goCluster - Developers Manual

Gunnar Wrobel

April 25, 2006

Biozentrum & Swiss Institute of Bioinformatics Klingelbergstrasse 50-70, CH-4056 Basel, Switzerland,
`http://www.biozentrum.unibas.ch/personal/primig/gocluster/`

1 Introduction

goCluster provides a framework that can be extended by modules of different types. This structure allows to significantly decrease the amount of time necessary to adapt an analysis script to the relevant biological question.

A *goCluster*-object is split into six subsections that each handle one aspect of the functional data analysis:

1. Data (clusterData): Holds the dataset that is being analyzed.
2. Annotation (clusterAnnotation): Handles retrival of annotation terms that describe the genes in the given dataset.
3. Algorithm (clusterAlgorithm): Responsible for clustering or partitioning the dataset into different subgroups. The subgroups may be overlapping or non-overlapping.
4. Statistic (clusterStatistic): Performs the main statistical analysis and will generate p-values for the annotation terms found in the subgroups generated by the clustering algorithm.
5. Significance (clusterSignif): Necessary for multiple testing correction. Usually a high number of different annotation terms will be analyzed and to prevent identifying a high number of false positive terms a correction is needed.
6. Visualization (clusterVisual): Visualizes the result once a dataset has been analyzed.

Each of these sections is represented by its own class (the name is given in parantheses). A module that is supposed to fill one of the sections with the required functionality needs to be derived from the corresponding class (e.g. a module that provides a new type of clustering algorithm needs to extend the class *clusterAlgorithm*).

All six classes mentioned above also derive from one common class, the *clusterModule*-class, that provides features common to all of the derived classes (e.g. configuration capabilities).

The following sections will illustrate the most important features of the *goCluster* framework. The final section adds a short introduction on how to include a new module.

2 clusterModule

```
setClass("clusterModule",
         representation(## Interactive configuration
                        config  = "list",
                        ## Non-interactive config checks
                        setup   = "list",
                        ## Main function of the class
                        execute = "function",
                        ## A list of protected slots
                        reset   = "list",
                        ## A list of children
                        children = "list"
                        )
         )
```

The *clusterModule* class defines several slots that hold information for the main functions that can be used with a *goCluster*-object (e.g. config, execute,...). The different slots will be described in greater detail in the subsequent sections.

In order to extend *goCluster* with a new module it is not mandatory to fill each of these slots. In many cases it will be sufficient to define a function for the *execute*-slot in order to provide a new type of functionality that is not yet included in *goCluster*.

3 clusterData

```
setClass("clusterData",
         representation(
                              ## An arbitrary name for the dataset
                              name    = "character",
                              ## The annotation of the dataset
                              anno = "clusterAnnotation"
                              ),
         prototype(config = list(
                           name =
                           function (object)
                           {
                             selectCore("Please select a name or title for the dataset.")
                           }
                           ),
                         setup = list(
                           name =
                           function(X, object)
                           {
                             if (!is.character(X) || !length(X) > 0)
                                 stop("The name of the dataset should be a short string!")
                           }
                           ),
                         children = list(
                           list(slot        = "anno",
                                description = "annotation data",
                                execute     = TRUE
                                )
                           )
                         ),
               contains = "clusterModule"
               )
```

The data section is the only part of *goCluster* that is not intended to be extensible since the data analysis is based on a standard *exprSet*-object as defined in the *Bioconductor* core packages.

Nevertheless the definition of the class given above demonstrates how the basic *clusterModule* class is extended into one of the six main classes that are used to construct the *goCluster* framework. The actual definition of the *clusterData* class is more complex but ispresented in a reduced form here in order to remain illustrative.

The *representation* section adds new slots that are specific to the *clusterData* class. Here the *name* entry and the *anno* slot are being defined. Since the *name* is an entry that should be specified by the user during configuration of the *goCluster* object, the *prototype* defines *config* to be a list that contains a function with the same name. This entry will later be used by the generic *config*-function to request the information from the user. The same approach is used for the *setup* entry. The functions defined here will be used to verify user input for the corresponding slots.

The list specified for the *children*-slot will be used internally by the *goCluster*-object in order to dispatch function calls to children classes.

