mm )

> hom
                  V1                 V2           V3
1 ENSMUSG00000059552 ENSMUST00000005371 1427739_a_at
2 ENSMUSG00000059552 ENSMUST00000005371 1426538_a_at
```

Generic biomaRt functions

The previous functions were all tailored to Ensembl. In this section we will see biomaRt functions that can be used to retrieve every data field that is made available by any BioMart. Three terms have to be introduced first: filters, attributes and values. A filter defines a restriction on the query. For example you want to restrict the output to all genes located on the X chromosome then the filter chromosome_name can be used with value 'X'. Attributes define the values we are interested in to retrieve. For example we want to retrieve the gene symbols or chromosomal coordinates.

The function listFilters can be used to retrieve all available filters in a dataset.

```
> filters = listFilters(ensmart)
> filters[1:5, ]

            name
1    affy_hc_g110
2   affy_hg_focus
3   affy_hg_u133a
4 affy_hg_u133a_2
5   affy_hg_u133b
           description
1        Affy hc g 110
2   Affy hg focus ID(s)
3   Affy hg u133a ID(s)
4 Affy hg u133a 2 ID(s)
5   Affy hg u133b ID(s)
```

The listAttributes function can be used to see which attributes are available in the selected dataset.

```
> attrib = listAttributes(ensmart)
> attrib[1:5, ]

        name description
1   adf_embl        embl
2     adf_go          go
3   adf_omim        omim
4    adf_pdb         pdb
5 adf_refseq      refseq
```

Once the filters and attributes are known, one can make a biomaRt query using the getBM function. An easy query could be to retrieve the gene symbol, chromosome name and band for a set of affy identifiers.

```
> getBM(attributes = c("affy_hg_u95av2",
+     "hgnc_symbol", "chromosome_name",
+     "band"), filters = "affy_hg_u95av2",
+     values = c("1939_at", "1503_at",
+         "1454_at"), mart = ensmart)

  affy_hg_u95av2 hgnc_symbol
1        1454_at       SMAD3
2        1939_at        TP53
  chromosome_name   band
1              15 q22.33
2              17  p13.1
```

Below we describe some more complicated examples.

Retrieving information on homologs

Within one Ensembl dataset there are attributes providing homology mappings to the other Ensembl species. In the next example, we start from the hsapiens dataset and a list of Entrez Gene identifiers. We can now query chromosomal positions of the corresponding genes in human, zebrafish, mouse and fly.

```
> getBM(attributes = c("hgnc_symbol",
+     "chromosome_name", "start_position",
+     "mouse_chromosome", "mouse_chrom_start",
+     "zebrafish_chromosome",
+     "zebrafish_chrom_start",
+     "drosophila_chromosome",
+     "drosophila_chrom_start"),
+     filter = "entrezgene", values = c("673",
+         "837"), mart = ensmart)

  hgnc_symbol chromosome_name
1        BRAF               7
2       CASP4              11
  start_position mouse_chromosome
1      140080754                6
2      104318810                9
  mouse_chrom_start
1          39543731
2           5308874
  zebrafish_chromosome
1                    4
2                   16
  zebrafish_chrom_start
1               9473158
2              47717138
  drosophila_chromosome
1                     X
2
  drosophila_chrom_start
1                2196282
2                     NA
```

Using more than one filter

The getBM function enables you to use more than one filter. In this case the filter argument should be a vector with the filter names. The values should be a list, where the first element of the list corresponds to the first filter and the second list element to the second filter and so on. The elements of this list are vectors containing the possible values for the corresponding filters. In the example below we want to retrieve the gene symbol, agilent probe identifier, chromosome name and the Ensembl transcript identifier for all the transcripts that are associated with one or more of the specified GO terms and that are located on either chromosome 1, 2, or the Y chromosome.

```
> go = c("GO:0051330", "GO:0000080",
+     "GO:0000114", "GO:0000082",
+     "GO:0000083", "GO:0045023",
+     "GO:0031568", "GO:0031657")
> chrom = c(1, 2, "Y")
> bm = getBM(attributes = c("hgnc_symbol",
+     "agilent_probe", "chromosome_name",
+     "ensembl_transcript_id"),
+     filters = c("go", "chromosome_name"),
+     values = list(go, chrom),
+     mart = ensmart)
> bm[1:2, ]

  hgnc_symbol agilent_probe
1      PPP1CB  A_23_P425579
2      PPP1CB  A_32_P102935
  chromosome_name
1               2
2               2
  ensembl_transcript_id
1       ENST00000379582
2       ENST00000379582
```

Discussion

The Bioconductor package **biomaRt** enables direct access from Bioconductor to BioMart databases such as Ensembl, creating a powerful integration of data analysis and biological databases. As such biomaRt enables one to use the latest annotations available from these databases. A possible drawback is that one needs to be online while performing the queries and downtime of the BioMart server makes queries impossible. A local BioMart database installation would be a solution to prevent this. Currently biomaRt will always use the most recent release of the BioMart databases. However, Ensembl and other BioMart databases archive previous releases. Further development of the package and the BioMart web service, should allow access to these archived database releases. The biomaRt package is designed so that new BioMart databases can be automatically included, once they provide public MySQL access or have an active BioMart web service. The tight integration of large public databases with data analysis in R makes biomaRt a powerful platform for biological data integration and data mining.

Bibliography

J. Ashurst, C. Chen, J. Gilbert *et al.* The Vertebrate Genome Annotation (VEGA) database. *Nucleic Acids Res*, 33:D459–465, 2005.

E. Birney, D. Andrews, M. Caccamo *et al.* Ensembl 2006. *Nucleic Acids Res*, 34:D556–D561, 2006.

S. Durinck, Y. Moreau, A. Kasprzyk *et al.* Biomart and Bioconductor: a powerful link between biological databases and microarray data analysis. *Bioinformatics*, 21:3439–3440, 2005.

The international hapmap consortium. A haplotype map of the human genome. *Nature*, 437:1299–1320, 2005.

A. Kasprzyk, D. Keefe, D. Smedley *et al.* Ensmart: A generic system for fast and flexible access to biological data. *Genome Res*, 14:160–169, 2004.

E. Schwarz, I. Antoshechkin, C. Bastiani *et al.* Wormbase: better software, richer content. *Nucleic Acids Res*, 34:D475–478, 2006.

S. Sherry, M. Ward, M. Kholodov *et al.* dbSNP: the NCBI database of genetic variation. *Nucleic Acids Res*, 29:308–311, 2001.

D. Ware, P. Jaiswal, J. Ni *et al.* Gramene, a resource for comparative grass genomics. *Nucleic Acids Res*, 30(1):103–105, 2002.

C. Wu, R. Apweiler, A. Bairoch *et al.* The Universal Protein Resource (UniProt): an expanding universe of protein information. *Nucleic Acids Res*, 34: D187–191, 2006.

Steffen Durinck
Oncogenomics Section
NIH/NCI
Gaithersburg MD, USA
durincks@mail.nih.gov

Identifying Interesting Genes with siggenes

by Holger Schwender, Andreas Krause and Katja Ickstadt

A common and important task in microarray experiments is the identification of genes whose expression values differ substantially between groups or conditions. Finding such differentially expressed genes requires methods that can deal with multiple testing problems in which thousands or even tens of thousands of hypotheses are tested simultaneously.

Usually, a statistic appropriate for testing if the expression levels are associated with a covariate of interest and the corresponding p-value are computed for each gene. Afterwards, these raw p-values are adjusted for multiplicity such that a Type I error rate is strongly controlled at a pre-specified level of significance. The classical example of such an error rate is the family-wise error rate (FWER), i.e. the probability of at least one false positive. This error rate, however, might be too conservative for a situation in which thousands of hypotheses are tested and several tens of genes should be identified. In the analysis of microarray data, another error rate has hence become very popular: The False Discovery Rate (FDR) which is loosely spoken the expected proportion of false positives among all rejected null hypotheses, i.e. identified genes.

There are, however, other ways to adjust for multiplicity: For example, QQ plots or the Bayesian framework can be employed for this purpose. If the observed test statistics are plotted against the values of the test statistics that would be expected under the null hypothesis most of the points will approximately lie on the diagonal. Those points that differ substantially from this line correspond to genes that are most likely differentially expressed. The Significance Analysis of Microarrays (SAM) proposed by Tusher et al. (2001) can be used to specify what "differ substantially" means. While Tusher et al. (2001) base their analysis on a moderated t statistic, Schwender et al. (2003) compare this approach with a SAM version based on Wilcoxon rank sums.

Efron et al. (2001) use an empirical Bayes analysis (EBAM) to model the distribution of the observed test statistics as a mixture of two components, one for the differentially expressed genes and the other for the not differentially expressed genes. Following their analysis, a gene is called differentially expressed if the corresponding posterior probability is larger than 0.9.

Both SAM and EBAM are implemented in the

Bioconductor package **siggenes**. In this article, we, however, will concentrate on SAM. In the following, we briefly describe the SAM procedure, its implementation in **siggenes** (for more details, see Schwender et al., 2003) and the test statistics already available in this package. Afterwards, we show how you can write your own function for other testing situations. Finally, we will give an example of how `sam` can be applied to gene expression data.

Significance Analysis of Microarrays

In SAM, a statistic d appropriate for testing if there is an association between the expression levels and the covariate of interest is computed for each of the m genes. These observed test scores are sorted and plotted against the scores expected under the null hypothesis, where the expected test scores $\bar{d}_{(i)}, i = 1, \ldots, m$, are computed as follows: If the null distribution is known, then $\bar{d}_{(i)}$ is the $(i - 0.5)/m$ quantile of this null distribution. Otherwise, $\bar{d}_{(i)}$ is assessed by

- generating B permutations of the group labels,
- computing the m test statistics and sorting them for each of the B permutations,
- averaging over the B i^{th}-smallest scores.

Two lines parallel to the diagonal in a distance of Δ are then drawn into this plot called the SAM plot. Any gene that has a d value

- larger than or equal to the d value of the gene, say d_{up}, that corresponds to the left-most point on the right side of the origin that lies above the upper Δ line,
- smaller than or equal to the d value of the gene, say d_{low}, that corresponds to the right-most point on the left side of the origin that lies below the lower Δ line

is called differentially expressed. Afterwards, the FDR is estimated by

- counting how many of the mB permuted test scores are larger than or equal to d_{up} or smaller than and equal to d_{low}, and dividing this number by B,
- dividing this average by the number of identified genes,
- multiplying this ratio by the prior probability that a gene is not differentially expressed (by default, `sam` estimates this probability by the procedure of Storey and Tibshirani, 2003).

This procedure is repeated for several values of Δ and the value of Δ is chosen that provides the best balance between the number of identified genes and the estimated FDR.

The following test statistics can be called in `sam` by setting the argument '`method`' to

`d.stat`: Moderated t and F statistics. The "usual" t or F statistics are computed if the fudge factor '`s0`' is set to zero. (The fudge factor is added to the denominator of the statistics to prevent genes with very low expression levels to become differentially expressed. For details, see Tusher et al., 2001).

`wilc.stat`: Wilcoxon rank sums for one and two class analyses.

`cat.stat`: Pearson's χ^2-statistic for testing categorical data such as SNP (Single Nucleotide Polymorphism) data (Schwender, 2005).

Writing Your Own Test Score Function

It is also possible to write your own function for another testing situation and use this function in `sam`. This function must have as input the two required arguments

'`data`': A matrix or data frame containing the data. Each row of this data set should correspond to one of the m variables, i.e. genes, and each column to one of the n observations.

'`cl`': A vector consisting of the class labels of the observations.

The function can also have additional optional arguments that can be called in `sam`.

The output of this function must be a list consisting of the following objects

'`d`': A numeric vector containing the test scores of the genes.

'`d.bar`': A numeric vector of length `na.exclude(d)` consisting of the sorted test scores expected under the null hypothesis.

'`p.value`': A numeric vector of the same length and order as '`d`' containing the p-values of the genes.

'`vec.false`': A numeric vector of the same length as '`d`' consisting of the one-sided numbers of falsely called genes (for more details, see Schwender et al., 2003, and the help files of `sam`).

'`s`': A numeric vector containing the standard errors of the expression values.

'`s0`': A numeric value specifying the fudge factor.

'`mat.samp`': A $B \times n$ matrix containing the permuted class labels.

'`msg`': A character vector containing messages that are displayed when the `SAM` specific `S4` methods `print` and `summary` are called.

'`fold`': A numeric vector containing the fold changes of the genes. Should be set to `numeric(0)` if another analysis than a two-class analysis is performed.

Assume, e.g., that we would like to perform a SAM analysis with the "usual" *t*-statistic assuming equal group variances and normality. The code of a function `t.stat` for such an analysis is given by

```
t.stat <- function(data, cl){
    require(genefilter) ||
        stop("genefilter required.")
    row.out <- rowttests(data, cl)
    d <- row.out$statistic
    m <- length(na.exclude(d))
    d.bar <- qt(((1:m) - 0.5)/m, row.out$df)
    p.value <- row.out$p.value
    vec.false <- m * p.value/2
    s <- row.out$dm/d
    # dm: differences in group means
    msg <- paste("SAM Two-Class Analysis",
         "Assuming Normality\n\n")
    list(d=-d, d.bar=d.bar, p.value=p.value,
        vec.false=vec.false, s=s, s0=0,
        mat.samp=matrix(numeric(0)),
        msg=msg, fold=numeric(0))
}
```

Please note that in the output of `t.stat` 'd' is set to `-d` since in `rowttests` the mean of group 2 is subtracted from the mean of group 1, whereas in `sam` the difference is taken the other way around.

Now `t.stat` can be used in `sam` by setting `method=t.stat`.

Example: ALL Data

As example we here employ one of the data sets used in Gentleman et al. (2005). The package **ALL** containing this data set can be downloaded by

```
> source(
    "http://www.bioconductor.org/getBioC.R")
> getBioC("ALL")

> library(ALL)
> data(ALL)
```

Even though it is in general not a good idea to filter genes / probe sets prior to a SAM analysis since SAM assumes that most of the genes are not differentially expressed, we here follow the code of Scholtens and von Heydebreck (2005) and filter the genes and select a subset of the samples.

