Introductory Graph Lab

Robert Gentleman, Wolfgang Huber, Vince Carey

June 12, 2004

```
> library(graph)
> library(Rgraphviz)
```

Note for MS Windows users: The R package *Rgraphviz* is an interface to the graph layout program *graphviz*. With this, you can directly visualize graphs from within R, for example using the `plot` method. Currently, *Rgraphviz* does not run under MS-Windows. However, *graphviz* does. Here, we provide a simple function that uses files for a uni-directional communication from R to *graphviz*.

```
> writeDot <- function(g, y = "dot", f) {
+     filegxl <- paste(f, ".gxl", sep = "")
+     filedot <- paste(f, ".dot", sep = "")
+     filegif <- paste(f, ".gif", sep = "")
+     saveXML(toGXL(g)$value(), file = filegxl)
+     system(paste("gxl2dot", filegxl, ">", filedot))
+     system(paste(y, "-Tgif", filedot, ">", filegif))
+     return(filegif)
+ }
```

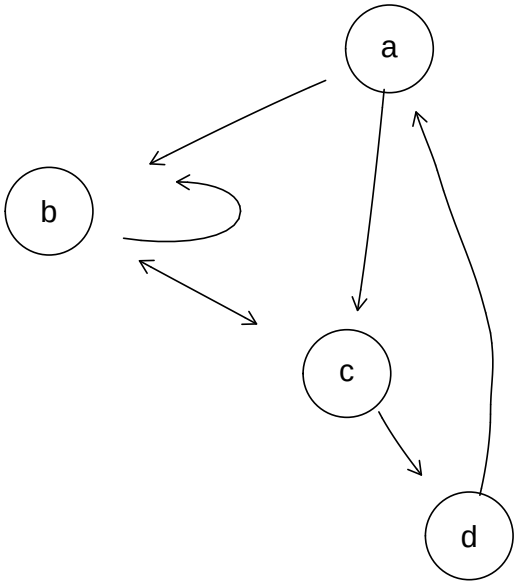
1 The *graph* package

First, we create a simple example graph:

```
> edges <- list(a = list(edges = 2:3), b = list(edges = 2:3), c = list(edges = c(2,
+     4)), d = list(edges = 1))
> g <- new("graphNEL", nodes = letters[1:4], edgeL = edges, edgemode = "directed")
> g

A graph with  directed  edges
Number of Nodes = 4
Number of Edges = 7
```

and plot it:

```
> plot(g, main = "My first graph")
> writeDot(g, f = "myfirstgraph")
```



We can find out about the nodes, edges, and node degrees of g:

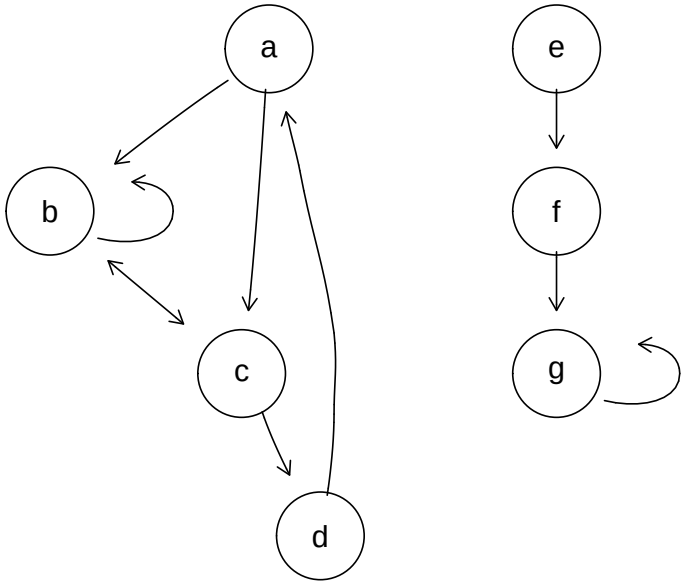
```
> nodes(g)

[1] "a" "b" "c" "d"

> edges(g)

$a
[1] "b" "c"

$b
[1] "b" "c"

$c
[1] "b" "d"

$d
[1] "a"

> degree(g)
```

```
$inDegree
a b c d
1 3 2 1

$outDegree
a b c d
2 2 2 1
```