4 clusterAnnotation

Modules that are intended to provide new types of annotation whithin the *goCluster* framework can store their annotation in the *annoset* slot of a *clusterAnnotation* object. The annoset slot needs to be a list that holds one or more matrices with two columns. Each matrix links the unique identifiers of the expression dataset with the corresponding annotation terms. The *annoset*-list can hold several such matrices so that a module can provide several annotations at the same time (e.g. the three different subsets of gene ontology: molecular function, cellular component, and biological process).

5 clusterAlgorithm

The clustering module will partition the dataset and save the result in the *clusterset* slot of the object. While clustering approaches like *kmeans* partition a dataset in non-overlapping groups hierarchical clustering allows to generate a tree which can be seen as another type of partitioning. But since each branch of the tree splits a larger group of genes into two smaller ones the partitions are overlapping. In order to allow for such a partitioning scheme the *clusterset* slot also holds a list. Each item of that list can either contain a vector of gene indices specifying the genes in the original dataset or it can be another list. This type of structure can be used to represent the full hierarchical tree.

6 clusterStatistic

The task of this type of module is the identification of all annotation terms linked to the genes of each group as they are defined by the *clusterset* slot in the algorithm part of the *goCluster* object. A p-value will be calculated for each of the annotation terms to indicate the probability of this annotation term being significantly enriched in the considered subgroup as compared to all the genes of the dataset. The result of the analysis needs to be stored in the *statset* slot of the object and is also represented by a list with the same structure as the corresponding *clusterset* list. But instead of gene indices the terminal nodes of this list structure carry vectors of p-values named with the corresponding annotation terms.

7 clusterSignif

The modules provided in this slot are intended to reduce the full tree of annotation terms to the terms that can be considered significant on the basis of of multiple correction procedures performed within this module. This selection of terms will also be represented by a list with the same structure as the *statset* but the terminal nodes will only retain the significant annotation terms.

8 clusterVisual

The visualization is the most flexible part of the *goCluster* framework and has no predefined structure. The modules are intended to provide convenience functions for a variety of different situations and the default situation is to use no visualization at all.

A special module derived from the *clusterVisual* class implements this "no visualization" feature and is useful to illustrate a minimal implementation of a *goCluster* module:

```
setClass("clusterVisual",
         representation(
                        ),
         contains = "clusterModule"
         )

setClass("clusterVisualNone", contains = "clusterVisual")
```

This shows the definition of the *clusterVisual* base class as well as the *clusterVisualNone* class as defined in the *goCluster* package.

This is the only module where such a minimal implementation is useful and other modules need at least an *execute* function that will fill the required slot of the class (e.g. *clusterset*, *statset*, ...).

9 New modules

For demonstration purposes the following code would define a new statistical module that could be used within the *goCluster* framework:

```
setClass("clusterStatisticNew",
         representation(
                        ),
         prototype(config = list(),
                   setup  = list(),
                   execute =
                   function(object, parent)
                   {

                     object@statset <- fancyNewStatistic(parent@algo@clusterset,
                                                         parent@data@anno@uniqueid,
                                                         parent@data@anno@annoset
                                                        )

                     object

                   }
                   ),
         contains = "clusterStatistic"
         )
```

Since the new statistic needs no configuration both *config* and *setup* contain no functions for setting parameters. Only the execute function has been defined and is not much more than a wrapper to the *fancyNewStatistic* function that will be called with all information necessary for calculating some useful p-values.

Defining the module this way inside an *R* session will allow you to immediately select the new module during the *goCluster* configuration dialog.

10 Additional functions

In order to give a concise overview on the *goCluster* object using the standard *print* method it is useful to define a short *print* function for each new module. The following is an example from the *clusterAlgorithmHclust* module:

```
setMethod("print", "clusterAlgorithmHclust", function(x, ...)
          {
            printData("Algorithm",
                      "Hclust",
                      c(paste("Distance measure:", x@distance),
                        paste("Clustering method:", x@method),
                        ifelse(length(x@clusterset)!=0,
                        "The dataset has been clustered.",
                        "The object holds no result yet.")))
          }
          )
```

In addition each visualization module should provide a *display* function that draws the image or performs any other function that will produce the visualization as expected from the module. See the provided modules for examples of such functions.