```
> library(genefilter)
> subALL <- filterALL()
```

(The code of `filterALL` can be found in the Appendix.) This leads to an `exprSet` object containing gene expression data of 2,391 probe sets and 79 samples.

Following Scholtens and von Heydebreck (2005) we would like to identify the probe sets whose expression values differ strongly between the samples for which

```
> mol.biol <- pData(subALL)$mol.biol
```

is equal to `"BCR/ABL"` and the samples for which `mol.biol=="NEG"`. Thus, `sam` is applied to this data set by specifying the required arguments '`data`' and '`cl`', where

'`data`' can either be a matrix, a data frame or an `exprSet` object containing the gene expression data,

'`cl`' is a vector containing the class labels of the samples. If '`data`' is an `exprSet` object, then '`cl`' can also be a character string naming the column of `pData(data)` that contains the class labels.

So

```
> library(siggenes)
> clALL <- ifelse(mol.biol=="BCR/ABL", 0, 1)
> dataALL <- exprs(subALL)
> out1 <- sam(dataALL, clALL,
+  var.equal = TRUE, rand = 123456)
```

leads to the same results as

```
> out2 <- sam(subALL, "mol.biol",
+  var.equal = TRUE, rand = 123456)
```

where '`var.equal`' is set to `TRUE` since we here would like to assume that the group variances are equal, and '`rand`' is set to 123456 to make the results of this analysis reproducible.

By default, the number of identified genes and the estimated FDR is computed for ten values of Δ equidistantly spaced between 0.1 and $\max_i |d_{(i)} - \bar{d}_{(i)}|$. The output of our SAM analysis is thus given by

```
> out1

SAM Analysis for the Two-Class Unpaired Case
Assuming Equal Variances

   Delta   p0   False Called     FDR
1    0.1 0.63 1900.61   2075 0.57726
```

```
2    0.7 0.63  125.07     400 0.19706
3    1.4 0.63    3.86      90 0.02703
4    2.0 0.63     0.1      25 0.00252
5    2.7 0.63       0       6       0
6    3.3 0.63       0       4       0
7    4.0 0.63       0       3       0
8    4.6 0.63       0       2       0
9    5.3 0.63       0       2       0
10   5.9 0.63       0       1       0
```

where p0 is the estimated prior probability that a gene is not differentially expressed, False is the number of falsely called genes (see Tusher et al., 2001), Called is the number of identified genes, and FDR = p0 * False / Called is the estimated FDR.

More information, e.g., the value of the fudge factor can be obtained using summary. Both summary and print can also be used to generate the above table for other values of Δ. For example,

```
> print(out1, seq(1.4, 2, 0.1))

SAM Analysis for the Two-Class Unpaired Case
Assuming Equal Variances

  Delta    p0 False Called      FDR
1   1.4 0.63  3.86     90 0.02703
2   1.5 0.63  2.36     77 0.01932
3   1.6 0.63  1.45     64 0.01428
4   1.7 0.63  0.80     54 0.00934
5   1.8 0.63  0.41     45 0.00574
6   1.9 0.63  0.26     36 0.00455
7   2.0 0.63  0.10     25 0.00252
```

Let's say our choice of Δ is 1.5. The SAM plot for this selection shown in Figure 1 is generated by

```
> plot(out1, 1.5, sig.col = c(3,2), pch = 16,
+  pos.stats = 2, cex = 0.6)
```

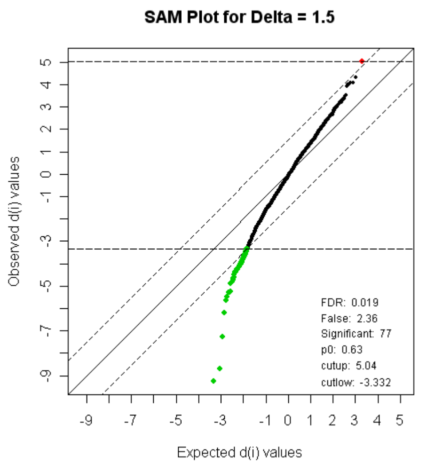


Figure 1: SAM Plot for $\Delta = 1.5$.

where

'sig.col' is a numeric value or vector specifying the color of the identified down- and up-regulated genes,

'pos.stats' indicates where the statistics are shown in the SAM plot,

'cex' specifies the relative size of the plotting symbols of the genes not identified as differentially expressed.

While the relative size of the symbols can be specified separately for the identified and the not identified genes, the symbol itself ('pch') is the same for both types of genes. For all arguments of the SAM specific method plot, see

```
> help.sam(plot)
```

Information about the identified genes such as their d values, the corresponding raw p-values and the q-values (see Storey and Tibshirani, 2003) can be obtained by

```
> summary(out1,1.5)
```

An excerpt from the output of summary is shown in Figure 2. This information can also be stored in a csv file via sam2excel or in an html file using sam2html. If 'data' is an exprSet object or 'chipname' is specified in sam2html, then the html file will also contain the gene symbols and links to public repositories such as Entrez, RefSeq and UniGene. If 'cdfname' is specified, links to the Affymetrix webpages of the identified probe sets will also be available. For example, the html file generated by

```
> sam2html(out1, 1.5, "out1.html", ll = TRUE,
+  cdfname = "HG-U95Av2")
```

is available at http://www.statistik.uni-dortmund.de/de/content/einrichtungen/lehrstuehle/personen/holgers/out1.html.

Finally, we would like to check if method="t.stat" (see previous section) really works.

```
> out3 <- sam(subALL, "mol.biol",
+  method = "t.stat")
> out3

SAM Two-Class Analysis Assuming Normality

   Delta    p0    False Called      FDR
1    0.1 0.63 1832.197   2050  0.56307
2    0.7 0.63   94.609    347  0.17177
3    1.3 0.63    3.555    100  0.02240
4    1.9 0.63    0.044     19  0.00145
5    2.5 0.63 0.000567      6 5.95e-05
6    3.1 0.63 3.33e-05      4 5.24e-06
7    3.7 0.63 2.92e-07      3 6.14e-08
8    4.4 0.63 5.73e-10      2 1.80e-10
9    5.0 0.63 5.73e-10      2 1.80e-10
10   5.6 0.63        0      0        0
```

```
SAM Analysis for the Two-Class Unpaired Case Assuming Equal Variances

s0 = 0

 Number of permutations: 100

 MEAN number of falsely called genes is computed.

 Delta: 1.5
 cutlow: -3.332
 cutup: 5.04
 p0: 0.63
 Significant Genes: 77
 Falsely Called Genes: 2.36
 FDR: 0.0193


Genes called significant (using Delta = 1.5):

    Row d.value  stdev  rawp q.value R.fold       Name
1   134   -9.26 0.1188     0       0  0.457  1636_g_at
2  1787   -8.69 0.1327     0       0  0.442   39730_at
3   133   -7.28 0.1652     0       0  0.420    1635_at
4  1890   -6.18 0.2878     0       0  0.429   40202_at
5  1193   -5.65 0.2388     0       0  0.460   37027_at
```

Figure 2: An excerpt from the output of `summary(out1, 1.5)`.

Since in both analyses we have computed the t statistic assuming equal group variances, the d values in both analyses should be the same:

```
> tmp <- sum(round(out1@d, 8) ==
+  round(out3@d, 8))
> tmp == length(out1@d)
[1] TRUE
```

Summary

The package **siggenes** contains functions for performing both a Significance Analysis of Microarrays (SAM) and an Empirical Bayes Analysis of Microarrays (EBAM). The function `sam` provides not only a set of statistics for standard tests such as the t test and F test but also the possibility to use user-written functions for other testing situations. After identifying a list of genes, not only statistics of these genes such as their test scores and p-values can be obtained but also links to public repositories containing biological information about these genes.

The EBAM functions are currently under revision to provide more user-friendly and less memory-consuming versions of these functions having all the features that the SAM functions already have.

This article will be available as vignette in **siggenes 1.7.1** and later.

Acknowledgements

The work of Katja Ickstadt and Holger Schwender has been supported by the Deutsche Forschungsgemeinschaft, Sonderforschungsbereich 475. The work of Andreas Krause was largely carried out while he was an employee of Novartis Pharma AG, Basel, Switzerland.

Appendix

```
filterALL <- function(){
    pdat <- pData(ALL)
    subset<-intersect(grep("^B",
        as.character(pdat$BT)),
        which(pdat$mol %in% c("BCR/ABL",
        "NEG")))
    eset <- ALL[, subset]
    require(genefilter)
    f1 <- pOverA(0.25, log2(100))
    f2 <- function(x) IQR(x) > 0.5
    selected <- genefilter(eset,
        filterfun(f1, f2))
    esetSub <- eset[selected, ]
    pdat <- pData(esetSub)
    esetSub$mol.biol <-
        as.character(esetSub$mol.biol)
    esetSub
}
```

Bibliography

B. Efron, R. Tibshirani, J. Storey and V. Tusher. Empirical Bayes Analysis of a Microarray Experiment. *Journal of the American Statistical Association*, 96, 1151–1160, 2001.

R. Gentleman, V. J. Carey, W. Huber, R. A. Irizarry and S. Dudoit, editors. *Bioinformatics and Computational Biology Solutions Using R and Bioconductor*. Springer, New York, 2005.

D. Scholtens and A. von Heydebreck. Analysis of Differential Gene Expression Studies. In: R. Gentleman, V. J. Carey, W. Huber, R. A. Irizarry, S. Dudoit, editors. *Bioinformatics and Computational Biology Solutions Using R and Bioconductor*. Springer, New York, 229–248, 2005.

H. Schwender. Modifying Microarray Analysis Methods for Categorical Data – SAM and PAM for SNPs. In: C. Weihs, W. Gaul, editors. *Classification – The Ubiquitous Challenge*. Springer, Heidelberg, 370–377, 2005.

H. Schwender, A. Krause and K. Ickstadt. Comparison of the Empirical Bayes and the Significance Analysis of Microarrays. *Technical Report*, SFB 475, University of Dortmund, Germany, 2003. URL http://www.sfb475.uni-dortmund.de/berichte/tr44-03.pdf.

J. D. Storey and R. Tibshirani. Statistical Significance for Genomewide Studies. *Proceedings of the National Academy of Sciences*, 100, 9440–9445, 2003.

V. Tusher, R. Tibshirani and G. Chu. Significance Analysis of Microarrays Applied to the Ionizing Radiation Response. *Proceedings of the National Academy of Science*, 98, 5116–5121, 2001.

Holger Schwender, Katja Ickstadt
Department of Statistics, SFB 475
University of Dortmund, Germany
holger.schwender@udo.edu,
ickstadt@statistik.uni-dortmund.de

Andreas Krause
Pharsight Corporation
Mountain View, CA, USA
akrause@pharsight.com

Reverse Engineering Genetic Networks using the GeneNet Package

by Juliane Schäfer, Rainer Opgen-Rhein, and Korbinian Strimmer

GeneNet is a package for analyzing high-dimensional (time series) data obtained from high-throughput functional genomics assays, such as expression microarrays or metabolic profiling. Specifically, **GeneNet** allows to infer large-scale gene association networks. These are graphical Gaussian models (GGMs) that represent multivariate dependencies in biomolecular networks by means of partial correlation. Therefore, the output of an analysis conducted by **GeneNet** is a graph where each gene corresponds to a node and the edges included in the graph portray direct dependencies between them.

GeneNet implements a specific learning algorithm that allows to estimate GGMs from small sample high-dimensional data that is both computationally as well as statistically efficient. This approach relies on analytic shrinkage estimation of covariance and (partial) correlation matrices and on model selection using (local) false discovery rate multiple testing. Hence, **GeneNet** includes a computational algorithm that decides which edges are to be included in the final network, in dependence of the *relative* values of the pairwise partial correlations.

In a recent comparative survey (Werhli et al., 2006) the **GeneNet** procedure was found to recover the topology of gene regulatory networks with similar accuracy as computationally much more demanding methods such as dynamical Bayesian networks (Friedman, 2004).

We note that the approach implemented in **GeneNet** should be regarded as an exploratory approach that may help to identify interesting genes (such as "hubs") or clusters of genes that are functionally related or co-regulated, rather than that it yields the precise network of mechanistic interactions. Therefore, the resulting network topologies need be interpreted and validated in the light of biological background information, ideally accompanied by further integrative analysis employing data from different levels of the cellular system.

Prerequisites

GeneNet is available from the CRAN repository and from the webpage http://strimmerlab.org/software/genenet/. It requires prior installation of four further R packages also found on CRAN: **corpcor**, **longitudinal**, **fdrtool**, and **locfdr** (Efron, 2004).

For installation of the required packages simply enter at the R prompt:

```
> install.packages( c("corpcor",
    "longitudinal", "fdrtool",
    "locfdr", "GeneNet") )
```

Preparation of Input Data

The input data must be arranged in a matrix where columns correspond to genes and where rows correspond to the individual measurements. Note that the data must already be properly preprocessed, i.e. in the case of expression data calibrated and normalized.

In the following we describe an example for inferring the gene association network among 102 genes from a microarray data set on the microorganism *Escherichia coli* with observations at 9 time points (Schmidt-Heck et al., 2004). These example data are part of **GeneNet**:

```
> library("GeneNet")
> data(ecoli)
> dim(ecoli)
[1]   9 102
```

Shrinkage Estimators of Covariance and (Partial) Correlation

The first step in the inference of a graphical Gaussian model is the reliable estimation of the partial correlation matrix:

```
> inferred.pcor <- ggm.estimate.pcor(ecoli)
> dim(inferred.pcor)
[1] 102 102
```

For this purpose, the function `ggm.estimate.pcor` offers an interface to a shrinkage estimator of partial correlation implemented in the **corpcor** package that is statistically efficient and can be used for analyzing small sample data. By default, the option `method="static"` is selected, which employs the function `pcor.shrink`. Standard graphical modeling theory (e.g. Whittaker, 1990) shows that the matrix of partial correlations $\tilde{\mathbf{P}} = (\tilde{\rho}_{ij})$ is related to the inverse of the covariance matrix $\boldsymbol{\Sigma}$. This relationship leads to the straightforward estimator

$$\tilde{r}_{ij} = -\hat{\omega}_{ij} / \sqrt{\hat{\omega}_{ii}\hat{\omega}_{jj}}, \qquad (1)$$

where

$$\hat{\boldsymbol{\Omega}} = (\hat{\omega}_{ij}) = \hat{\boldsymbol{\Sigma}}^{-1}. \qquad (2)$$

In Equation 2, it is absolutely crucial that the covariance is estimated accurately, and that $\hat{\boldsymbol{\Sigma}}$ is well conditioned – otherwise the above formulae will result in a rather poor estimate of partial correlation (cf. Schäfer and Strimmer, 2005a). For this purpose, the `pcor.shrink` function uses an analytic shrinkage estimator of the correlation matrix developed in Schäfer and Strimmer (2005b). This linearly combines the unrestricted sample correlation with a suitable correlation target in a weighted average. Selecting this target requires some diligence: specifically, we choose to shrink the empirical correlations $\mathbf{R} = (r_{ij})$ towards the identity matrix, while empirical variances are left intact. In this case the analytically determined shrinkage intensity is

$$\lambda^{\star} = \frac{\sum_{i \neq j} \mathrm{var}(r_{ij})}{\sum_{i \neq j} r_{ij}^2}. \qquad (3)$$

The resulting shrinkage estimate exhibits a number of favorable properties. For instance, it is much more efficient, always positive definite, and well conditioned. It is inexpensive to compute and does not require any tuning parameters, as the analytically derived optimal shrinkage intensity is estimated from the data. Moreover, there are no assumptions about the underlying distributions of the individual estimates, except for the existence of the first two moments. These properties carry over to derived quantities, such as partial correlations. Furthermore, the resulting estimates are in a form that allows for fast computation of their inverse using the Woodbury matrix identity.

Note that the function `ggm.estimate.pcor` also allows the specification of a `protect` argument, with default value `protect=0`. This imposes limited translation (Efron and Morris, 1972) onto the specified fraction of entries of the estimated shrinkage correlation matrix, thereby protecting those components against overshrinkage (see also Opgen-Rhein and Strimmer, 2006c).

Taking Time Series Aspects Into Account

Standard Gaussian graphical models assume i.i.d. data whereas in practice many expression data sets result from time course experiments. One possibility to generalize the above procedure correspondingly is to employ dynamic (partial) correlation (Opgen-Rhein and Strimmer, 2006a). This is available in the function `ggm.estimate.pcor` by specifying the option `method="dynamic"`, which in turn relies on the **longitudinal** package for computation.

The key difference between dynamical and i.i.d. correlation is that the former takes into account the time that has elapsed between two subsequent measurements. In particular, dynamical correlation allows for unequally spaced time points as often encountered in genomic studies. All small sample learning procedures (shrinkage) developed for i.i.d.

correlation also carry over to dynamical correlation (Opgen-Rhein and Strimmer, 2006b).

Network Search and Model Selection

The second crucial part of gene association network inference is model selection, i.e. assigning statistical significance to the edges in the GGM network:

```
> test.results <-
             ggm.test.edges(inferred.pcor)
> dim(test.results)
[1] 5151    6
```

For this purpose a mixture model,

$$f(\tilde{r}) = \eta_0 f_0(\tilde{r}; \kappa) + (1 - \eta_0) f_A(\tilde{r}) \,, \tag{4}$$

is fitted to the observed partial correlation coefficients $\tilde{r}$ using the subroutine `cor.fit.mixture`. f_0 is the distribution under the null hypothesis of vanishing partial correlation, η_0 is the (unknown) proportion of "null edges", and f_A the distribution of observed partial correlations assigned to actually existing edges. The latter is assumed to be an arbitrary nonparametric distribution that vanishes for values near zero. This allows for κ, η_0, and even f_A to be determined from the data – see Efron (2004) for an algorithm.

Subsequently, two-sided p-values corresponding to the null hypothesis of zero partial correlation are computed for each potential edge using the function `cor0.test`. Large-scale simultaneous testing is then conducted by obtaining q-values via the function `fdr.control` with the specified value of η_0 taken into account. `fdr.control` uses the algorithms described in Benjamini and Hochberg (1995) and Storey (2002). An alternative to the q-value approach is to use the empirical Bayes local false discovery rate (fdr) statistic (Efron, 2004). This fits naturally with the above mixture model setup, and in addition takes account of the dependencies among the estimated partial correlation coefficients. The posterior probability that a specific edge exists given $\tilde{r}$ equals

$$\mathrm{IP}(\text{non-null edge}|\tilde{r}) = 1 - \mathrm{fdr}(\tilde{r}) = 1 - \frac{\eta_0 f_0(\tilde{r}; \kappa)}{f(\tilde{r})} . \tag{5}$$

Following Efron (2005), we typically consider an edge to be "significant" if its local fdr is smaller than 0.2, or equivalently, if the probability of an edge to be "present" is larger that 0.8:

```
> signif <- test.results$prob > 0.80
> sum(signif)
[1] 66
> test.results[signif,]
```

Network Visualization

The network plotting functions in **GeneNet** rely extensively on the infrastructure offered by the **graph** and **Rgraphviz** packages (cf. contribution of Seth Falcon in this R News issue).

First, a `graph` object must be generated containing all significant edges:

```
> node.labels <- colnames(ecoli)
> gr <- ggm.make.graph(
     test.results[signif,],
     node.labels)
> gr
A graphNEL graph with undirected edges
Number of Nodes = 102
Number of Edges = 66
```

Subsequently, the resulting object can be inspected by running the command

```
> show.edge.weights(gr)
```

Finally, the gene network topology of the graphical Gaussian model can be visualized using the function `ggm.plot.graph`:

```
> ggm.plot.graph(gr,
      show.edge.labels=FALSE,
      layoutType="fdp")
```

The plot resulting from the analysis of the `ecoli` data is shown in Figure 1. For `show.edge.labels=TRUE` the partial correlation coefficients will be printed as edge labels. Note that on some platforms (e.g. Windows) the default `layoutType="fdp"` may not yet be available. In this case an alternative variant such as `"layoutType=neato"` needs to be specified.

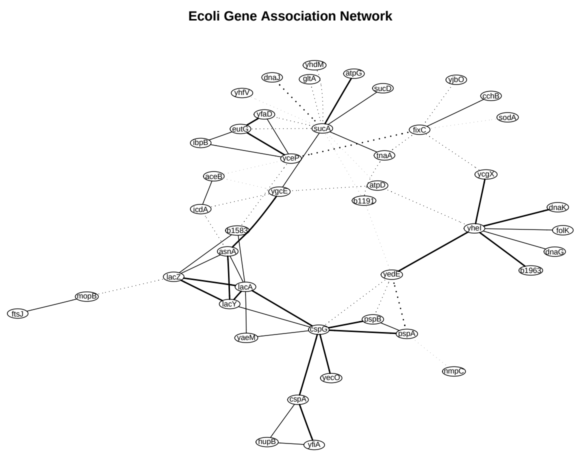


Figure 1: Sparse graphical Gaussian model for 102 genes inferred from an *E. coli* microarray data set with 9 data points. Full and dotted lines indicate positive and negative partial correlation, respectively.

Release History of GeneNet and Example Scripts

The package **GeneNet** emerged from a reorganization of the (now obsolete) package **GeneTS**. This was split into the **GeneNet** part dealing with gene network reconstruction, and the package **GeneCycle** for cell cycle and periodicity analysis (Wichert et al., 2004; Ahdesmäki et al., 2005).

On the home page of **GeneNet** we collect example scripts in order to guide users of **GeneNet** when conducting their own analyses. Currently, this includes the above *E. coli* data but for instance also a network analysis of *A. thaliana* diurnal cycle genes. We welcome further contributions from the biological community.

Bibliography

M. Ahdesmäki, H. Lähdesmäki, R. Pearson, H. Huttunen, and O. Yli-Harja. Robust detection of periodic time series measured from biological systems. *BMC Bioinformatics*, 6:117, 2005.

Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *J. R. Statist. Soc. B*, 57:289–300, 1995.

B. Efron. Large-scale simultaneous hypothesis testing: the choice of a null hypothesis. *J. Am. Statist. Assoc.*, 99:96–104, 2004.

B. Efron. Local false discovery rates. Preprint, Dept. of Statistics, Stanford University, 2005.

B. Efron and C. N. Morris. Limiting the risk of Bayes and empirical Bayes estimators – part II: The empirical Bayes case. *J. Am. Statist. Assoc.*, 67:130–139, 1972.

N. Friedman. Inferring cellular networks using probabilistic graphical models. *Science*, 303:799–805, 2004.

R. Opgen-Rhein and K. Strimmer. Inferring gene dependency networks from genomic longitudinal data: a functional data approach. *REVSTAT*, 4:53–65, 2006a.

R. Opgen-Rhein and K. Strimmer. Using regularized dynamic correlation to infer gene dependency networks from time-series microarray data. In *Proceedings of the 4th International Workshop on Computational Systems Biology (WCSB 2006)*, 12-13 June 2006, Tampere, volume 4, pages 73–76, 2006b.

R. Opgen-Rhein and K. Strimmer. Accurate ranking of differentially expressed genes by a distribution-free shrinkage approach. In review.

J. Schäfer and K. Strimmer. An empirical Bayes approach to inferring large-scale gene association networks. *Bioinformatics*, 21:754–764, 2005a.

J. Schäfer and K. Strimmer. A shrinkage approach to large-scale covariance matrix estimation and implications for functional genomics. *Statist. Appl. Genet. Mol. Biol.*, 4(1):Article 32, 2005b.

W. Schmidt-Heck, R. Guthke, S. Toepfer, H. Reischer, K. Duerrschmid, and K. Bayer. Reverse engineering of the stress response during expression of a recombinant protein. In *Proceedings of the EUNITE symposium, 10-12 June 2004, Aachen, Germany*, pages 407–412, 2004. Verlag Mainz.

J. D. Storey. A direct approach to false discovery rates. *J. R. Statist. Soc. B*, 64:479–498, 2002.

A. Werhli, M. Grzegorczyk, and D. Husmeier. Comparative evaluation of reverse engineering gene regulatory networks with relevance networks, graphical Gaussian models and Bayesian networks. *Bioinformatics*, 22:2523–2531, 2006.

J. Whittaker. *Graphical Models in Applied Multivariate Statistics*. Wiley, New York, 1990.

S. Wichert, K. Fokianos, and K. Strimmer. Identifying periodically expressed transcripts in microarray time series data. *Bioinformatics*, 20:5–20, 2004.

Juliane Schäfer, ETH Zurich, Switzerland
Rainer Opgen-Rhein, University of Munich, Germany
Korbinian Strimmer, University of Munich, Germany
juliane.schaefer@stat.math.ethz.ch
opgen-rhein@stat.uni-muenchen.de
korbinian.strimmer@lmu.de

A Multivariate Approach to Integrating Datasets using made4 and ade4

by Aedín C. Culhane and Jean Thioulouse

The public microarray repositories, ArrayExpress and the GeneExpression Omnibus (GEO), now contain over 100,000 microarray gene expression profiles (Table 1). This is a considerable data resource.

However the average number of arrays per study is only between 30 and 40 (Table 1). Given that the number of features (genes) on microarrays now exceeds 50,000, this presents a considerable dimensionality problem. Low case to feature ratio is likely to remain an issue, as cost and availability of biomaterial, such as biopsy tissue, are often limiting. As a result, meta-analysis or merging data from multiple studies is attractive.

Table 1. Public Microarray Databases[a]

Database	Arrays	Studies
ArrayExpress[b]	44,602	1,487
GEO[c]	87,073	2,353

[a]Statistics: ArrayExpress (June 2006), GEO (5 July 2006)
[b]`http://www.ebi.ac.uk/arrayexpress/`
[c]`http://www.ncbi.nlm.nih.gov/geo/`

Unfortunately, matching of variables (gene probes) from different microarray technologies is challenging. Numerous microarray platforms have been developed and a number of studies have reported disappointingly low correlations between different technologies. Matching of probes by their DNA sequence reduces cross-platform inconsistency (Carter et al., 2005), and functions to perform sequence matching are available in the Bioconductor package **matchprobes**. EnsEMBL alignments of DNA probe sequences to the human and other genomes can be retrieved using the package **biomaRt**. However even the performance of matched probes may vary across platforms. These differences may be due to real biological effects where probes on different platforms detect different splice variants or homologues of a gene.

A different approach is simply to examine genes or cases with covariant trends across matched datasets. We have described the application of co-inertia analysis (CIA) for visualization and analysis of such trends across microarray datasets (Culhane et al., 2003). Functions to perform these analyses are provided in the Bioconductor package **made4** (Culhane et al., 2005). **made4** is an extension to **ade4** (Thioulouse et al., 1997; Chessel et al., 2004), an extensive R package for multivariate analysis of ecological data.

Our multivariate approach for cross-platform analysis of microarray data may be easily applied to heterogeneous datasets. Increasingly, microarray experiments are performed in parallel with proteomics, metabolomics or other high throughput array technologies. Typically the identity of peaks or spots in proteomics or metabolomics data is unknown. Therefore mapping probes, spots or peaks across datasets is not possible. In analysis of these data, we are simply exploring features (peaks or spots) that have similar trends across datasets and are correlated with a covariate of interest. CIA is suitable for such an analysis.

We will describe the application of CIA to cross-platform visualization of microarray data and other functions in **made4** and **ade4** for multivariate analysis of biological datasets.

Co-inertia analysis

CIA is a multivariate analysis method that describes the relationship between two data tables (Dray et al., 2003). It can be used on quantitative, qualitative or distance matrices. Classical methods, like principal component (PCA) or correspondence analysis (CA), aim at summarizing a table by searching orthogonal axes on which the projection of the sampling points (cases) have the highest possible variance. This characteristic ensures that the associated graphs (factor maps) best represent the initial data (see Figure 1).

To extract information common to two tables, canonical analysis (CANCOR, Gittins, 1985) searches successive pairs of axes (one for each table) with a maximum correlation. The problem is that this analysis may lead to axes with high correlation, but low percentages of explained variance. This means that it will be difficult to give a biological interpretation to these axes. To overcome this difficulty, CIA searches for pairs of axes with maximum covariance (instead of correlation). This ensures that CIA axes will have both a high correlation and also good percentages of explained variance for each table. Computations are based on the cross table between the variables of the two tables. The importance of each axis is given by the percentage of total co-inertia, which is similar to the percentage of explained variance for canonical axes.

CIA has been successfully applied to visualization of cross-platform relationships between microarray datasets (Culhane et al., 2003). CIA is an attractive approach as it can be applied to data where the number of variables (genes) far exceeds the number of cases, as seen in microarray data. Given data with such low sample size, CANCOR cannot be used and canonical correspondence analysis (Ter Braak, 1986) is reduced to a plain CA (Dray et al., 2003).

Monte-Carlo tests can be used to check the significance of the relationship between the two tables. The method consists of performing many random permutation of the cases (arrays), followed by the recomputation of the total co-inertia. By comparing the total co-inertia obtained in the normal analysis with the co-inertias obtained after randomization, one can estimate the probability of the observed relationship between the tables.

PCA or CA of microarray data

PCA and CA are well suited to exploratory analysis of microarray data, and complement popular clustering approaches. While clustering investigates pairwise distances among objects highlighting fine relationships, PCA and CA examine the variance of the whole dataset highlighting general trends and gradients (reviewed by Brazma and Culhane, 2005). To perform a PCA or CA on a microarray dataset using **made4**, use the function **ord**.

To illustrate we apply CA to a microarray gene expression profiling study of 4 childhood tumors (NB, BL-NHL, EWS, RMS; Khan et al., 2001). A subset of these expression data (`khan$train`, 306 genes x 64 cases), a factor describing the class of each case (`khan$train.classes`, length=64) and a data frame of gene annotation are available in dataset `khan` in **made4**.

```
library(made4)
data(khan)
dataset = khan$train
fac =     khan$train.classes
geneSym = khan$annotation$Symbol

results.coa <- ord(dataset, type="coa")
par(mfrow= c(1,2))
plotarrays(results.coa, classvec=fac)
plotgenes(results.coa, genelabels= geneSym)
```

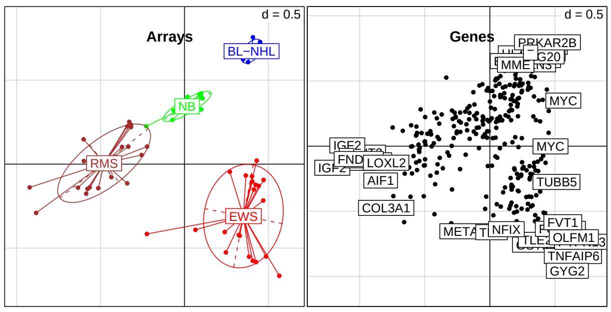


Figure 1: CA of a 306 gene subset of Khan dataset (Khan et al., 2001). **A)** Plot of arrays **B)** Plot of genes. The further a gene and case are projected in the same direction from the origin, the stronger association between that gene and case (gene is upregulated in that array sample).

Cross-platform analysis using CIA

To perform CIA, objects in the dataset must be "matchable". For example, where multiple studies are performed on the same samples, CIA can detect co-varying patterns across datasets. If the cases are matched, there is no constraint to match the variables (genes) and the number of variables in each dataset may differ.

In Example 2, we examine a panel of 60 cell lines from the National Cancer Institute (NCI60) that have been subjected to gene expression profiling using Affymetrix (Staunton et al., 2001) and spotted cDNA (Ross et al., 2000) arrays. We apply **cia** to subsets of these 2 datasets which are available in **made4**.

```
data(NCI60)
names(NCI60)
  [1] "Ross"    "Affy"    "classes" "Annot"
fac = NCI60$classes[,2]
results.cia = cia(NCI60$Affy, NCI60$Ross)
par(mfrow=c(1,2))
plotarrays(results.cia, clabel=0)
plotarrays(results.cia, clabel=0,
           classvec=fac)
```

In Figure 2, matched cases are joined by a line. If two cases (arrays) have similar profiles, they will be projected close together. Therefore, the shorter the length of connecting line the greater the correlation. In Figure 2B, one green case is represented by a long line, indicating a large cross-platform difference between the two expression profiles for this cell line. This may suggest a quality issue in one dataset.

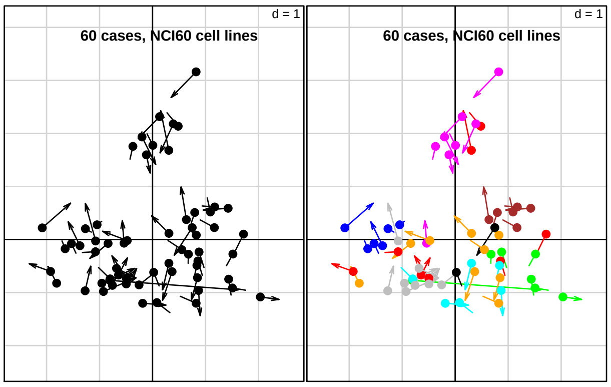


Figure 2: CIA of NCI60 datasets Affy (closed circles) and Ross (arrows). **B).** Same as A) but cases are colored by class (cancer cell line phenotype). Further details and interpretation in Culhane (2003).

CIA using "matched" genes

Equally CIA could be performed on variables (genes). Visualization of matched genes across platforms is often useful when there is a one:many match of gene probes. On older arrays, a gene was generally only represented by one probe, but on recent microarrays a gene maybe represented by 5 or more probes (or probesets). In Example 3, we examine microarray studies of acute lymphoblastic leukemia (ALL) using older hu6800 (Golub et al., 1999) or more recent u95av2 (Chiaretti et al., 2004) Affymetrix arrays. These datasets are available in Bioconductor packages **golubEsets** and **ALL**.

```
library(affy)
library(ALL)
data(ALL)
ALL.fac <- substring(ALL$BT,1,1)
library(golubEsets)
data(Golub_Train)
golub <- Golub_Train[,1:27]  #ALL data
golub.data <- exprs(golub)
#footnote 1
golub.data[] <- as.double(golub.data)
golub.fac <-golub$T.B.cell
```

We performed a *t*-test on the Golub data, using **rowttests** in the **genefilter** package, to select genes which were significantly (P<0.001) associated with T-cell or B-cell ALL.

```
library(genefilter)
ttests <- rowttests(golub.data,golub.fac)
nsignf <- sum(ttests$p.val < 0.001)
topGeneInd <- order(ttests$p.val)[1:nsignf]
ttests.signf <-
    rownames(golub.data)[topGeneInd]
```

There were 109 significant gene probes on the hu6800 arrays, which were matched to genes on the u95av2 using **biomaRt**.

