Graphs in the \texttt{gRbase} package

Søren Højsgaard

March 11, 2013

Contents

1 Introduction 1

2 Graphs 2
  2.1 Undirected graphs 2
  2.2 Directed acyclic graphs (DAGs) 4
  2.3 Graph coercion 5
  2.4 Plotting graphs 6

3 Advanced graph operations 7
  3.1 Moralization 7
  3.2 Topological sort 8
  3.3 Getting cliques 9
  3.4 Maximum cardinality search 10
  3.5 Triangulation 11
  3.6 RIP ordering / junction tree 12
  3.7 Minimal triangulation and maximum prime subgraph decomposition 13

4 Time and space considerations 13
  4.1 Time 13
  4.2 Space 14

5 Graph queries 15

1 Introduction

For the \texttt{R} community, the packages \texttt{graph}, \texttt{RBGL}, \texttt{Rgraphviz} and \texttt{igraph} are extremely useful tools for graph operations, manipulation and layout. The \texttt{gRbase} package adds some additional tools to these fine packages. The most important tools are:
1. Undirected and directed acyclic graphs can be specified using formulae or an adjacency list using the functions \texttt{ug()} and \texttt{dag()}. This gives graphs represented in one of the following forms:\footnote{There is a fourth form: \texttt{igraph} objects. These, however, will probably not be supported in the future.}

- A \texttt{graphNEL} object (the default),
- A dense adjacency matrix. By this we mean a “standard” matrix in \texttt{R}.
- A sparse adjacency matrix. By this we mean a \texttt{dgCMatrix} from the \texttt{Matrix} package.

2. Some graph algorithms are implemented in \texttt{gRbase}. These can be applied to graphs represented as \texttt{graphNEL}s and matrices. The most important algorithms are:

- \texttt{moralize()}, (moralize a directed acyclic graph)
- \texttt{mcs()}, (maximum cardinality search for undirected graph)
- \texttt{triangulate()}, (triangulate undirected graph)
- \texttt{rip()}, (RIP ordering of cliques of triangulated undirected graph)
- \texttt{getCliques()}, (get the (maximal) cliques of an undirected graph)
- \texttt{minimalTriang()} (minimal triangulation of undirected graph)\footnote{Needs more work}
- \texttt{mpd()} (maximal prime subgraph decomposition of undirected graph)\footnote{Needs more work}

The general scheme is the following: There is a \texttt{mcs()} function and \texttt{mcs()} methods for \texttt{graphNEL}s and for the two matrix types. The workhorse is the function \texttt{mcsMAT()} and the various methods coerces the graph to a (sparse) matrix and invokes \texttt{mcsMAT()}. 

2 Graphs

Undirected graphs can be created by the \texttt{ug()} function and directed acyclic graphs (DAGs) by the \texttt{dag()} function.

The graphs can be specified either using formulae or a list of vectors; see examples below.

2.1 Undirected graphs

An undirected graph is created by the \texttt{ug()} function.
As graphNEL: The following specifications are equivalent (notice that ":" and "*" can be used interchangably):

```
R> ug11 <- ug(~a:b:c + c:d + d:e + a:e + f:g)
R> ug11 <- ug(~a*b*c + c*d + d*e + a*e + f*g)
R> ug12 <- ug(c("a","b","c"),c("c","d"),c("d","e"),c("a","e"),c("f","g"))
R> ug13 <- ug(~a:b:c, ~c:d, ~d:e + a:e + f:g)
R> ug13 <- ug(~a*b*c, ~c*d, ~d*e + a*e + f*g)
R> ug11
```

A graphNEL graph with undirected edges
Number of Nodes = 7
Number of Edges = 7

As adjacency matrix: A representation as an adjacency matrix can be obtained with one of the following equivalent specifications:

```
R> ug11m <- ug(~a*b*c + c*d + d*e + a*e + f*g, result="matrix")
R> ug12m <- ug(c("a","b","c"),c("c","d"),c("d","e"),c("a","e"),c("f","g"),
                result="matrix")
R> ug11m
```

```
     a b c d e f g
a 0 1 1 0 1 0 0
b 1 0 1 0 0 0 0
c 1 1 0 1 0 0 0
d 0 0 1 0 1 0 0
e 1 0 0 1 0 0 0
f 0 0 0 0 0 0 1
g 0 0 0 0 0 1 0
```

```
R> ug11M <- ug(~a*b*c + c*d + d*e + a*e + f*g, result="Matrix")
R> ug12M <- ug(c("a","b","c"),c("c","d"),c("d","e"),c("a","e"),c("f","g"),
                result="Matrix")
R> ug11M
```

```
    a b c d e f g
  a 0 . . . . .
b . 1 . . . .
c . 1 . . . .
d . . 1 . . .
e . . 1 . . .
f . . . . . . 1
g . . . . . 1 .
```

