A simple graph system - `gRash`

Søren Højsgaard

December 19, 2007

Contents

1 Introduction

This is a technical note which describes a simple "graph system" in R called gRash. The system is used in the gRain package[1] for graphical independence networks. Thus gRash is not an R package but a part of an R package.

For the R community, the triplet triplet of the packages graph, RBGL and Rgraphviz consitutues tool for graph operations, manipulation and layout. The gRash system is not intended to be a strong competitor for these fine packages. On the contrary, part of the gRash functionality uses the other packages.

The main virtue of the gRash system is that graphs are specified in a way closer to normal text book representations and the same applies to some extent to the graph operations.

Only undirected and directed acyclic graphs are implemented.

2 Graphs

2.1 Undirected graphs

An undirected graph is created by the newugsh function. The graph can be specified by an incidence list in either of two different forms:

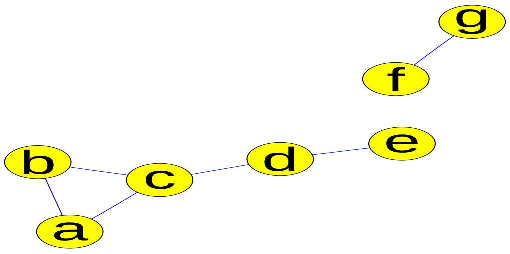
```
> ug1 <- newug(~a + b + c, ~c + d, ~d + e, ~f + g)
> ug1 <- newug(c("a", "b", "c"), c("c", "d"), c("d", "e"), c("f", "g"))
> ug1
```

```
Undirected graph
Nodes: a b c d e f g
Edges: a~b a~c b~c c~d d~e f~g
```

Graphs are displayed with plot:

```
> plot(ug1)
```

[1] The package is not on CRAN but is available from http://gbi.agrsci.dk/~shd/Public/gRainweb/



2.2 Directed acyclic graphs

A directed acyclic graph can be specified as:

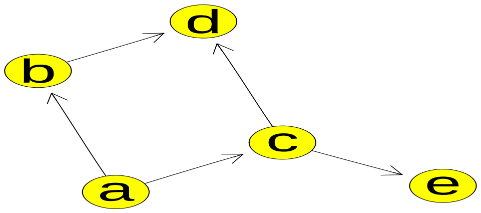
```
> dag1 <- newdag(~a, ~b + a, ~c + a, ~d + b + c, ~e + c)
> dag1 <- newdag("a", c("b", "a"), c("c", "a"), c("d", "b", "c"), c("e", "c"))
> dag1
```

```
Directed graph
Nodes: a b c d e
Edges: b<-a c<-a d<-b d<-c e<-c
```

Here `~a` means that “a” has no parents while `~d+b+c` means that “d” has parents “b” and “c”.

Graphs are displayed with `plot`:

```
> plot(dag1)
```



3 Operations on undirected graphs

3.1 Graph queries

Many features of a graph are obtained by asking queries using the `queryg` function:

3.1.1 Nodes

```
> queryg(ug1, "nodes")
```

```
a b c d e f g
```

3.1.2 Edges

```
> queryg(ug1, "edges")
```

```
a b
a c
b c
c d
d e
f g
```

3.1.3 Cliques

```
> queryg(ug1, "cliques")
```

```
c b a
c d
e d
f g
```

3.1.4 Connected components

```
> queryg(ug1, "concomp")
```

```
a b c d e
f g
```

3.1.5 Closure

```
> queryg(ug1, "cl", "c")
```

```
c a b d
```

3.1.6 Adjacencies

```
> queryg(ug1, "adj", "c")
```

```
a b d
```

3.1.7 Simplicial nodes

Nodes whose boundary is complete.

```
> queryg(ug1, "simplicialNodes")
```

```
a b e f g
```

3.1.8 Is complete

Is the graph complete?

```
> queryg(ug1, "is.complete")
```

```
[1] FALSE
```

3.1.9 Is simplical

Is a node/set simplical?

```
> queryg(ug1, "is.simplicial", "a")
```

```
[1] TRUE
```

```
> queryg(ug1, "is.simplicial", c("a", "b", "d"))
```

```
[1] FALSE
```

3.1.10 Is triangulated

```
> queryg(ug1, "is.triangulated")
```

```
[1] TRUE
```

3.1.11 Is A and B separated by S

```
> queryg(ug1, "separates", c("a", "b"), c("e", "f"), "d")
```

```
[1] TRUE
```

3.1.12 Subgraph

```
> queryg(ug1, "subgraph", c("a", "b", "c"))
```

```
Undirected graph
Nodes: a b c
Edges: c~b c~a b~a
```

3.2 Adjancency matrix

```
> convertg(ug1, to = "matrix")
```

```
      a     b     c     d     e     f     g
a FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
b  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE
c  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE
d FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE
e FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
f FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
g FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
```

3.3 Triangulation and Maximum Cardinality Search

3.3.1 Maximum cardinality search

Testing for whether a graph is triangulated is based on Maximum Cardinality Search:

```
> g <- newug(~a + b, ~b + c, ~c + d, ~d + e, ~e + a)
> mcs(g)
```

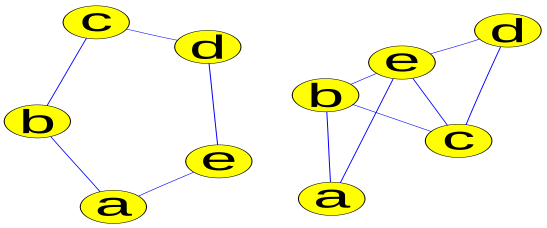
```
NULL
```

3.3.2 Triangulation

```
> tg <- triangulate(g)
> tg
```

```
Undirected graph
Nodes: a b c d e
Edges: a~b b~c c~d a~e b~e c~e d~e
```

```
> par(mfrow = c(1, 2))
> plot(g)
> plot(tg)
```



3.3.3 RIP ordering of the cliques

A RIP ordering of the cliques of a triangulated graph:

```
> rip <- ripOrder(tg)
> names(rip)
```

```
nodes cliques separators pa nLevels ch
```

```
> rip
```

```
Cliques
  1 a b e
  2 b e c
  3 e c d
Separators
  1 NA
  2 b e
  3 e c
Parents
  1 NA
  2 1
  3 2
```

4 Operations on directed acyclic graphs

4.1 Graph queries

Many features of a graph are obtained by asking queries using the `queryg` function as above:

4.1.1 Parents

```
> queryg(dag1, "pa", "d")
```

```
b c
```

4.1.2 Children

```
> queryg(dag1, "ch", "c")
```

```
d e
```

4.1.3 Ancestral set

```
> queryg(dag1, "ancestralSet", c("b", "e"))
```

```
a b c e
```

4.2 Moralization

```
> moralize(dag1)
```

```
Undirected graph
Nodes: a b c d e
Edges: a~b a~c b~c b~d c~d c~e
```

4.3 Ancestral graph

```
> ancestralGraph(dag1, c("b", "e"))
```

```
Directed graph
Nodes: a b c e
Edges: e<-c c<-a b<-a
```

4.4 Checking for acyclicity

If a directed graph contains cycles, then NULL is returned

```
> newdag(~a + b, ~b + c, ~c + a)
```

```
NULL
```