```
library(biomaRt)
mart <- useMart("ensembl", mysql=TRUE)
mart <- useDataset("hsapiens_gene_ensembl",
                   mart)
pRef <- getBM(attributes="affy_hg_u95av2",
              values=ttests.signf,
              filters="affy_hugenefl",
              mart=mart)
anyNA <- function(x) any(is.na(x))
pRef <- pRef[!apply(pRef, 1, anyNA), ]
dupSet <- function(x, a)
    subset(x, a %in% a[duplicated(a)])
pMany <- dupSet(pRef, pRef$affy_hugenefl)
```

Of the 109 hu6800 probesets, 96 mapped to 133 u95av2 probesets. Therefore 29 hu6800 probesets mapped to more than 1 u95av2 probesets. These 29:66 "one to many" matches were examined using **cia**.

[1]This processing is only required with golubEset and is not normal processing of ExpressionSet datasets

```
hu6800set <-
    exprs(golub[pMany$affy_hugenefl, ])
u95av2set <-
    exprs(ALL[pMany$affy_hg_u95av2, ])

cia.out <- cia(t(hu6800set), t(u95av2set))
coordVar1 <- cia.out$coinertia$co
coordVar2 <- cia.out$coinertia$li
par(mfrow=c(2, 2))
plotarrays(cia.out, sub="Genes")
plotarrays(cia.out, clabel=0,
           classvec=pMany$affy_hugenefl)
plotarrays(coordVar1, classvec=golub.fac)
plotarrays(coordVar2, classvec=ALL.fac)
```

In Figure 3 we observe that the probesets selected using the older hu6800 platform, do appear to discriminate B and T cells expression profiles on u95av2 arrays. However it appears that only a few gene probes contribute a significant amount of variance across both datasets.

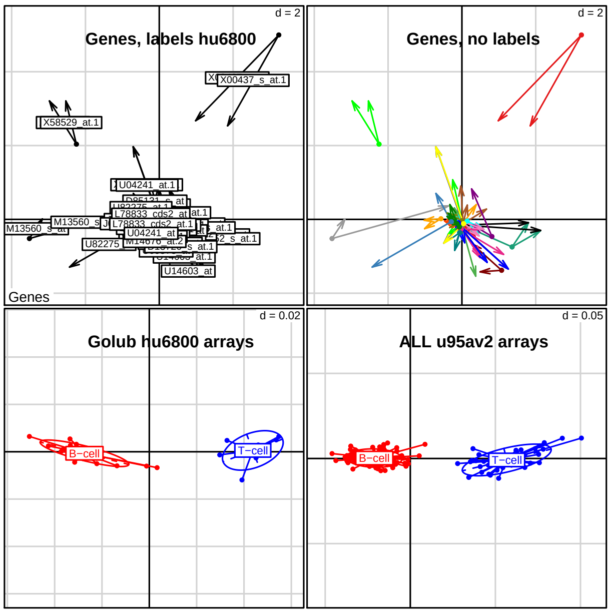


Figure 3: CIA of a set of genes in golub and ALL datasets. Projection of probesets **A)** with and **B)** without hu6800 probe labels, and arrays **C),D)**.

One nice feature of this analysis is that one:many probeset matches are clearly visualised. For example, M13560_s_at, X58529_at, X00437_s_at have 2 matches on the hgu95av2 platform. We can see that only one M13560_s_at u95av2 probeset matches has a high loading on the B-cell end of axis 1 (horizontal), indicating that the expression of only one of these probesets agrees with the older hu6800 array.

Combining microarray data with gene sequence information

We have also used CIA to integrate microarray data with counts of motif occurrences in gene promoters, to discover which promoter motifs are most associated with the main patterns of gene expression in a dataset (Jeffery et al., 2006). We have also extended this approach using between group analysis (Culhane et al., 2002), a supervised method where groupings of arrays or tissues of a-priori interest are contrasted with the rest. Using between group CIA, we identify gene motifs (or other gene features) that are most associated with a gene expression classifier (Jeffery et al., 2006).

Using **ade4** codon usage may be investigated using internal correspondence analysis, a variant of between groups and within groups analyses (Lobry and Chessel, 2003). CIA has also been applied to study the relationships between amino-acid physico-chemical properties and protein composition (Thioulouse and Lobry, 1995). These analyses maybe facilitated using the **seqinr** package (Charif et al., 2005). **seqinr** is an interface between R and the ACNUC (Gouy et al., 1985) sequence retrieval system for nucleotide and protein sequence databases such as GenBank, EMBL and SWISS-PROT.

Although we have only described analysis of 2 tables, many other multivariate analysis methods are available in **ade4**, which are easily extended using the duality diagram (class **dudi**) (Chessel et al., 2004). There are several functions for analysis of three-way or multiple tables (class **ktab** class, and functions **sepan**, **statis**, **pta**, **mcoa**, **mfa**, **foucart**). Distance matrices can be integrated in this framework through principal coordinates analysis (**dudi.pco** function), and the **kdist** class in the case of k distance matrices measured on the same individuals.

GUIs : ade4TkGUI, Rweb

The **made4** package was created to ease the use of multivariate data analysis of microarray gene-expression data. Indeed, it has two main advantages: it is an interface between the **ade4** package and Bioconductor data objects and classes, and it provides wrapper functions to simplify the use of multivariate analysis functions implemented in **ade4**.

Another approach to the simplification is the use of a graphical user interface (GUI). A new package (**ade4TkGUI**) has been developed using the **tcltk** package to provide a GUI to **ade4**. This GUI has two special features. The first one is a centralized graphical display of **ade4 dudi** objects). The second one is a dynamic view of factor maps, allowing exploration of sample and variable sets by way of zooming, panning, and searching on labels. An Rweb interface to **seqinr** and **ade4** multivariate analysis is also available (`http://pbil.univ-lyon1.fr/Rweb/Rweb.general.html`).

Summary

The Bioconductor package **made4** facilitates multivariate analysis of microarray data, and builds on extensive experience of multivariate data analysis in ecology. Multivariate data analysis methods provide many useful tools to extract meaningful biological information from these large data sets. Sometimes, these methods are overlooked because they are thought to be complicated and subject to barely met application hypotheses. This is partly true in the framework of the Gaussian approximation model of multivariate analysis. But the geometric model (for example Le Roux and Rouanet, 2004), and the duality diagram (Holmes, 2006) lift most of these assumptions.

Bibliography

A. Brazma and A.C. Culhane. Algorithms for gene expression analysis. In M.J. Dunn, L.B. Jorde, P.F.R. Little, and S. Subramaniam, editors, *Encyclopedia of Genetics, Genomics, Proteomics and Bioinformatics*. John Wiley and Sons, London, 2005.

S.L. Carter, A.C. Eklund, B.H. Mecham *et al.* Redefinition of affymetrix probe sets by sequence overlap with cdna microarray probes reduces cross-platform inconsistencies in cancer-associated gene expression measurements. *BMC Bioinformatics*, 6: 107, 2005.

D. Charif, J. Thioulouse, J.R. Lobry *et al.* Online synonymous codon usage analyses with the ade4 and seqinr packages. *Bioinformatics*, 21(4):545–7, 2005.

D. Chessel, A. Dufour, and J. Thioulouse. The ADE4 package – I: One-table methods. *RNews*, 4(1):5–10, June 2004.

S. Chiaretti, X. Li, R. Gentleman *et al.* Gene expression profile of adult T-cell acute lymphocytic leukemia identifies distinct subsets of patients with different response to therapy and survival. *Blood*, 103(7):2771–8, 2004.

A.C. Culhane, G. Perrière, E. C. Considine *et al.* Between-group analysis of microarray data. *Bioinformatics*, 18(12):1600–8, 2002.

A.C. Culhane, G. Perrière, and D.G. Higgins. Cross-platform comparison and visualisation of gene expression data using co-inertia analysis. *BMC Bioinformatics*, 4:59, 2003.

A.C. Culhane, J. Thioulouse, G. Perrière *et al.* Made4: an R package for multivariate analysis of gene expression data. *Bioinformatics*, 21(11):2789–90, 2005.

S. Dray, D. Chessel, and J. Thioulouse. Co-inertia analysis and the linking of ecological tables. *Ecology*, 84:3078–3089, 2003.

R. Gittins. *Canonical analysis, a review with applications in ecology. Vol.12 of Biomathematics.* Springer- Verlag, Berlin, 1985.

T.R. Golub, D.K. Slonim, P. Tamayo *et al.* Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286:531–537, 1999.

M. Gouy, C. Gautier, M. Attimonelli *et al.* ACNUC–a portable retrieval system for nucleic acid sequence databases: logical and physical designs and usage. *Comput Appl Biosci*, 1(3):167–72, 1985.

S. Holmes. Multivariate analysis: The french way. In D. Nolan and T. Speed, editors, *Festschrift for David Freedman*. IMS, Beachwood, OH, 2006.

I.B. Jeffery, S.F. Madden, P.A. McGettigan *et al.* Integrating transcription factor binding site information with gene expression datasets. *BMC Bioinformatics*, 7:359, 2006.

J. Khan, J.S. Wei, M. Ringner *et al.* Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nat Med*, 7(6):673–9, 2001.

B. Le Roux and H. Rouanet. *Geometric Data Analysis.* Kluwer Academic Publishers, Dordrecht, 2004.

J. R. Lobry and D. Chessel. Internal correspondence analysis of codon and amino-acid usage in thermophilic bacteria. *J Appl Genet*, 44(2):235–61, 2003.

D.T. Ross, U. Scherf, M.B. Eisen *et al.* Systematic variation in gene expression patterns in human cancer cell lines. *Nat Genet*, 24(3):227–35, 2000.

J. E. Staunton, D.K. Slonim, H.A. Coller *et al.* Chemosensitivity prediction by transcriptional profiling. *Proc Natl Acad Sci USA*, 98(19):10787–92, 2001.

C. Ter Braak. Canonical correspondence analysis: a new eigenvector technique for multivariate direct gradient analysis. *Ecology*, 69:1167–1179, 1986.

J. Thioulouse and J. Lobry. Co-inertia analysis of amino-acid physico-chemical properties and protein composition with the ade package. *Comput Appl Biosci*, 11(3):321–9, 1995.

J. Thioulouse, D. Chessel, S. Dolèdec *et al.* ADE-4: a multivariate analysis and graphical display software. *Statistics and Computing*, 7(1):75–83, 1997.

Aedín C. Culhane
Department of Biostatistics and Computational Biology, Dana-Farber Cancer Institute & Department of Biostatistics, Harvard School of Public Health, Boston, MA, USA.
`aedin@jimmy.harvard.edu`

Jean Thioulouse
Biométrie et Biologie Evolutive,
CNRS & Université Lyon 1, France.
`jthioulouse@biomserv.univ-lyon1.fr`

Using amap and ctc Packages for Huge Clustering

by Antoine Lucas and Sylvain Jasson

Introduction

Huge clustering is often required in the field of DNA microarray (DeRisi et al., 1997) analysis. A new use of clustering results appears with presentation and exploration software like *TreeView* (Eisen et al., 1998).

DNA microarray is the most appropriate method for high throughput gene studies, allowing expression evaluation of vast gene numbers in different cells types or conditions. From a technical point of view, microarray analysis first needs image processing (for example *Imagene* (`http://www.biodiscovery.com`), *BZScan* (Lopez et al., 2004) or *ScanAlyze* (Eisen et al., 1998)) that gives large tables of data, followed by statistical processing including data normalization.

A main goal of microarray analysis is to detect coregulated genes presenting similar expression profiles, which can be achieved by various classification techniques. In this area, hierarchical clustering is of special interest as it allows multi-scale cluster visualization.

Some R extensions provide efficient clustering tools (mainly: **stats** and **cluster**; Struyf et al., 1997). The packages **amap** and **ctc** aim to complete the set of clustering tools for R with:

- Additional features to standard clustering functions.
- A novel PCA method, robust to extreme values.

- Fast and optimized and parallelized algorithms, which drastically reduce time and memory requirements so that any computer is able to cluster large data sets.

- The possibility of an external visualization with software such as *Treeview* (Eisen et al., 1998) and *Freeview* (`http://magix.fri.uni-lj.si/freeview`), as shown in Figure 1. This visualization software provides convenient cluster exploration and browsing to find relevant information in large cluster trees.

Description

Amap and **ctc** packages are complementary. The former implementa all statistical algorithms and the latter is used for all interactions with other software such as the Eisen suite and makes it possible to launch *Xcluster* (`http://genetics.stanford.edu/~sherlock/cluster.html`) software within R.

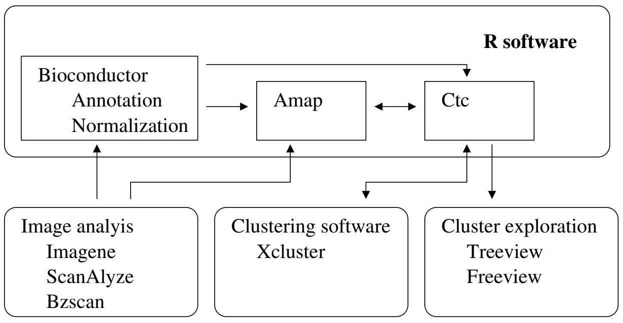
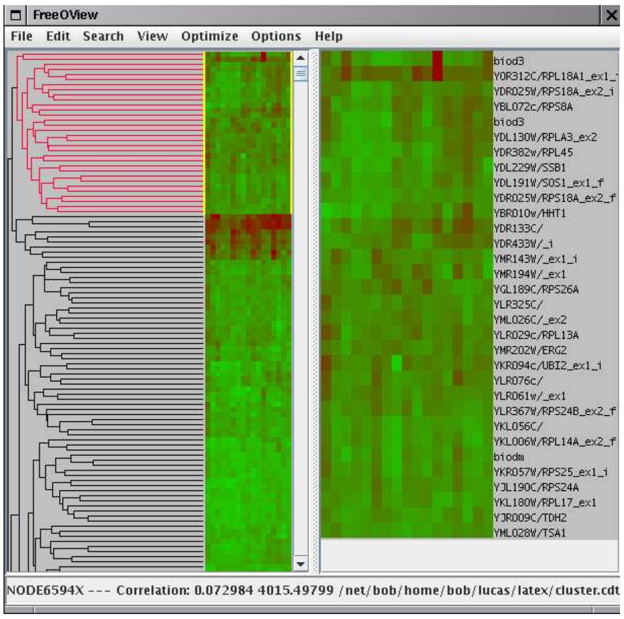


Figure 1: **Amap** and **ctc** usage for microarray analysis.

The **amap** package includes standard hierarchical clustering and k-means analysis. The novel features of **amap** are a larger selection of distances set like Pearson or Spearman (rank-based metric) adapted to microarray data and a better hierarchical clustering implementation (see the benchmark section).

Clustering can be pre-processed by a principal component analysis, as it projects data into an orthogonal vector space, which avoids counting correlated variables twice. With this analysis, a few extreme values may strongly affect the main components. As the high throughput implies a fully automated data acquisition and therefore outliers generation, we implement robust statistic tools including a principal component analysis. The main idea of such tools is to minimize the isolated points affected by lowering their relative weight. This is a recent method described by Caussinus et al. (2003).



