RBGL: R interface to boost graph library

VJ Carey `stvjc@channing.harvard.edu`

April 7, 2003

Summary. A very preliminary implementation of an interface from R to the Boost Graph Library (BGL, an alternative to STL programming for mathematical graph objects) is presented. *This 2003 update employs the* graph *class of Bioconductor.*

1 Working with the Bioconductor graph class

An example object representing file dependencies is included, as shown in Figure 1.

```
> library(RBGL)

Loading required package: graph
Loading required package: Biobase
Welcome to Bioconductor
         Vignettes contain introductory material.  To view,
         simply type: openVignette()
         For details on reading vignettes, see
         the openVignette help page.
Creating a new generic function for "summary" in package
reposTools
Creating a new generic function for "print" in package
Ruuid

> data(FileDep)
> print(FileDep)

A graph with  directed  edges
Number of Nodes = 15
Number of Edges = 19

> library(Rgraphviz)
```

```
> z <- plot(FileDep)
```

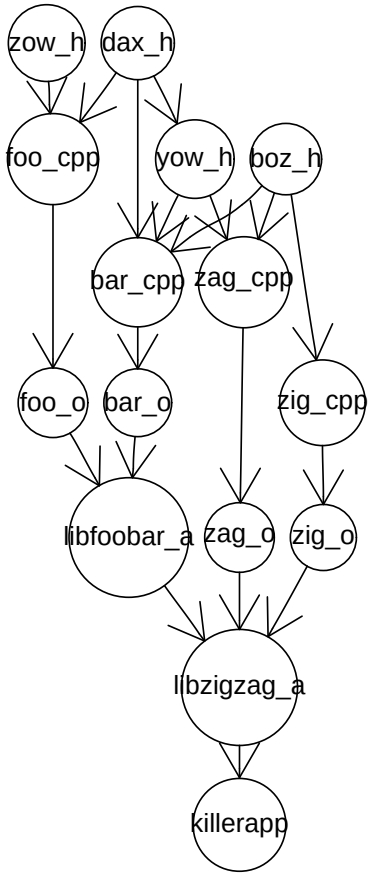


Figure 1: File dependency digraph example from Boost library.

```
Creating a new generic function for "plot" in package
Rgraphviz
Creating a new generic function for "lines" in package
Rgraphviz
```

1.1 Topological sort

The `tsort` function will return the indices of vertices in topological sort order:

```
> ts <- tsort(FileDep)
> print(nodes(FileDep)[ts + 1])
```

```
 [1] "zow_h"       "boz_h"       "zig_cpp"     "zig_o"       "dax_h"
 [6] "yow_h"       "zag_cpp"     "zag_o"       "bar_cpp"     "bar_o"
[11] "foo_cpp"     "foo_o"       "libfoobar_a" "libzigzag_a" "killerapp"
```

Note that if the input graph is not a DAG, BGL `topological_sort` will check this and throw 'not a dag'. This is crudely captured in the interface (a message is written to the console and zeroes are returned).

```
#FD2 <- FileDep
# now introduce a cycle
#FD2@edgeL[["bar_cpp"]]$edges <- c(8,1)
#tsort(FD2)
```

1.2 Kruskal's minimum spanning tree

This function just returns a list of edges, weights and nodes determining the minimum spanning tree (MST) by Kruskal's algorithm.

```
> km <- fromGXL(file(system.file("GXL/kmstEx.gxl", package = "graph")))

Loading required package: XML

> print(KMST(km))

$edgeList
     [,1] [,2] [,3] [,4]
[1,]    1    4    5    2
[2,]    3    5    1    4

$weights
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1

$nodes
[1] "A" "B" "C" "D" "E"
```

1.3 Depth first search

This function returns a list of node indices by discovery and finish order.

```
> df <- fromGXL(file(system.file("XML/dfsex.gxl", package = "RBGL")))
> print(o <- dfsBGL(df))
```

```
> z <- plot(km)
```