The functions `adj` and `acc` provide the names of the *adjacent* and *accessible* nodes:

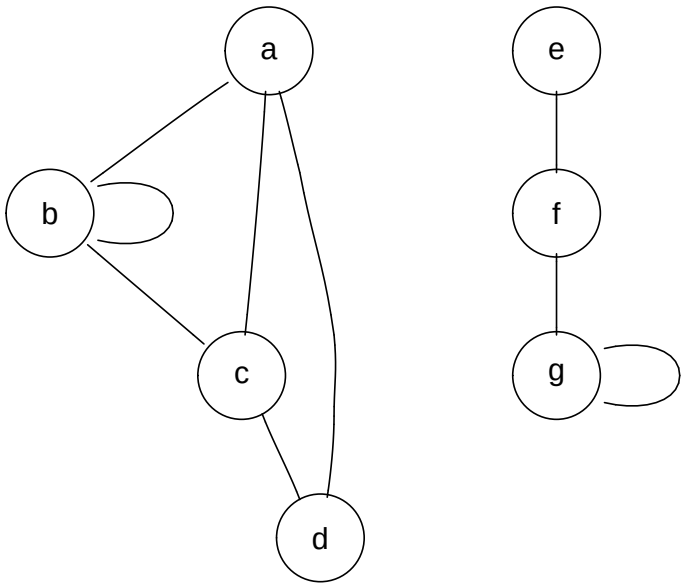
```
> edges <- list(a = list(edges = 2:3), b = list(edges = 2:3), c = list(edges = c(2,
+     4)), d = list(edges = 1), e = list(edges = 6, 7), f = list(edges = 7),
+     g = list(edges = 7))
> g <- new("graphNEL", nodes = letters[1:7], edgeL = edges, edgemode = "directed")
> plot(g, main = "Example for adj, acc")
> adj(g, c("b", "c"))

$b
[1] "b" "c"

$c
[1] "b" "d"

> acc(g, c("b", "c"))

$b
a c d
3 1 2

$c
a b d
2 1 1
```

Example for adj, acc



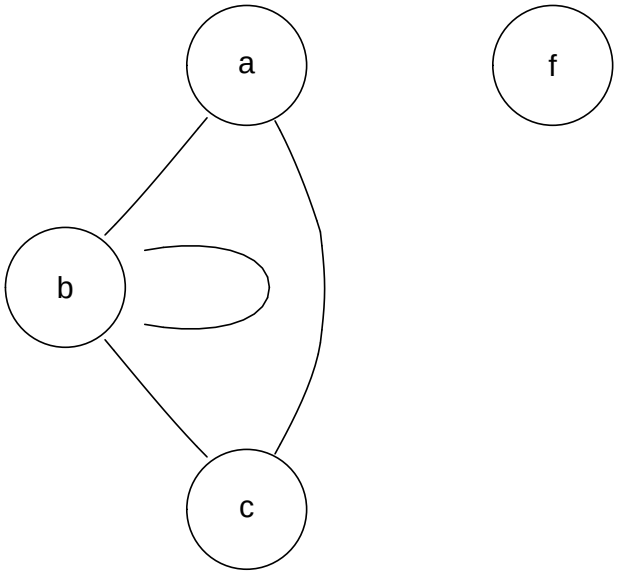
From the directed graph, we can construct the corresponding *undirected graph*:

```
> ug <- ugraph(g)
> plot(ug, main = "Undirected Graph")
```



... and a *subgraph*

```
> sg <- subGraph(c("a", "b", "c", "f"), ug)
> plot(sg, main = "subGraph")
```



... and the *boundary* of the subgraph within the larger graph.

```
> bd <- boundary(sg, ug)
> bd

$a
[1] "d"

$b
character(0)

$c
[1] "d"

$f
[1] "e" "g"
```