```
NODE6594X --- Correlation: 0.072984 4015.49799 /net/bob/home/bob/lucas/latex/cluster.cdt
```

Figure 2: Results window with *Freeview* displaying a sub-cluster (red part of the tree).

The **ctc** package has tools to export cluster trees to visualization software that can explore trees at any scale to find the suitable magnification depending on the data specificity and the biological expertise as shown on Figure 2.

The **ctc** package also makes it convenient to use *Xcluster* software within R. *Xcluster* performs clustering with a very small memory allocation (it does not compute the whole distance matrix).

We propose the possibility to import results from Eisen *Cluster* software to perform post clustering processing with R. Other conversion functions are designed to dialog between R and Eisen software suite.

Benchmark

We compare time and memory use with other main implementations of standard hierarchical clustering: average link for agglomeration method and Euclidean distance (see Figure 3).

It appears that *Xcluster* has less memory needs (less than 100 MB when others methods use more than 1.5 GB) since the algorithm used does not compute exhaustive distance while agglomerating clusters. When using the complete distance matrix, memory use is $O(n^2)$ Hierarchical clustering from **amap**, **stats** or **cluster** packages returns the same tree but **cluster** includes a post processing that reworks the tree display. The **amap** implementation of `hcluster` or `hclusterpar` functions are significantly faster and allow us to pass the limit of 15000 genes on a recent server in less than half an hour.

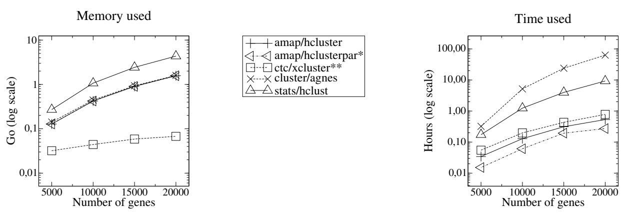


Figure 3: Benchmark of R package. Sample: simulated data, 5000 to 20000 genes under 200 conditions, using average link and Euclidean distance for clustering. Computer: dual Xeon processor server, with 4 GB RAM and 20 GB swap. System command "time" provides time and memory usage. R version 2.0.1
* `hclusterpar` is a parallelized version of `hcluster`, uses all CPU.
** `Xcluster` uses a slightly simplified algorithm.

Web application

As many end-users use graphical and intuitive interfaces, we propose a way to skip the R command line austerity while using a web interface. We provide files 'amap.php' and 'ctc.php' as part of the packages, which produce both form and CGI script, with any standard *apache* and *php* server.

A more sophisticated web application can be tested and downloaded at url: `http://bioinfo.genopole-toulouse.prd.fr/microarray`.

Methods and implementation

The **amap** core library is implemented in C. The package runs on Linux, Windows, and Mac OS X. Multi-threading and parallelization are disabled on Windows. Both **amap** and **ctc** use the free and open source license GPL.

The **amap** package is hosted on a sourceforge like project manager at `http://mulcyber.toulouse.inra.fr/projects/amap` by Inra that provides a cvs repository and a bug tracker.

The **amap** package is also available on CRAN, and the **ctc** package is available on Bioconductor.

Acknowledgments

We wish to thank Laurent Gautier and Steffen Möller for their contributions.

A. Lucas was supported by the *FNS Réseau des Génopôles* contract number A1457.

Bibliography

H. Caussinus, M. Fekri, S. Hakam, and A. Ruiz-Gazen. A monitoring display of multivariate outliers. *Computational Statistics & Data Analysis*, 44: 237–252, October 2003.

J. DeRisi, V. Iyer, and P. Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 278(5338):680–6, 1997.

M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. USA*, 95: 14863–14868, 1998.

F. Lopez, J. Rougemont, B. Loriod *et al.* Feature extraction and signal processing for nylon DNA microarrays. *BMC Genomics*, 5(1):38, Jun 2004. Evaluation Studies.

A. Struyf, M. Hubert, and P. Rousseeuw. Integrating robust clustering techniques in S-PLUS. *Computational Statistics and Data Analysis*, 26:17–37, Nov 1997.

Antoine Lucas
`antoinelucas@gmail.com`

Sylvain Jasson
Unité de Biométrie et Intelligence Artificielle, INRA, Castanet Tolosan, France
`sylvain.jasson@toulouse.inra.fr`

Model-based Microarray Image Analysis

by Chris Fraley and Adrian E. Raftery

DNA microtechnology has enabled biologists to simultaneously monitor the expression levels of thousands of genes or portions of genes under multiple experimental conditions. Many microarray platforms exist; what they all have in common is that the gene expression data is obtained via image analysis of the array segments or spots corresponding to the individual experiments.

A common method for making DNA arrays consists of printing the single-stranded DNA representing the genes on a solid substrate using a robotic spotting device. The arrayed DNA spots are then mixed and hybridized with the cDNA extracted from the experimental and control samples. In the two-color array, these samples are treated before hybridization with both Cy3 (green) and Cy5 (red) fluorescent dyes. After hybridization, the arrays are scanned at the corresponding wavelengths separately to obtain the images corresponding to the two channels. The fluorescence measurements are used to determine the relative abundance of mRNA or DNA in the samples.

The quantification of the amount of fluorescence from the hybridized sample can be affected by a variety of defects that occur during both the manufacturing and processing of the arrays, such as perturbations of spot positions, irregular spot shapes, holes in spots, unequal distribution of DNA probe within spots, variable background, and artifacts such as dust and precipitates. Ideally these events should be automatically recognized in the image analysis, and the estimated intensities adjusted to take account of them.

Li et al. (2005) proposed a method for segmenting microarray image blocks, along with a robust model-based method for estimating foreground and background intensities. Peaks and valleys are first located in the image signal with a sliding window to automatically separate the microarray blocks into regions containing the individual spots. Model-based clustering (McLachlan and Peel 2000, Fraley and Raftery 2002) is then applied to the (univariate) sum of the intensities of the two channels measuring the red and green signals for each spot region to provide an initial segmentation. Models are fit for up to three groups (background, foreground, uncertain), and the number of groups present is then determined via the Bayesian Information Criterion (BIC). Whenever there is more than one group, the segmentation is postprocessed in order to remove artifacts. This is done by reclassifying connected components in the brightest group that are below a certain threshold in size as unknown. The procedure is described in Figure 1.

1. Automatic gridding.
2. Model-based clustering for ≤ 3 groups.
3. Foreground / background determination:
    - If there is more than one group, threshold connected components. The foreground is taken to be the group of highest mean intensity and the background the group of lowest mean intensity.
    - If there is only one group, it is assumed that no foreground signal is detected.

Figure 1: Basic Procedure for Model-based Segmentation of Microarray Blocks.

This approach combines the strengths of histogram-based and spatial methods. It deals effectively with inner holes and artifacts. It also provides a formal inferential basis for deciding when no foreground signal is present in a spot. The method has been shown to compare favorably with other methods on experimental microarray data with replicates (Li et al. 2005).

The method is implemented in the Bioconductor package `spotSegmentation`, which consists of two basic functions:

`spotgrid`: determines spot locations in blocks within microarray slides

`spotseg`: determines foreground and background signals within individual spots

The `spotseg` function uses the R package `mclust` (Fraley and Raftery, 1999, 2003, 2006) for model-based clustering. Other life-sciences applications of model-based clustering include grouping coexpressed genes (Yeung et al. 2001), *in vivo* MRI of patients with brain tumors (Wehrens et al. 2002), and contrast-enhanced MRI for breast tumors (Forbes et al. 2004).

The `spotSegmentation` functions will be illustrated on the first block from the first microarray slide image from van't Wout et al. (2003), available as a dataset in Bioconductor under the name `HIVcDNAvantWout03`. The encoded image data from the two channels for this block are provided as datasets `hiv1raw` and `hiv2raw`, and can be obtained via the `data` command.

```
> data(hiv1raw)
> data(hiv2raw)
```

The data come from a supplementary website `http://ubik.microbiol.washington.edu/HIV/array1/supplemental.htm`, where they are encoded for compact storage. We have chosen to provide these data as given there, so that the following transformation is needed in order to extract the intensities:

```
> dataTrans <- function(x,A=4.7154240E-05)
  matrix((256*256-1-x)^2*A,nrow=450,ncol=1000)
> hiv1 <- dataTrans(hiv1raw)
> hiv2 <- dataTrans(hiv2raw)
```

Note that this transformation is specific to this data; in general stored image data must be converted as needed to image intensities. Figure 2 shows the image data for the two channels in reverse gray scale. These plots can be obtained with the `spotSegmentation` package using the following commands:

```
> plotBlockImage(sqrt(hiv1))
> plotBlockImage(sqrt(hiv2))
```

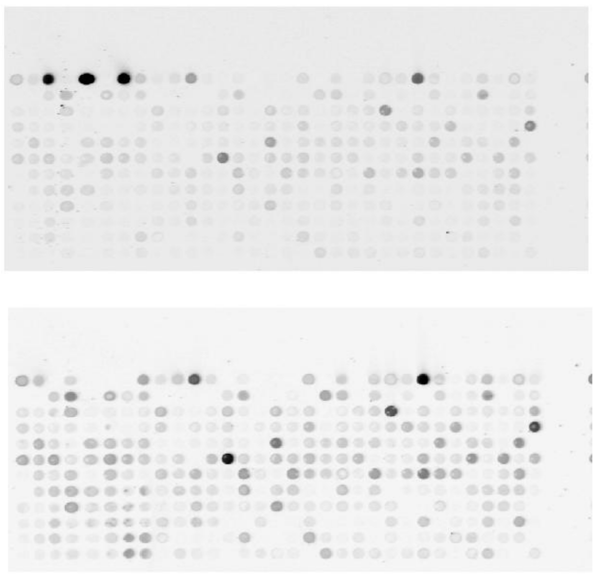


Figure 2: Reverse gray-scale plots of image intensities from channel 1 (Cy3 green) and channel 2 (Cy5 red) of the first block from the first slide of HIV data from the Bioconductor dataset HIVcDNAvantWout03.

The function spotgrid can be used to divide the microarray image block into a grid separating the individual spots.

