ontoTools: software for working with ontologies in Bioconductor

VJ Carey `stvjc@channing.harvard.edu`

April 14, 2011

Contents

1 Introduction

An ontology is a vocabulary arranged as a rooted directed acyclic graph (rDAG, or just DAG). This package provides tools for working with such data structures. A key concern is the interpretation of sets of objects mapped to an ontology.

This package relies heavily on the *graph* package from Bioconductor, and the *SparseM* package from UIUC, available from CRAN.

2 Basic structures

2.1 Rooted DAG

The `rootedDAG` class is built using a `graph::graphNEL` and specifying a root.

This package comes with `litOnto`, which is a `graphNEL`.

```
> library(graph)
> library(ontoTools)
> data(litOnto)
> print(litOnto)
```

```
A graphNEL graph with directed edges
Number of Nodes = 12
Number of Edges = 14
```

```
> print(class(litOnto))
```

```
[1] "graphNEL"
attr(,"package")
[1] "graph"
```

Children point to parents by default. When Rgraphviz is available, the graphical structure can be plotted using `plot`.

It is possible to reverse the directions of arcs in such a graph.

The `rootedDAG` object is constructed by hand.

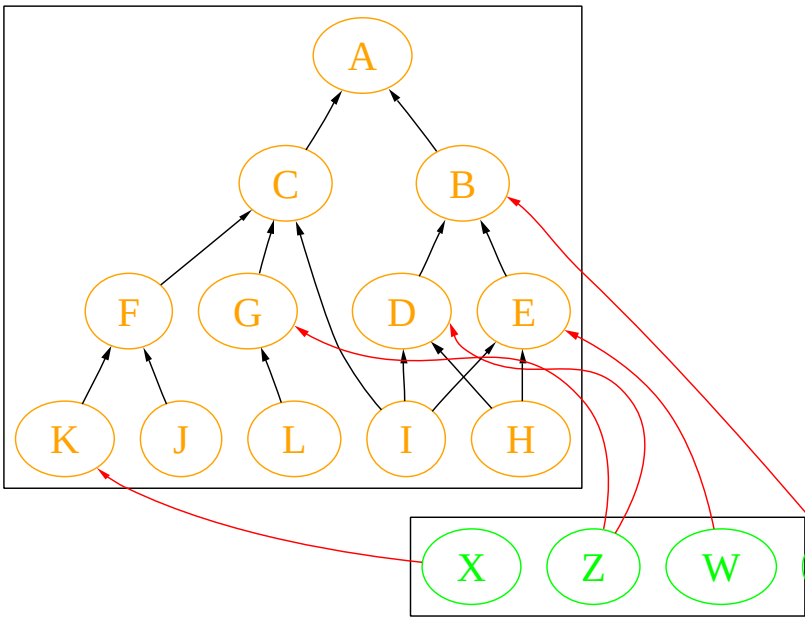
```
> g1 <- new("rootedDAG", DAG = litOnto, root = "A")
> show(DAG(g1))
```

```
A graphNEL graph with directed edges
Number of Nodes = 12
Number of Edges = 14
```

```
> root(g1)
[1] "A"
```

2.2 compoundGraph

This structure was devised to allow presentations like the following:



The R code that generated the dot code from which this image was created, and the dot language code for the image, are as follows:

```
> data(litObj)
> com <- new("compoundGraph", grList=list(litOnto,litObj),
```

```
+          between= list(c("W","E"), c("X", "K"), c("Y","B"),
+         c("Z","D"), c("Z","G")))
> compRendList<-list(
+       list( prenodes="node [fontsize=28 color=orange fontcolor=orange];",
+          preedges="edge [color=black];"),
+       list( prenodes="node [fontsize=28 color=green fontcolor=green];",
+          preedges="edge [color=black];"),
+       betweenRend = list( preedges = "edge [color = red]"))
> ff <- "demoComp.dot"
> toDot(com, ff, compRendList)

dot file written to demoComp.dot  use 'dot -Tps [.dot] [.ps] to render

> cat(readLines(ff),sep="\n")

digraph G {
subgraph cluster_1
 {
 node [fontsize=28 color=orange fontcolor=orange];
 "A" ;
 "B" ;
 "C" ;
 "D" ;
 "E" ;
 "F" ;
 "G" ;
 "H" ;
 "I" ;
 "J" ;
 "K" ;
 "L" ;
 edge [color=black];
 edge [dir=back] "A" -> "B" ;
 edge [dir=back] "A" -> "C" ;
 edge [dir=back] "B" -> "D" ;
 edge [dir=back] "B" -> "E" ;
 edge [dir=back] "C" -> "F" ;
 edge [dir=back] "C" -> "G" ;
 edge [dir=back] "D" -> "H" ;
 edge [dir=back] "E" -> "H" ;
 edge [dir=back] "D" -> "I" ;
 edge [dir=back] "C" -> "I" ;
 edge [dir=back] "E" -> "I" ;
```

```
 edge [dir=back] "F" -> "J" ;
 edge [dir=back] "F" -> "K" ;
 edge [dir=back] "G" -> "L" ;
}
subgraph cluster_2
 {
 node [fontsize=28 color=green fontcolor=green];
 "W" ;
 "X" ;
 "Z" ;
 edge [color=black];
}
 edge [color = red] edge [dir=back] "E"->"W";
 edge [dir=back] "K"->"X";
 edge [dir=back] "B"->"Y";
 edge [dir=back] "D"->"Z";
 edge [dir=back] "G"->"Z";
}
```

2.3 ontology

This is just a lightly annotated `rootedDAG`.

```
> g1 <- new("rootedDAG", DAG = litOnto, root = "A")
> o1 <- new("ontology", name = "demo", version = "0.1", rDAG = g1)
> show(o1)

Ontology object demo, version 0.1
 root= A
 Terms:
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L"
```

2.4 Mapped key-value list

At present this is not a class.

```
> kvlist <- list(W = "E", X = "K", Y = "B", Z = c("D", "G"))
> litMap <- otkvList2namedSparse(names(kvlist), LETTERS[1:12],
+     kvlist)
> print(litMap)

named sparse matrix of dim[1]  4 12
northwest 4x4:
```

```
  A B C D
W 0 0 0 0
X 0 0 0 0
Y 0 1 0 0
Z 0 0 0 1
```

2.5 OOC: object-ontology complex

This combines an ontology and a mapped key-value list that specifies how objects map to terms.

```
> ooc1 <- makeOOC(o1, litMap)
> show(ooc1)

object-ontology complex with ontology:
Ontology object demo, version 0.1
 root= A
 Terms:
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L"
object-ontology map:
named sparse matrix of dim[1]  4 12
northwest 4x4:
  A B C D
W 0 0 0 0
X 0 0 0 0
Y 0 1 0 0
Z 0 0 0 1
```

3 Basic methods

3.1 Matrix conversion of DAG; namedSparse extension

This should be a generic. There are two basic sorts of matrices available: `child2parent` which has 1 in element (i,j) if node i is the child of node j, and an accessibility matrix that has 1 in element (i,j) if node i may be reached from node j.

3.1.1 child2parent

```
> g1 <- new("rootedDAG", DAG = litOnto, root = "A")
> mg1d <- getMatrix(g1, "child2parent", "dense")
> print(mg1d)
```

```
  A B C D E F G H I J K L
A 0 0 0 0 0 0 0 0 0 0 0 0
B 1 0 0 0 0 0 0 0 0 0 0 0
C 1 0 0 0 0 0 0 0 0 0 0 0
D 0 1 0 0 0 0 0 0 0 0 0 0
E 0 1 0 0 0 0 0 0 0 0 0 0
F 0 0 1 0 0 0 0 0 0 0 0 0
G 0 0 1 0 0 0 0 0 0 0 0 0
H 0 0 0 1 1 0 0 0 0 0 0 0
I 0 0 1 1 1 0 0 0 0 0 0 0
J 0 0 0 0 0 1 0 0 0 0 0 0
K 0 0 0 0 0 1 0 0 0 0 0 0
L 0 0 0 0 0 0 1 0 0 0 0 0
```