7 x 7 sparse Matrix of class "dgCMatrix"
  a b c d e f g
a 1 . . . . .
b . . . . . .
c . . . . . .
d . . . . . .
e . . . . . .
f . . . . . . 1
g . . . . . 1 .
2.2 Directed acyclic graphs (DAGs)

A directed acyclic graph is created by the `dag()` function.

**As graphNEL:** The following specifications are equivalent (notice that "-" and "*" can be used interchangably):

\[
\begin{align*}
R & \text{> } \text{dag11 <- dag("a + b:a + c:a:b + d:c:e + e:a + g:f")}
R & \text{> } \text{dag11 <- dag("a + b*a + c*a*b + d*c*e + e*a + g*f")}
R & \text{> } \text{dag12 <- dag("a", c("b","a"), c("c","a","b"), c("d","c","e"), c("e","a"),c("g","f"))}
R & \text{> } \text{dag13 <- dag("a", "b*a", "c*a*b", "d*c*e", "e*a", "g*f")}
R & \text{> } \text{dag11}
\end{align*}
\]

A graphNEL graph with directed edges
Number of Nodes = 7
Number of Edges = 7

Here -a means that "a" has no parents while -d:b:c means that "d" has parents "b" and "c".

**As adjacency matrix:** A representation as an adjacency matrix can be obtained with

\[
\begin{align*}
R & \text{> } \text{dag11m <- dag("a + b:a + c:a:b + d:c:e + e:a + g:f", result="matrix")}
R & \text{> } \text{dag12m <- dag("a", c("b","a"), c("c","a","b"), c("d","c","e"), c("e","a"),c("g","f"), result="matrix")}
R & \text{> } \text{dag11m}
\end{align*}
\]

\[
\begin{array}{cccccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{g} & \text{f} \\
\text{a} & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
\text{b} & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\text{c} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\text{d} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{e} & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\text{g} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{f} & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

\[
\begin{align*}
R & \text{> } \text{dag11M <- dag("a + b:a + c:a:b + d:c:e + e:a + g:f", result="Matrix")}
R & \text{> } \text{dag12M <- dag("a", c("b","a"), c("c","a","b"), c("d","c","e"), c("e","a"),c("g","f"), result="Matrix")}
R & \text{> } \text{dag11M}
\end{align*}
\]
2.3 Graph coercion