```
> hivGrid <- spotgrid( hiv1, hiv2,
          rows = 12, cols = 32, show = TRUE)

> hivGrid
$rowcut


 [1] 105 138 163 189 219 244 271 297 326 354
     379 407 438

$colcut
 [1]  11  41  66  94 126 163 192 222 250 279
      307 338 364 392 419 445 474 501 531 558
      587 614 641 671 697 727 754 782 808 836
      862 889 922
```

Here we have used the knowlege that there are 12 rows and 32 columns in a block of the microarray image. The show option allows display of the image, shown in the top pannel of Figure 3.
The individual spots can now be segmented using the function spotseg. The following segments all spots in the block:

```
hivSeg <- spotseg( hiv1, hiv2,
           hivGrid$rowcut, hivGrid$colcut)

plot(hivSeg)
```

The corresponding plot is shown in the bottom panel in Figure 3.

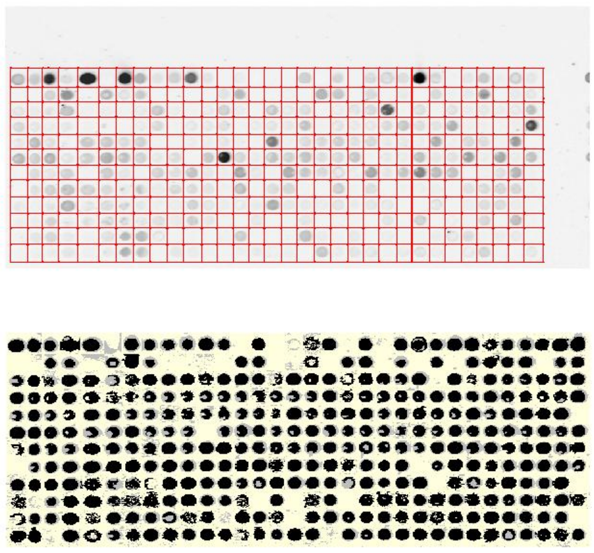


Figure 3: Above: The grid delimiting microarray spots determined by function spotgrid, superimposed on the sum of the intensities for the two channels for the Bioconductor dataset HIVcDNAvantWout03. Below: The segmented spots produced by spotseg. The color scheme is as follows: *black* denotes the spots, *yellow* denotes background, *gray* denotes pixels of uncertain classification.

It is possible to process a subset of the regions in the grid using the arguments R for grid (as opposed to pixel) row location of the spot and C for grid column location. The show option in spotseg can be used to display details for each spot as it is classified. When more than one spot is processed, the graphics command par(ask = TRUE) should be set so that the displays can be stepped through. The following is an example of the segmenting and display of an individual splot.

```
hivSeg <- spotseg( hiv1, hiv2,
           hivGrid$rowcut, hivGrid$colcut
           R = 1, C = 4, show = TRUE)
```

The resulting display is shown in Figure 4.
Mean and median pixel intensities for the foreground and background for each channel and each spot can be recovered through the summary function applied to the output of spotseg. For example, the following extracts the summary intensities for the spot shown in Figure 4.

```
> hivSumry <- summary(hivSeg)

> hivSumry$channel1$foreground$mean[1,4]
[1] 1475.053

> hivSumry$channel2$background$median[1,4]
[1] 249.0123
```

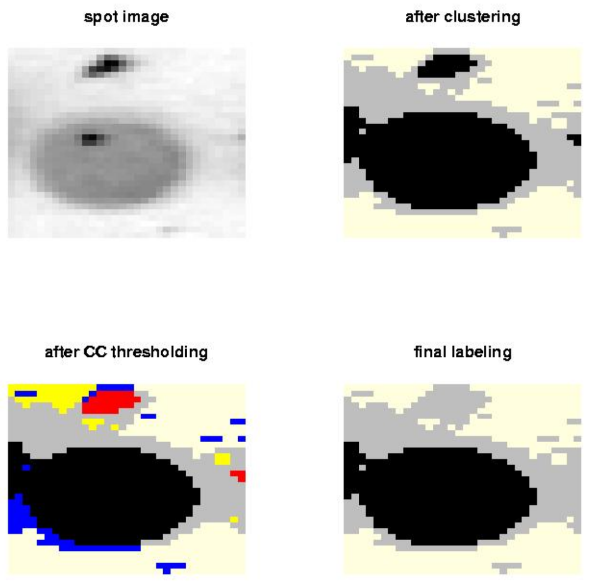


Figure 4: `spotseg` processing of the 1,4 section of the gridded HIV image data. Clockwise from top left: gray-scale image, labeled image after model-based clustering (light yellow: lowest intensity; black: highest intensity), clustered image with connected components less than threshold in size labeled (bright yellow, blue, red denote components below threshold in size for the light yellow, gray, and black groups, respectively), final labeling.

Summary

The Bioconductor package `spotSegmentation` provides functionality for automatic gridding of microarray blocks given the number of rows and columns of spots. It also provides functionality for determining foreground and background of spots in microarray images via model-based clustering. This approach deals effectively with inner holes and artifacts, as well as providing a formal inferential basis for deciding when no foreground signal is present in a spot.

Bibliography

F. Forbes, N. Peyrard, C. Fraley, D. Georgian-Smith, D. Goldhaber, and A. Raftery. Model-based region-of-interest selection in dynamic breast MRI. *Journal of Computer Assisted Tomography*, 30:675–687, 2006.

C. Fraley and A. E. Raftery. MCLUST: Software for model-based cluster analysis. *Journal of Classification*, 16:297–306, 1999.

C. Fraley and A. E. Raftery. Model-based clustering, discriminant analysis and density estimation. *Journal of the American Statistical Association*, 97:611–631, 2002.

C. Fraley and A. E. Raftery. Enhanced software for model-based clustering, density estimation, and discriminant analysis: MCLUST. *Journal of Classification*, 20:263–286, 2003.

Q. Li, C. Fraley, R. E. Bumgarner, K. Y. Yeung, and A. E. Raftery. Donuts, scratches, and blanks: Robust model-based segmentation of microarray images. *Bioinformatics*, 21:2875–2882, 2005.

G. J. McLachlan and D. Peel. *Finite Mixture Models*. Wiley, 2000.

R. Wehrens, A. W. Simonetti and L. M. .C. Buydens, Mixture modeling of medical magnetic resonance data. *Journal of Chemometrics*, 16:274-282, 2002.

A. B. van't Wout, G. K. Lehrman, S. A. Mikeeva, G. C. O'Keefe, M. G. Katze, R. E. Bumgarner, G. K. Geiss, and J. I. Mullins. Cellular gene expression upon human immunodeficiency type 1 infection of CD4(+)-T-cell lines. *Journal of Virology*, 77(2):1392–1402, January 2003.

K. Y. Yeung, C. Fraley, A. Murua, A. E. Raftery, and W. L. Ruzzo. Model-based clustering and data transformation for gene expression data. *Bioinformatics 17*, 977–987, 2001.

C. Fraley and A. E. Raftery. MCLUST version 3 for R: Normal mixture modeling and model-based clustering. Technical Report No. 504, University of Washington, September, 2006.

Chris Fraley, Adrian Raftery
Department of Statistics, Box 354322
University of Washington
Seattle, WA 98195-4322 USA
`fraley@stat.washington.edu`
`raftery@stat.washington.edu`

Sample Size Estimation for Microarray Experiments Using the ssize Package

by Gregory R. Warnes

Abstract

mRNA Expression Microarray technology is widely applied in biomedical and pharmaceutical research. The huge number of mRNA concentrations estimated for each sample make it difficult to apply traditional sample size calculation techniques and has left most practitioners to rely on rule-of-thumb techniques. In this paper, we briefly describe and then demonstrate a simple method for performing and visualizing sample size calculations for microarray experiments as implemented in the **ssize** R package.

Note

This document is a simplified version of the manuscript

> Warnes, G. R., Liu, P. (2006) Sample Size Estimation for Microarray Experiments, Technical Report, Department of Biostatistics and Computational Biology, University of Rochester.

which has been available as a pre-publication manuscript since 2004. Please refer to that document for a detailed discussion of the sample size estimation method and an evaluation of its performance.

Introduction

High-throughput microarray experiments allow the measurement of expression levels for tens of thousands of genes simultaneously. These experiments have been used in many disciplines of biological research, including neuroscience (Mandel *et al.*, 2003), pharmacogenomic research, genetic disease and cancer diagnosis (Heller, 2002). As a tool for estimating gene expression and single nucleotide polymorphism (SNP) genotyping, microarrays produce huge amounts of data which can providing important new insights.

Microarray experiments are rather costly in terms of materials (RNA sample, reagents, chip, etc), laboratory manpower, and data analysis effort. It is critical, therefore, to perform proper experimental design, including sample size estimation, before carrying out these experiments. Since tens of thousands of variables (gene expressions) may be measured on each individual chip, it is essential appropriately take into account multiple testing and dependency among variables when calculating sample size.

Method

Overview

Warnes and Liu (2006) provide a simple method for computing sample size for microarray experiments, and reports on a series of simulations demonstrating its performance. Surprisingly, despite its simplicity, the method performs exceptionally well even for data with very high correlation between measurements.

The key component of this method is the generation of a cumulative plot of the proportion of genes achieving a desired power as a function of sample size, based on simple gene-by-gene calculations. While this mechanism can be used to select a sample size numerically based on pre-specified conditions, its real utility is as a visual tool for understanding the trade off between sample size and power. In our consulting work, this latter use as a visual tool has been exceptionally valuable in helping scientific clients to make the difficult trade offs between experiment cost and statistical power.

Assumptions

In the current implementation, we assume that a microarray experiment is set up to compare gene expressions between one treatment group and one control group. We further assume that microarray data has been normalized and transformed so that the data for each gene is sufficiently close to a normal distribution that a standard 2-sample pooled-variance t-test will reliably detect differentially expressed genes. The tested hypothesis for each gene is:

$$H_0 : \mu_T = \mu_C$$

versus

$$H_1 : \mu_T \neq \mu_C$$

where μ_T and μ_C are means of gene expressions for treatment and control group respectively.

Computations

The proposed procedure to estimate sample size is:

1. Estimate standard deviation (σ) for each gene based on *control samples* from existing studies performed on the same biological system. (While samples from the study to be performed are not, of course, generally available, control samples from other studies using the same biological system are often readily available.)

2. Specify values for

   (a) minimum effect size, Δ, (log of fold-change for log-transformed data)

   (b) maximum family-wise type I error rate, α

   (c) desired power, $1-\beta$.

3. Calculate the per-test Type I error rate necessary to control the family-wise error rate (FWER) using the Bonferroni correction:

$$\alpha_G = \frac{\alpha}{G} \tag{1}$$

   where G is the number of genes on the microarray chip.

4. Compute sample size separately for each gene according to the standard formula for the two-sample t-test with pooled variance:

$$\begin{aligned} &1-\beta \\ &= 1 - T_{n_1+n_2-2}\left(t_{\alpha_G/2,n_1+n_2-2} | \frac{\Delta}{\sigma\sqrt{\frac{1}{n_1}+\frac{1}{n_2}}}\right) \\ &+ T_{n_1+n_2-2}\left(-t_{\alpha_G/2,n_1+n_2-2} | \frac{\Delta}{\sigma\sqrt{\frac{1}{n_1}+\frac{1}{n_2}}}\right) \end{aligned} \tag{2}$$

   where $\mathrm{T}_d(\bullet|\theta)$ is the cumulative distribution function for non-central t-distribution with d degree of freedom and the non-centrality parameter θ.

5. Summarize the necessary sample size across all genes using a cumulative plot of required sample size verses power. An example of such a plot is given in Figure 3 for which we assume equal sample size for the two groups, $n = n_1 = n_2$.

On the cumulative plot, for a point with x coordinate n, the y coordinate is the proportion of genes which require a sample size smaller than or equal to n, or equivalently the proportion of genes with power greater than or equal to the specified power $(1-\beta)$ at sample size n. This plot allows users to visualize the relationship between power for all genes and required sample size in a single display. A sample size can thus be selected for a proposed microarray experiment based on user-defined criterion. For the plot in Figure 3, for example, requiring 80% of genes to achieve the 80% power yields a sample size of 10.

Similar plots can be generated by fixing the sample size and varying one of the other parameters, namely, significance level (α), power $(1-\beta)$, or minimum effect size (Δ). Two such plots are shown in Figures 2 and 4.

Functions

There are three pairs of functions available in the **ssize** package.