Typically a sparse representation will be needed.

```
> ng1 <- getMatrix(g1, "child2parent", "sparse")
> ng1 <- new("namedSparse", mat=ng1)
> dimnames(ng1) <- list(as.character(1:dim(ng1@mat)[1]),
+      as.character(1:dim(ng1@mat)[2]))
> print(ng1@mat)

An object of class "matrix.csr"
Slot "ra":
 [1] 0 1 1 1 1 1 1 1 1 1 1 1 1 1

Slot "ja":
 [1] 1 1 1 2 2 3 3 4 5 4 3 5 6 6 7

Slot "ia":
 [1]  1  2  3  4  5  6  7  8 10 13 14 15 16

Slot "dimension":
[1] 12 12

> print(as.matrix(ng1))

   1 2 3 4 5 6 7 8 9 10 11 12
1  0 0 0 0 0 0 0 0 0  0  0  0
2  1 0 0 0 0 0 0 0 0  0  0  0
3  1 0 0 0 0 0 0 0 0  0  0  0
4  0 1 0 0 0 0 0 0 0  0  0  0
5  0 1 0 0 0 0 0 0 0  0  0  0
```

```
6  0 0 1 0 0 0 0 0 0  0  0  0
7  0 0 1 0 0 0 0 0 0  0  0  0
8  0 0 0 1 1 0 0 0 0  0  0  0
9  0 0 1 1 1 0 0 0 0  0  0  0
10 0 0 0 0 0 1 0 0 0  0  0  0
11 0 0 0 0 0 1 0 0 0  0  0  0
12 0 0 0 0 0 0 1 0 0  0  0  0
```

We have not implemented a full set of element subsetting operations as we expect the SparseM authors to do so. However, efficient operations using names of rows and columns are essential. These are implemented using hashed environments.

```
> dimnames(ng1) <- list(letters[1:12], LETTERS[1:12])
> print(class(ng1))

[1] "namedSparse"
attr(,"package")
[1] "ontoTools"

> print(getSlots("namedSparse"))

    Dimnames          mat
      "list" "matrix.csr"

> print(as.matrix(ng1))

  A B C D E F G H I J K L
a 0 0 0 0 0 0 0 0 0 0 0 0
b 1 0 0 0 0 0 0 0 0 0 0 0
c 1 0 0 0 0 0 0 0 0 0 0 0
d 0 1 0 0 0 0 0 0 0 0 0 0
e 0 1 0 0 0 0 0 0 0 0 0 0
f 0 0 1 0 0 0 0 0 0 0 0 0
g 0 0 1 0 0 0 0 0 0 0 0 0
h 0 0 0 1 1 0 0 0 0 0 0 0
i 0 0 1 1 1 0 0 0 0 0 0 0
j 0 0 0 0 0 1 0 0 0 0 0 0
k 0 0 0 0 0 1 0 0 0 0 0 0
l 0 0 0 0 0 0 1 0 0 0 0 0
```

3.1.2 Accessibility

Currently this is only handled for an ontology.

```
> show(accessMat(o1))

named sparse matrix of dim[1] 12 12
northwest 4x4:
  A B C D
A 0 0 0 0
B 1 0 0 0
C 1 0 0 0
D 1 1 0 0
```

3.2 Coverage matrix for an OOC

This matrix has element 1 in row r, column c if object corresponding to row r is covered by the term corresponding to column c. This means that the term for column c is accessible from some term to which the object for row r was explicitly mapped.

```
> print(coverageMat(ooc1))

named sparse matrix of dim[1]  4 12
northwest 4x4:
  A B C D
W 1 1 0 0
X 1 0 1 0
Y 1 1 0 0
Z 1 1 1 1
```

3.3 Depth of terms in an ontology

We need to be able to get the set of terms at a certain depth (root is depth 0) and the depth of any given term.

```
> print(ontoDepth(g1))

A B C D E F G H I J K L
0 1 1 2 2 2 2 3 3 3 3 3

> ds1 <- depthStruct(g1)
> print(ds1$tag2depth("B"))

[1] 1

> print(ds1$depth2tags(3))

[1] "H" "I" "J" "K" "L"
```

4 Applications to large data objects

4.1 Key-value list for LocusLink to GO

The `org.Hs.eg.db` metadata package from Bioconductor includes org.Hs.egGO, to regarded as a key-value environment mapping. The otkvEnv2namedSparse function can be used with this environment and the GOMF terms to create the object to term mapping. See ooMapLL2GOMFdemo.rda.

