How to Use pkgDepTools

Seth Falcon

April 26, 2022

1 Introduction

The pkgDepTools package provides tools for computing and analyzing dependency relationships among R packages. With it, you can build a graph-based representation of the dependencies among all packages in a list of CRAN-style package repositories. There are utilities for computing installation order of a given package and, if the RCurl package is available, estimating the download size required to install a given package and its dependencies.

This vignette demonstrates the basic features of the package.

2 Graph Basics

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 1 shows a part of the Bioconductor dependency graph for to the Category package. Since circular dependencies are not allowed, the resulting dependency graph will be a directed acyclic graph (DAG).

3 Building a Dependency Graph

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

```
> library("Rgraphviz")
```

The `makeDepGraph` function 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 *graphNEL*[1] 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 `getOption("pkgType")`; 4) `keep.builtin` which will keep packages that come with a standard R install in the dependency graph (the default is `FALSE`).

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 when the `dosize` argument is `TRUE` (this is the default). Obtaining the size of packages requires the RCurl package and can be time consuming for large repositories since a seprate HTTP request must be made for each package. In the examples below, we set `dosize=FALSE` to speed the computations.

```
> library(BiocManager)
> biocUrl <- repositories()["BioCsoft"]
> biocDeps <- makeDepGraph(biocUrl, type="source", dosize=FALSE)

> biocDeps

A graphNEL graph with directed edges
Number of Nodes = 3318
Number of Edges = 18335

> edges(biocDeps)["annotate"]

$annotate
[1] "AnnotationDbi" "XML"           "Biobase"       "DBI"
[5] "xtable"        "BiocGenerics"  "httr"

> ## if dosize=TRUE, size in MB is stored
> ## as a node attribute:
> ## nodeData(biocDeps, n="annotate", attr="size")
```

---

[1]See `help("graphNEL-class")`

4 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 shown in Figure 1 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)
```

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`. The `getInstallOrder` function provides such a "preview".

For computing installation order, it is useful to have a single graph representing the relationships among all packages in all available repositories. Below, we create such a graph combining all CRAN and Bioconductor packages.

```
> allDeps <- makeDepGraph(repositories(), type="source",
+                         keep.builtin=TRUE, dosize=FALSE)
>
```

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
character(0)

$total.size
numeric(0)
```

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

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

```
$packages
 [1] "methods"         "utils"           "graphics"
 [4] "stats"           "BiocGenerics"    "Biobase"
 [7] "stats4"          "S4Vectors"       "IRanges"
[10] "DBI"             "bit"             "bit64"
[13] "rlang"           "glue"            "cli"
[16] "vctrs"           "blob"            "fastmap"
[19] "cachem"          "memoise"         "pkgconfig"
[22] "Rcpp"            "plogr"           "RSQLite"
[25] "curl"            "jsonlite"        "tools"
[28] "mime"            "sys"             "askpass"
[31] "openssl"         "R6"              "httr"
[34] "png"             "zlibbioc"        "XVector"
[37] "bitops"          "RCurl"           "GenomeInfoDbData"
[40] "GenomeInfoDb"    "grDevices"       "crayon"
[43] "Biostrings"      "KEGGREST"        "AnnotationDbi"
[46] "grid"            "lattice"         "Matrix"
[49] "graph"           "BH"              "RBGL"
[52] "XML"             "xtable"          "annotate"
[55] "GSEABase"        "splines"         "survival"
[58] "genefilter"      "Category"        "GO.db"
[61] "AnnotationForge" "Rgraphviz"       "GOstats"
```

```
$total.size
[1] NA
```

The edge directions of the dependency graph can be reversed and the resulting graph used to determine the set of packages that make use of (even indirectly) a given package. For example, one might like to know which packages make use of the methods package. Here is one way to do that:

```
> allDepsOnMe <- reverseEdgeDirections(allDeps)
> usesMethods <- dijkstra.sp(allDepsOnMe, start="methods")$distance
> usesMethods <- usesMethods[is.finite(usesMethods)]
> length(usesMethods) - 1 ## don't count methods itself
```

```
[1] 17769

> table(usesMethods)

usesMethods
    0     1     2     3     4     5     6
    1  5393 11078  1253    42     2     1

>

> toLatex(sessionInfo())
```

- R version 4.2.0 RC (2022-04-19 r82224), x86_64-pc-linux-gnu

- Locale: LC_CTYPE=en_US.UTF-8, LC_NUMERIC=C, LC_TIME=en_GB,
  LC_COLLATE=C, LC_MONETARY=en_US.UTF-8,
  LC_MESSAGES=en_US.UTF-8, LC_PAPER=en_US.UTF-8, LC_NAME=C,
  LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8,
  LC_IDENTIFICATION=C

- Running under: Ubuntu 20.04.4 LTS

- Matrix products: default

- BLAS: /home/biocbuild/bbs-3.15-bioc/R/lib/libRblas.so

- LAPACK: /home/biocbuild/bbs-3.15-bioc/R/lib/libRlapack.so

- Base packages: base, datasets, grDevices, graphics, grid, methods,
  stats, utils

- Other packages: Biobase 2.56.0, BiocGenerics 0.42.0,
  BiocManager 1.30.17, RBGL 1.72.0, RCurl 1.98-1.6, Rgraphviz 2.40.0,
  graph 1.74.0, pkgDepTools 1.62.0

- Loaded via a namespace (and not attached): bitops 1.0-7,
  compiler 4.2.0, stats4 4.2.0, tools 4.2.0

Figure 1: The dependency graph for the Category package.