```
pow(sd, n, delta, sig.level,
  alpha.correct = "Bonferroni")
power.plot(x, xlab = "Power",
  ylab = "Proportion of Genes with"
         " Power >= x",
  marks = c(0.7, 0.8, 0.9), ...)

ssize(sd, delta, sig.level, power,
  alpha.correct = "Bonferroni")
ssize.plot(x,
  xlab = "Sample Size (per group)",
  ylab = "Proportion of Genes Needing Sample"
         " Size <= n",
  marks = c(2, 3, 4, 5, 6, 8, 10, 20), ...)

delta(sd, n, power, sig.level,
  alpha.correct = "Bonferroni")
delta.plot (x, xlab = "Fold Change",
  ylab = "Proportion of Genes with "
         "Power >= 80\% at\\n"
 "Fold Change=delta",
  marks = c(1.5, 2, 2.5, 3, 4, 6, 10), ...)
```

pow, power.plot compute and display a cumulative plot of the fraction of genes achieving a specified power for a fixed sample size (`n`), effect size (`delta`), and significance level (`sig.level`).

ssize,ssize.plot compute and display a cumulative plot of the fraction of genes for which a specified sample size is sufficient to achieve a specified power (`power`), effect size (`delta`), and significance level (`sig.level`).

delta,delta.plot compute and display a cumulative plot of the fraction of genes which can achieve a specified power (`power`), for a specified sample size (`n`), and significance level (`sig.level`) for a range of effect sizes.

Example

First, we need to load the **ssize** package:

```
> library("ssize")
> library("xtable")
> library("gdata")
```

The **ssize** package provides an example data set containing gene expression values for smooth muscle cells from a control group of untreated healthy volunteers processed using Affymetrix U95 chips and normalized per the Robust Multi-array Average (RMA) method of Irizarry *et al.* (2003).

```
> data("exp.sd")
> exp.sd <- exp.sd[1:1000]
```

This data was calculated via:

```
library("affy")
load("probeset_data.Rda")
expression.values <- exprs(probeset.data)
covariate.data <- pData(probeset.data)
controls <- expression.values[,
            covariate.data$GROUP=="Control"]
exp.sd <- apply(controls, 1, sd)
```

Let's see what the distribution looks like:

```
> par(cex = 2)
> xlab <- c("Standard Deviation",
+     "(for data on the log scale)")
> hist(exp.sd, n = 40, col = "cyan",
+     border = "blue", main = "",
+     xlab = xlab, log = "x")
> dens <- density(exp.sd)
> scaled.y <- dens$y * par("usr")[4]/
+             max(dens$y)
> lines(dens$x, scaled.y,
+     col = "red", lwd = 2)
```

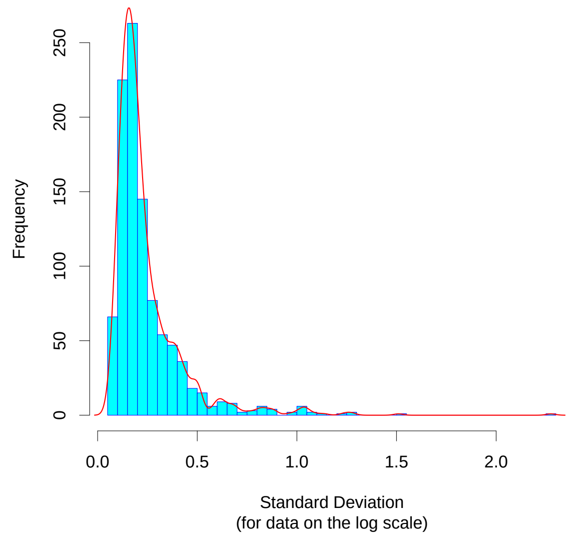


Figure 1: Standard deviations for of logged example data

As is often the case, this distribution is extremely right skewed, even though the standard deviations were computed on the $\log_2$ scale.

So, now lets see the functions in action. First, define the parameter values we will be investigating:

```
> n <- 6
> fold.change <- 2
> power <- 0.8
> sig.level <- 0.05
```

Now, the functions provided by the **ssize** package can be used to address several questions:

1. What is the necessary per-group sample size for 80% power when $\delta = 1.0$, and $\alpha = 0.05$?

```
> all.size <- ssize(sd = exp.sd,
+     delta = log2(fold.change),
+     sig.level = sig.level,
+     power = power)
> par(cex = 1.3)
> ssize.plot(all.size, lwd = 2,
+     col = "magenta", xlim = c(1,
+         20))
> xmax <- par("usr")[2] -
+     1
> ymin <- par("usr")[3] +
+     0.05
> legend(x = xmax, y = ymin,
+     legend = strsplit(paste("fold change=",
+         fold.change, ",",
+         "alpha=", sig.level,
+         ",", "power=",
+         power, ",", "# genes=",
+         nobs(exp.sd), sep = ""),
+         ",")[[1]], xjust = 1,
+     yjust = 0, cex = 0.9)
> title("Sample Size to Detect 2-Fold Change")
```

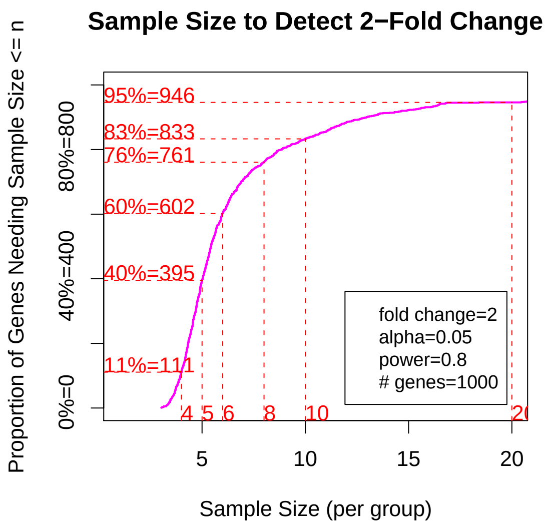


Figure 2: Sample size required to detect a 2-fold treatment effect.

This plot illustrates that a sample size of 10 is required to ensure that at least 80% of genes have power greater than 80%. It also shows that a sample size of 6 is sufficient if only 60% of the genes need to achieve 80% power.

2. What is the power for 6 patients per group with $\delta = 1.0$, and $\alpha = 0.05$?

```
> all.power <- pow(sd = exp.sd,
+     n = n, delta = log2(fold.change),
+     sig.level = sig.level)
> power.plot(all.power, lwd = 2,
+     col = "blue")
```

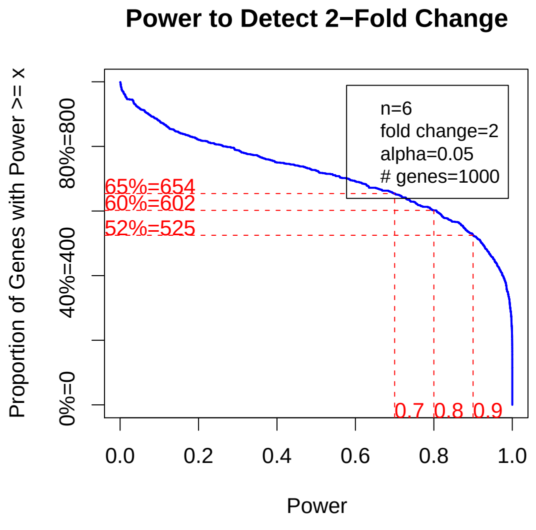


Figure 3: Effect of Sample Size on Power

This plot shows that only 60% of genes achieve at 80% power at this sample size and significance level.

3. How large does a fold-change need to be for 80% of genes to achieve 80% power for an experiment for $n = 6$ patients per group and $\alpha = 0.05$?

```
> all.delta <- delta(sd = exp.sd,
+     power = power, n = n,
+     sig.level = sig.level)
> delta.plot(all.delta, lwd = 2,
+   col = "magenta",
+   xlim = c(1, 10),
+   ylab = paste("Proportion of Genes with ",
+               "Power >= 80% \n",
+               "at Fold Change of delta"))
```

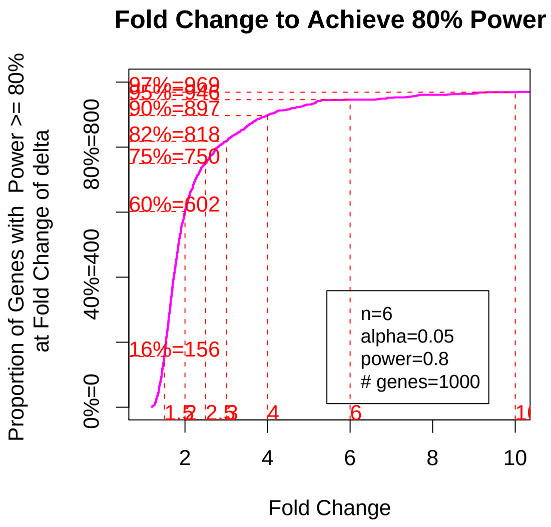


Figure 4: Given sample size, this plot allows visualization of the fraction of genes achieving the specified power for different fold changes.

This plot shows that for a fold change of 2.0, only 60% of genes achieve 80% power, while a fold change of 3.0 will be detected with 80% power for 80% of genes.

Modifications

While the **ssize** package has been implemented using the simple 2-sample pooled t-test, you can easily modify the code for other circumstances. Simply replace the call to `power.t.test` in each of the functions `pow`,`ssize`,`delta` with the appropriate computation for the desired experimental design.

Future Work

Peng Liu is currently developing methods and code for substituting False Discovery Rate for the Bonferroni multiple comparison adjustment.

Contributions

Contributions and discussion are welcome.

Acknowledgment

This work was supported by Pfizer Global Research and Development.

Bibliography

Benjamini, Y. and Hochberg, Y. (1995) Controlling the false discovery rate: a practical and powerful approach to multiple testing, *Journal of Royal Statistical Society B*, **57:1**, 289-300.

Dow,G.S. (2003) Effect of sample size and p-value filtering techniques on the detection of transcriptional changes induced in rat neuroblastoma (NG108) cells by mefloquine, *Malaria Journal*, **2**, 4.

Heller, M. J. (2002) DNA microarray technology: devices, systems, and applications, *Annual Review in Biomedical Engineering*, **4**, 129-153.

Hwang,D., Schmitt, W. A., Stephanopoulos, G., Stephanopoulos, G. (2002) Determination of minimum sample size and discriminatory expression patterns in microarray data, *Bioinformatics*, **18:9**, 1184-1193.

Irizarry, R.A., Hobbs, B., Collin, F. *et al.* (2003) Exploration, normalization, and summaries of high density oligonucleotide array probe level data, *Biostatistics*, **4:2**, 249-264.

Mandel, S., Weinreb, O., Youdim, M. B. H. (2003) Using cDNA microarray to assess Parkinson's disease models and the effects of neuroprotective drugs, *TRENDS in Pharmacological Sciences*, **24:4**, 184-191.

Storey, J., (2002) A direct approach to false discovery rates, *Journal of Royal Statistical Society B*, **64:3**, 479-498.

Warnes, G. R., Liu, P. (2006) Sample Size Estimation for Microarray Experiments, Technical Report, Department of Biostatistics and Computational Biology, University of Rochester.

Yang, Y. H., Speed, T. Design and analysis of comparative microarray experiments, *in Statistical analysis of gene expression microarray data*, Chapman and Hall, 51.

Yang, M. C. K., Yang, J. J., McIndoe, R. A., She, J. X. (2003) Microarray experimental design: power and sample size considerations, *Physiological Genomics*, **16**, 24-28.

Zien, A., Fluck, J., Zimmer, R., Lengauer, T. (2003) Microarrays: how many do you need?, *Journal of Computational Biology*, **10:3-4**, 653-667.

Gregory Warnes
University of Rochester
Dept. of Biostatistics and Computational Biology
Rochester, New York 14642, USA
warnes@bst.rochester.edu

R News is a publication of the R Foundation for Statistical Computing. Communications regarding this publication should be addressed to the editors. All articles are copyrighted by the respective authors. Please send submissions to regular columns to the respective column editor and all other submissions to the editor-in-chief or another member of the editorial board. More detailed submission instructions can be found on the R homepage.

R Project Homepage:
`http://www.R-project.org/`

This newsletter is available online at
`http://CRAN.R-project.org/doc/Rnews/`