4.2 GO graph and ontology for molecular function

The buildGOGraph function was used to create goMFgraph.1.15.rda on the basis of the GO package.

With this graph in hand, we construct the rooted DAG

```
> data(goMFgraph.1.15)
> gomfrDAG <- new("rootedDAG", root = "GO:0003674", DAG = goMFgraph.1.15)
```

and then the ontology:

```
> GOMFonto <- new("ontology", name = "GOMF", version = "bioc 1.3.1",
+     rDAG = gomfrDAG)
```

The term accessibility matrix is important for semantic calculations. This is a slow calculation and its result is archived.

```
gomfAmat <- accessMat(GOMFonto)
save(gomfAmat,file="gomfAmat.rda", compress=TRUE)
```

4.3 Mapping from LocusLink to GO

The LocusLink database is a collection of one-to-many mappings from genes or gene-associated sequence to GO tags. This object-to-term ('ot') association is provided as an environment-like resource by bioconductor in the org.Hs.egGO package. We will use a named sparse matrix structure to encode the object to term mapping. For this example, the mapping is archived after computation by the following code:

```
library(org.Hs.eg.db)
data(goMFgraph1.15)
obs <- ls(org.Hs.egGO)
tms <- nodes(goMFgraph.1.15)
ooMapLL2GOMFdemo <- otkvEnv2namedSparse( obs, tms, org.Hs.egGO )
save(ooMapLL2GOMFdemo,file="ooMapLL2GOMFdemo.rda")
```

5 Applications

5.1 Lord's *Bioinformatics* 2003 paper

Lord's paper on semantic similarity measures defines a probability $p(c)$ for each concept term c. Semantic similarity between terms c_1 and c_2 can be measured as $-\log p_{ms}(c_1, c_2)$, where p_{ms} is the so-called "probability of the minimum subsumer", defined as

$$p_{ms}(c_1, c_2) = \min_{c \in S(c_1, c_2)} \{p(c)\},$$

where $S(c_1, c_2)$ is the set of all terms that are refined by both c_1 and c_2.

We indicate how these computations can be performed using test data in *ontoTools*. Most of the functions defined here as x.vig are included in the package (sometimes with slight modifications), lacking the vig suffix.

5.1.1 Setup of the data

We begin by attaching the package and convering the `litOnto` information (encoding the 12-term ontology) into a `rootedDAG`.

```
> library(ontoTools)
> data(litOnto)
> g1 <- new("rootedDAG", DAG = litOnto, root = "A")
```

Now we create an ontology object based on this rooted DAG:

```
> o1 <- new("ontology", name = "demo", version = "0.1", rDAG = g1)
```

The mapping from objects to the concepts in the ontology uses a key-value list, with multimappings represented by vectors.

```
> kvlist <- list(W = "E", X = "K", Y = "B", Z = c("D", "G"))
```

A special function compute the object to term map:

```
> demomap <- otkvList2namedSparse(names(kvlist), LETTERS[1:12],
+     kvlist)
```

We obtain the object-ontology complex.

```
> demoooc <- makeOOC(o1, demomap)
```

The map from objects to terms is:

```
> print(OOmap(demoooc))
```

```
named sparse matrix of dim[1]  4 12
northwest 4x4:
  A B C D
W 0 0 0 0
X 0 0 0 0
Y 0 1 0 0
Z 0 0 0 1
```