Graphs can be coerced between different representations using `as()`; for example

```r
R> as(ug11, "matrix")
    a b c d e f g
a 0 1 1 0 1 0 0
b 1 0 1 0 0 0 0
c 1 1 0 1 0 0 0
d 0 0 1 0 1 0 0
e 1 0 0 1 0 0 0
f 0 0 0 0 0 1 0
g 0 0 0 0 0 1 0

R> as(as(ug11, "matrix"), "dgCMatrix")

7 x 7 sparse Matrix of class "dgCMatrix"
  a b c d e f g
a . 1 1 . 1 . .
b . 1 . . . . .
c . . 1 . . . .
d . . . 1 . . .
e . . 1 . . . .
f . . . . 1 . .
g . . . . . 1 .

R> as(as(as(ug11, "matrix"), "Matrix"), "graphNEL")
A graphNEL graph with undirected edges
Number of Nodes = 7
Number of Edges = 7

R> as(as(as(as(ug11, "matrix"), "Matrix"), "graphNEL"), "igraph")
IGRAPH UNW- 7 7 --
+ attr: name (v/c), label (v/c), weight (e/n)
NOTICE! There is one thing to notice when coercing a dense matrix to a sparse matrix. Consider this

\begin{verbatim}
R> m <- matrix(1:4,nrow=2)
R> as(m, "Matrix")
\end{verbatim}

\begin{verbatim}
2 x 2 Matrix of class "dgeMatrix"
 [,1] [,2]
[1,] 1 3
[2,] 2 4
\end{verbatim}

\begin{verbatim}
R> as(m, "dgCMatrix")
\end{verbatim}

\begin{verbatim}
2 x 2 sparse Matrix of class "dgCMatrix"
[,1] [,2]
[1,] 1 3
[2,] 2 4
\end{verbatim}

In the first case, the matrix coercion method will, based on properties of the matrix, impose a specific type on the result. The graph algorithms in gRbase are based on the dgCMatrix form and hence the latter case is more safe. Coercion to a dgCMatrix can be made much faster with the following function from gRbase:

\begin{verbatim}
R> asdgCMatrix(m)
\end{verbatim}

\begin{verbatim}
2 x 2 sparse Matrix of class "dgCMatrix"
[,1] [,2]
[1,] 1 3
[2,] 2 4
\end{verbatim}

### 2.4 Plotting graphs

Graphs represented as graphNEL objects are displayed with plot(). There is no plot() method for graphs represented as adjacency matrices, so here coercion is one option:

\begin{verbatim}
R> par(mfrow=c(1,2))
R> plot(ug11)
R> plot(as(ug11m,"graphNEL"))
\end{verbatim}
An alternative for adjacency matrices is the `gplot()` function in the `sna` package:

```r
R> par(mfrow=c(1,2))
R> library(sna)
R> gplot(ug11m, label=colnames(ug11m), gmode="graph")
R> gplot(dag11m, label=colnames(dag11m))
```

### 3 Advanced graph operations

#### 3.1 Moralization

```r
R> apropos("\^moralize\")

[1] "moralize.Matrix" "moralize.graphNEL" "moralize.igraph"
[4] "moralize.matrix"
```

A moralized directed acyclic graph is obtained with

```r
R> dag11.mor <- moralize(dag11)
```

```r
R> par(mfrow=c(1,2))
R> plot(dag11)
R> plot(dag11.mor)
```
For the alternative representations

```r
R> moralize(dag11m)

      a b c d e g f
     a 0 1 1 0 1 0 0
     b 1 0 1 0 0 0 0
     c 1 1 0 1 1 0 0
     d 0 0 1 0 1 0 0
     e 1 0 1 1 0 0 0
     g 0 0 0 0 0 0 1
     f 0 0 0 0 0 1 0
```

```r
R> moralize(dag11M)

   7 x 7 sparse Matrix of class "dgCMatrix"
     a b c d e g f
     a . 1 1 . 1 .
     b 1 . 1 . . .
     c 1 1 . 1 . .
     d . 1 . 1 . .
     e 1 . 1 . . .
     g . . . . . . 1
     f . . . . . 1 .
```

### 3.2 Topological sort

A topological ordering of a directed graph is a linear ordering of its vertices such that, for every edge $(u \rightarrow v)$, $u$ comes before $v$ in the ordering. A topological ordering is possible if and only if the graph has no directed cycles, that is, if it is a directed acyclic graph (DAG). Any DAG has at least one topological ordering.

```r
R> topoSort(dag11)
```
The `dag()` function actually allows specification of a directed graph with cycles (a check can be imposed with `forceCheck=TRUE`). Below is a directed graph with a cycle:

```R
R> topoSort(dag(~a:b+b:c+c:a))
character(0)
```

### 3.3 Getting cliques

In graph theory, a clique is often a complete subset of a graph. A maximal clique is a clique which can not be enlarged. In statistics (and that is the convention we follow here) a clique is usually understood to be a maximal clique.

Finding the cliques of a general graph is an NP complete problem. Finding the cliques of triangulated graph is linear in the number of cliques.