We can also define *edge weights* on our graphs:

```
> edges <- list(a = list(edges = 2:3, weights = 1:2), b = list(edges = 2:3,
+     weights = c(0.5, 1)), c = list(edges = c(2, 4), weights = c(2:1)),
```

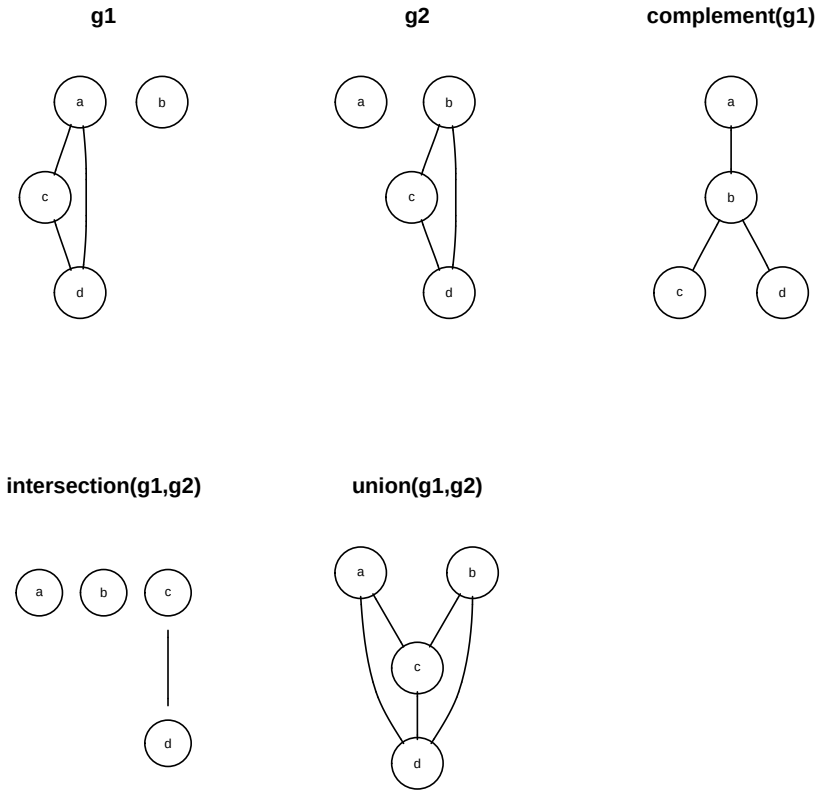
```
+     d = list(edges = 1, weights = 3))
> g <- new("graphNEL", nodes = letters[1:4], edgeL = edges, edgemode = "directed")
> edgeWeights(g)

$a
2 3
1 2

$b
  2   3
0.5 1.0

$c
2 4
2 1

$d
1
3
```

Graph manipulation functions allow to add and remove edges:

```
> g1 <- addNode("e", g)
> g2 <- removeNode("d", g)
> g3 <- addEdge("e", "a", g1, pi/2)
> g4 <- removeEdge("e", "a", g3)
> identical(g4, g1)

[1] TRUE
```

Graph algebra: complement, union, intersection

```
> par(mfrow = c(2, 3))
> V <- letters[1:4]
> set.seed(4713)
> g1 <- randomGraph(V, 1, 0.55)
> g2 <- randomGraph(V, 1, 0.55)
> plot(g1, main = "g1")
> plot(g2, main = "g2")
> plot(complement(g1), main = "complement(g1)")
> plot(intersection(g1, g2), main = "intersection(g1,g2)")
> plot(union(g1, g2), main = "union(g1,g2)")
> par(mfrow = c(1, 1))
```