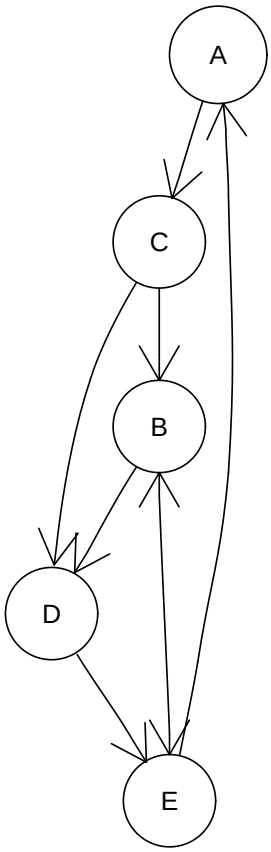


Figure 2: Kruskal MST example from Boost library.

```
> z <- plot(df)
```

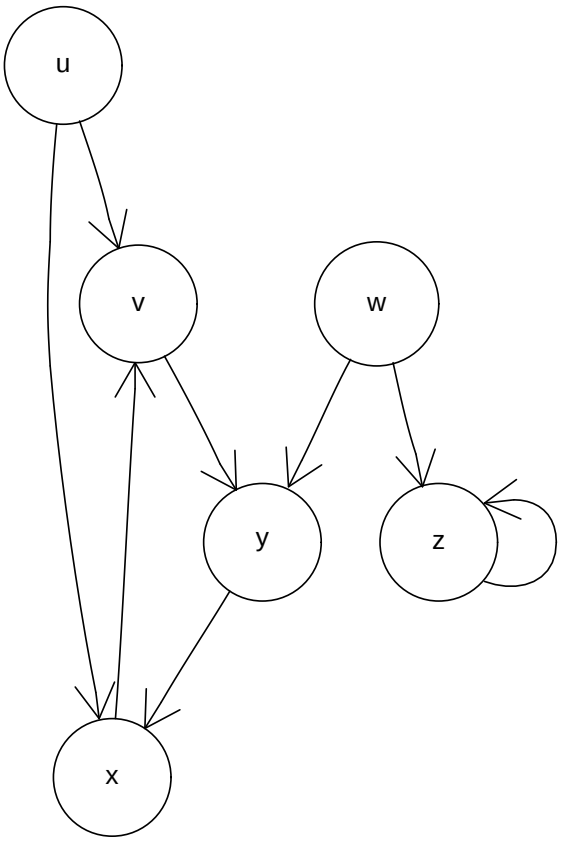


Figure 3: DFS example from Boost library.

```
$discovered
[1] 1 2 5 4 3 6

$finish
[1] 4 5 2 1 6 3
```

Here is the list of nodes in DFS discovery order.

```
> print(nodes(df)[o$discovered])

[1] "u" "v" "y" "x" "w" "z"
```

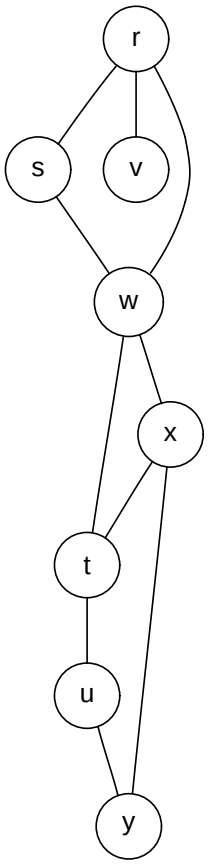
1.4 Breadth first search

This function returns a vector of node indices for a breadth-first search (BFS) starting at the node indexed by `init.ind`.

```
> bf <- fromGXL(file(system.file("XML/bfsex.gxl", package = "RBGL")))
> bf@edgemode <- "undirected"
> print(o <- bfsBGL(bf, init.ind = 2))

[1] 2 1 6 5 3 7 4 8

> z <- plot(bf)
```



The nodes in BFS order starting with the second node are

```
> print(nodes(bf)[o])

[1] "s" "r" "w" "v" "t" "x" "u" "y"
```