The coverage matrix can be directly computed:

```
> cov1 <- coverageMat(demoooc)
> print(cov1)

named sparse matrix of dim[1]  4 12
northwest 4x4:
  A B C D
W 1 1 0 0
X 1 0 1 0
Y 1 1 0 0
Z 1 1 1 1
```

5.1.2 Concept probabilities

We begin with the concept probabilities. The counts of usages per term is

```
> ACC <- accessMat(ontology(demoooc))
> acctms <- dimnames(ACC)[[2]]
> print(acctms)

 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L"

> MAP <- OOmap(demoooc)
> nterms <- function(x) length(nodes(DAG(rDAG(x))))
> ontoTerms <- function(x) nodes(DAG(rDAG(x)))
> print(ontoTerms)

function (x)
nodes(DAG(rDAG(x)))

> usageCount.vig <- function(MAP, ACC) {
+     maptms <- dimnames(MAP)[[2]]
+     acctms <- dimnames(ACC)[[2]]
+     usages <- rep(0, length(maptms))
+     names(usages) <- maptms
```

```
+     for (i in 1:nrow(MAP)) {
+         hits <- maptms[as.matrix.ok(MAP[i, ]) == 1]
+         usages[hits] <- usages[hits] + 1
+         for (j in 1:length(hits)) {
+             anctags <- acctms[as.matrix.ok(ACC[hits[j], ]) ==
+                 1]
+             usages[anctags] <- usages[anctags] + 1
+         }
+     }
+     usages
+ }
> print(usages <- usageCount.vig(MAP, ACC))

A B C D E F G H I J K L
5 3 2 1 1 1 1 0 0 0 1 0
```

because whenever a term is used, all its ancestors are implicitly used as well (Lord section 2). This inherited usage does not recurse, however. A basic usage event fires up the graph once; the fact that a descendent is used transitively does not increment the usage of its ancestors.

The total number of basic usage events is

```
> print(N <- max(usages))

[1] 5
```

The vector of concept probabilities is

```
> print(usages/N)

  A   B   C   D   E   F   G   H   I   J   K   L
1.0 0.6 0.4 0.2 0.2 0.2 0.2 0.0 0.0 0.0 0.2 0.0
```

So we have

```
> conceptProbs.vig <- function(ooc) {
+     if (is(occ, "OOC"))
+         stop("arg must have class OOC")
+     pc <- usageCount.vig(OOmap(ooc), accessMat(ontology(ooc)))
+     pc/max(pc, na.rm = TRUE)
+ }
```

5.1.3 Identifying the most informative subsumer

This task involves the ontology (to identify subsumers) and then the concept probabilities.

```
> subsumers.vig <- function(c1, c2, ont) {
+     if (!is(ont, "ontology"))
+         stop("ont must have class ontology")
+     tmp <- colSums(accessMat(ont)[c(c1, c2), ])
+     names(tmp[tmp == 2])
+ }
> print(subsumers.vig("I", "K", ontology(demoooc)))
```

```
[1] "A" "C"
```

Thus we can use

```
> pms.vig <- function(c1, c2, ooc) {
+     if (!is(ooc, "OOC"))
+         stop("arg must have class OOC")
+     if (any(!(c(c1, c2) %in% nodes(DAG(rDAG(ontology(ooc)))))))
+         stop("some term not found in ontology DAG nodes")
+     S <- subsumers.vig(c1, c2, ontology(ooc))
+     pc <- conceptProbs.vig(ooc)
+     min(pc[S])
+ }
> print(pms("I", "K", demoooc))
```

```
progress in ( 1 , 4 ):
[1] 0.4
```

Now we can compute semantic similarity relative to an OOC:

```
> semsim.vig <- function(c1, c2, ooc) -log(pms.vig(c1, c2, ooc))
```