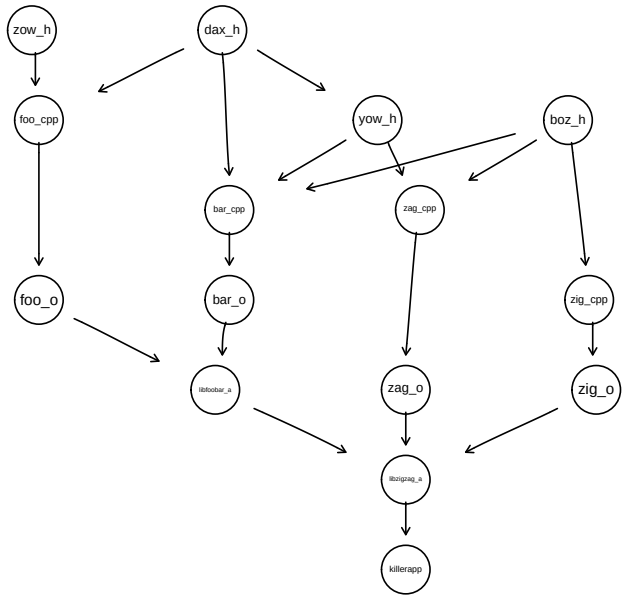
2 The *RBGL* package

The *tsort* function calculates the *topological sort order* for a directed acyclic graph. The topological sort order is defined as follows: if edge (u, v) appears in the graph, then u comes before v in the ordering.

```
> library(RBGL)
> data(FileDep)
> plot(FileDep, main = "Topological sort order example graph")
> ts <- tsort(FileDep)

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.

> 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"
```

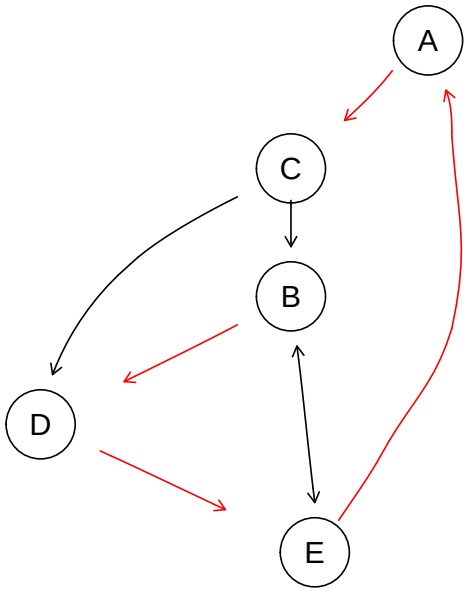
Topological sort order example graph



The function *mstree.kruskal* provides Kruskal's minimum spanning tree:

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

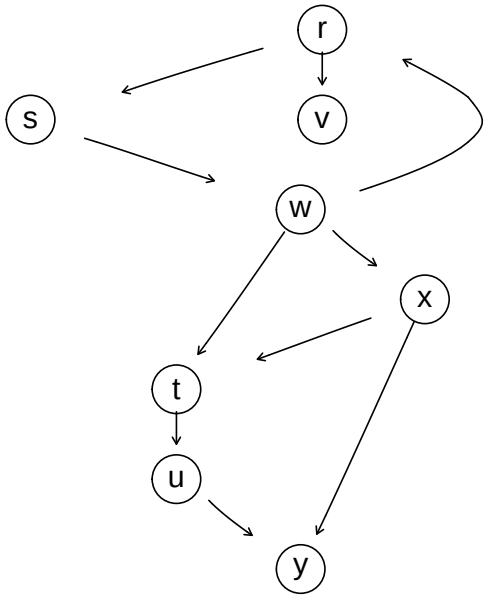
Topological sort order example graph



Breadth first and *depth first* search:

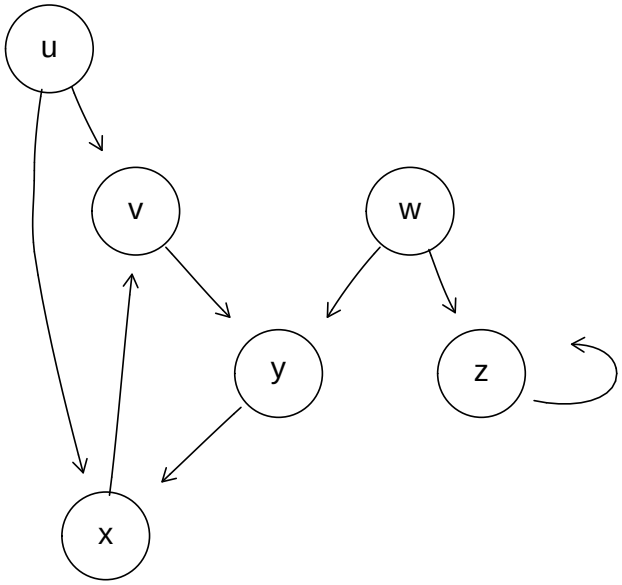
```
> dd <- fromGXL(file(system.file("XML/bfsex.gxl", package = "RBGL")))
> plot(dd, main = "Breadth first search example graph")
> br <- bfs(dd, "r")
> nodes(dd)[br]

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

> bs <- bfs(dd, "s")
> nodes(dd)[bs]

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

Breadth first search example graph




```
> dd <- fromGXL(file(system.file("XML/dfsex.gxl", package = "RBGL")))
> plot(dd, main = "Depth first search example graph")
> df <- dfs(dd, "u")
> nodes(dd)[df$discovered]

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

> nodes(dd)[df$finish]

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

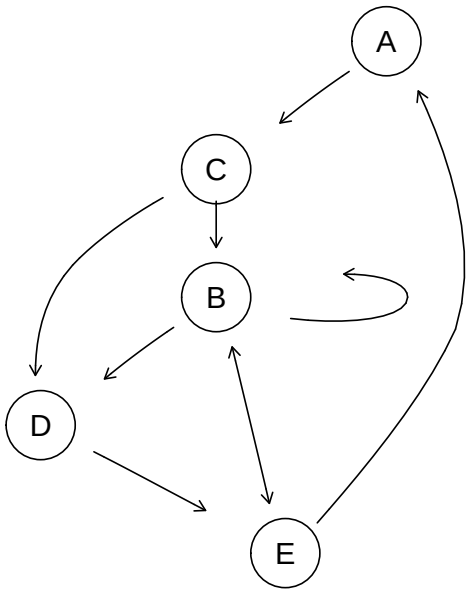
Depth first search example graph




```
> g <- fromGXL(file(system.file("XML/dijkex.gxl", package = "RBGL")))
> plot(g, main = "Shortest path example graph")
> sp.between(g, "E", "C")

$"E:C"
$"E:C"$path
[1] "E" "A" "C"

$"E:C"$length
[1] 2

$"E:C"$pweights
E->A A->C
   1    1

> dijkstra.sp(g)

$distances
A B C D E
0 6 1 4 5
```

```
$penult
A B C D E
1 5 1 3 4

$start
A
1
```

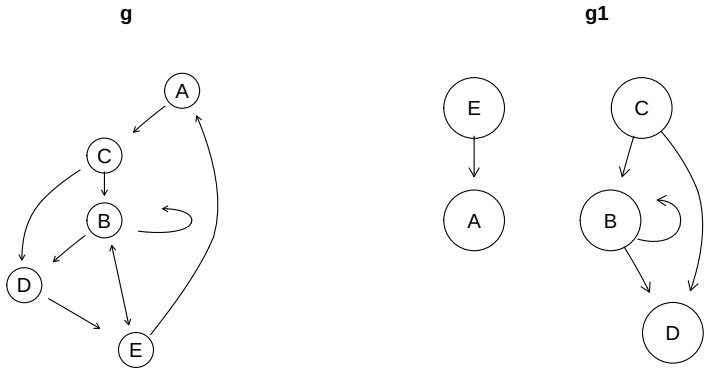
Shortest path example graph




```
> g1 <- removeEdge("A", "C", g)
> g1 <- removeEdge("D", "E", g1)
> g1 <- removeEdge("B", "E", g1)
> g1 <- removeEdge("E", "B", g1)
> connectedComp(g)

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

> connectedComp(g1)
```

```
$"1"
[1] "A" "E"

$"2"
[1] "B" "C" "D"
```