5.2 Iyer's time-course experiment

5.2.1 Setup

We attach the Iyer517 library and the IyerAnnotated data structure. IyerAnnotated is a data.frame that gives the mapping from probes in Iyer517 to up to 5 GO MF tags.

```
> library(ontoTools)
> library(Iyer517)
> data(IyerAnnotated)
> print(IyerAnnotated[1:3, ])
```

```
  Iclust       GB seqno locusid        GO1        GO2  GO3  GO4  GO5
1      N   W95909     1   80298       <NA>       <NA> <NA> <NA> <NA>
2      A AA045003     2    6414 GO:0008430       <NA> <NA> <NA> <NA>
3      A AA044605     3    5300 GO:0003755 GO:0005515 <NA> <NA> <NA>
```

We also attach the concept probabilities for GO MF, relative to the LocusLink usages.

```
> data(LL2GOMFcp.1.15)
> print(LL2GOMFcp.1.15[1:3])

  GO:0000010   GO:0000014   GO:0000016
4.675082e-05 4.675082e-05 2.337541e-05
```

We also define a function that extracts the most informative of a set of tags:

```
> getMostInf <- function(tags, pc = LL2GOMFcp.1.15) {
+     if (length(tags) == 1)
+         return(tags)
+     if (all(is.na(tags)))
+         return(NA)
+     tags[pc[tags] == min(pc[tags])][1]
+ }
```

and apply this to the probe-specific tag sets:

```
> GOmost <- apply(IyerAnnotated[, 5:9], 1,
+   function(x) getMostInf(as.character(x[!is.na(x)])))
```

We also need an OOC for GOMF relative to LocusLink:

```
> data(goMFgraph.1.15)
> data(LL2GOMFooMap.1.15)
> data(goMFamat.1.15)
> gomfrDAG <- new("rootedDAG", root = "GO:0003674", DAG = goMFgraph.1.15)
> GOMFonto <- new("ontology", name = "GOMF", version = "bioc 1.3.1",
+     rDAG = gomfrDAG)
> LLGOMFOOC <- makeOOC(GOMFonto, LL2GOMFooMap.1.15)
```

5.2.2 Filtering the IyerAnnotated tags

The Iyer data have been clustered.

```
> print(table(IyerAnnotated$Iclust))

  A   B   C   D   E   F   G   H   I   J   N
100 145  34  43   7  34  15  63  19  25  32
```

We will work with the four largest clusters, extracting the most informative tag for each cluster.

```
> getMostInfTags <- function(let) {
+ grp.GO <- GOmost[ IyerAnnotated$Iclust==let ]
+ grp.GO[!is.na(grp.GO)]
+ }
> A.GO <- getMostInfTags("A")
> B.GO <- getMostInfTags("B")
> D.GO <- getMostInfTags("D")
> H.GO <- getMostInfTags("H")
>
```

5.2.3 Computing pairwise semantic similarity

The next function computes the pairwise semantic similarities in a group of tags. However, to save time, the loop is truncated at 30 pairs.

```
> getSim <- function(x, Nchk = 30) {
+     library(combinat)
+     prs <- combn(x, 2)
+     sim <- rep(NA, ncol(prs))
+     for (i in 1:ncol(prs)) {
+         if (i > Nchk)
+             break
+         cat(i)
+         if (any(!(c(prs[1, i], prs[2, i]) %in% goMFamat.1.15@Dimnames[[1]])))
+             sim[i] = NA
+         else sim[i] <- semsim(prs[1, i], prs[2, i], acc = goMFamat.1.15,
+             pc = LL2GOMFcp.1.15, ooc = LLGOMFOOC)
+     }
+     sim[1:Nchk]
+ }
```

We compute the pairwise similarities for the 4 groups.

```
> Asim <- getSim(A.GO)

12345678910111213141516171819202122324252627282930

> save(Asim, file = "Asim.rda")
> Bsim <- getSim(B.GO)

12345678910111213141516171819202122324252627282930
```

```
> save(Bsim, file = "Bsim.rda")
> Dsim <- getSim(D.GO)

12345678910111213141516171819202122324252627282930

> save(Dsim, file = "Dsim.rda")
> Hsim <- getSim(H.GO)

12345678910111213141516171819202122324252627282930

> save(Hsim, file = "Hsim.rda")
```

Display the similarity statistics:

```
> boxplot(Asim, Bsim, Dsim, Hsim)
```