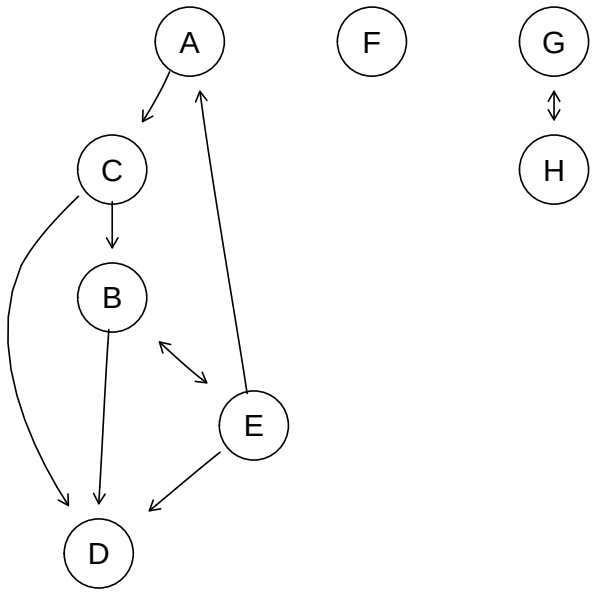
```
> par(mfrow = c(1, 2))
> plot(g, main = "g")
> plot(g1, main = "g1")
```




```
> km <- fromGXL(file(system.file("XML/kmstEx.gxl", package = "RBGL")))
> km@nodes <- c(km@nodes, "F", "G", "H")
> km@edgeL$F <- list(edges = numeric(0))
> km@edgeL$G <- list(edges = 8)
> km@edgeL$H <- list(edges = 7)
> strongComp(km)

$"1"
[1] "D"

$"2"
[1] "A" "B" "C" "E"

$"3"
[1] "F"

$"4"
[1] "G" "H"

> connectedComp(ugraph(km))

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

```
$"2"
[1] "F"

$"3"
[1] "G" "H"
```

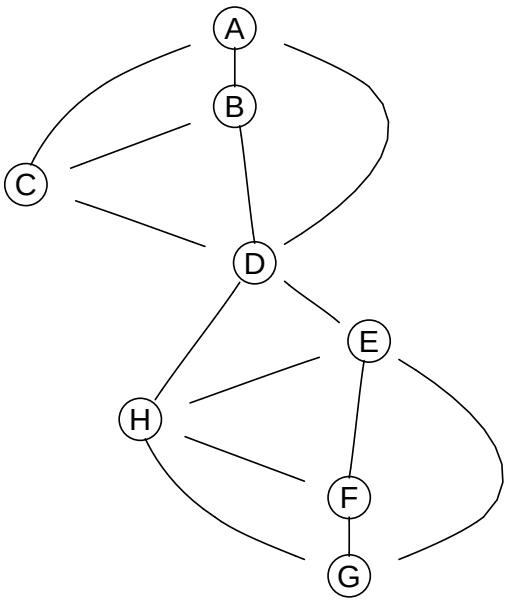
```
> plot(km, main = "km: connected components example graph")
```




```
> g <- fromGXL(file(system.file("XML/conn.gxl", package = "RBGL")))
> edgeConnectivity(g)
```

```
$connectivity
[1] 2

$minDisconSet
$minDisconSet[[1]]
[1] "D" "E"

$minDisconSet[[2]]
[1] "D" "H"
```

```
> plot(g, main = "g: edgeConnectivity example graph")
```




```
> attrs <- getDefaultAttrs()
> attrs$node$fillcolor <- "red"
> attrs$node$height <- "1"
> attrs$node$label <- " "
> myplot <- function(m) plot(FileDep, m, main = m, attrs = attrs)
> par(mfrow = c(1, 3))
> myplot("dot")
> myplot("neato")
> myplot("twopi")
```