```R
R> str(getClique(ug11))
List of 5
$ : chr [1:3] "a" "b" "c"
$ : chr [1:2] "a" "e"
$ : chr [1:2] "d" "c"
$ : chr [1:2] "d" "e"
$ : chr [1:2] "f" "g"
```

```R
R> str(getClique(ug11m))
List of 5
$ : chr [1:3] "a" "b" "c"
$ : chr [1:2] "a" "e"
$ : chr [1:2] "d" "c"
$ : chr [1:2] "d" "e"
$ : chr [1:2] "f" "g"
```

```R
R> str(getClique(ug11M))
List of 5
$ : chr [1:3] "a" "b" "c"
$ : chr [1:2] "a" "e"
$ : chr [1:2] "d" "c"
$ : chr [1:2] "d" "e"
$ : chr [1:2] "f" "g"
```
3.4 Maximum cardinality search

R> apropos("^mcs\\.")

[1] "mcs.Matrix" "mcs.graphNEL" "mcs.igraph" "mcs.matrix"

Testing for whether a graph is triangulated is based on Maximum Cardinality Search. If character(0) is returned the graph is not triangulated. Otherwise a linear ordering of the nodes is returned.

R> mcs(ug11)

character(0)

R> mcs(ug11m)

character(0)

R> mcs(ug11M)

character(0)

R> mcs(dag11.mor)

[1] "a" "b" "c" "e" "d" "g" "f"

R> mcs(as(dag11.mor,"matrix"))

[1] "a" "b" "c" "e" "d" "g" "f"

R> mcs(as(dag11.mor,"Matrix"))

[1] "a" "b" "c" "e" "d" "g" "f"

R> mcs(dag11)

character(0)
3.5 Triangulation

R> apropos("^triangulate\.")

[1] "triangulate.Matrix" "triangulate.graphNEL" "triangulate.igraph"
[4] "triangulate.matrix"

Triangulate an undirected graph by adding extra edges to the graph:

R> (tug11<-triangulate(ug11))

A graphNEL graph with undirected edges
Number of Nodes = 7
Number of Edges = 8

R> (tug11m<-triangulate(ug11m))

    a b c d e f g
a 0 1 1 0 1 0 0
b 1 0 1 0 0 0 0
c 1 1 0 1 1 0 0
d 0 0 1 0 1 0 0
e 1 0 1 1 0 0 0
f 0 0 0 0 0 0 1
g 0 0 0 0 0 1 0

R> (tug11M<-triangulate(ug11M))

7 x 7 sparse Matrix of class "dgCMatrix"

    a b c d e f g
a . 1 1 . 1 .
b 1 . . . .
c 1 1 . 1 1 .
d . 1 . 1 .
e 1 . 1 1 .
f . . . . . 1
g . . . . 1 .

R> par(mfrow=c(1,2))
R> plot(ug11)
R> plot(tug11)
3.6 RIP ordering / junction tree

R> apropos("rip\\.")

[1] "rip.Matrix" "rip.graphNEL" "rip.igraph" "rip.matrix"

A RIP ordering of the cliques of a triangulated graph can be obtained as:

R> rr <- rip(tug11)
R> rr

cliques
  1 : c a b
  2 : e a c
  3 : d c e
  4 : g f
separators
  1 :
  2 : a c
  3 : c e
  4 :
parents
  1 : 0
  2 : 1
  3 : 2
  4 : 0

R> rr <- rip(tug11m)
R> rr <- rip(tug11M)

R> plot(rr)
3.7 Minimal triangulation and maximum prime subgraph decomposition

An undirected graph uG is triangulated (or chordal) if it has no cycles of length \( \geq 4 \) without a chord which is equivalent to that the vertices can be given a perfect ordering. Any undirected graph can be triangulated by adding edges to the graph, so called fill-ins which gives the graph TuG. A triangulation TuG is minimal if no fill-ins can be removed without breaking the property that TuG is triangulated. A related concept is the minimum triangulation, which is the graph with the smallest number of fill-ins. The minimum triangulation is unique. Finding the minimum triangulation is NP-hard.

4 Time and space considerations

4.1 Time

It is worth noticing that working with graphs represented as graphNEL objects is somewhat slower working with graphs represented as adjacency matrices. Consider finding the cliques of an undirected graph represented as a graphNEL object or as a matrix:

\[
R \> \text{system.time(}\{\text{for (ii in 1:200) maxClique(ug11)}\}) \quad \text{## in RBGL}
\]

\[
\begin{array}{ccc}
\text{user} & \text{system} & \text{elapsed} \\
0.11 & 0.00 & 0.11
\end{array}
\]

\[
R \> \text{system.time(}\{\text{for (ii in 1:200) maxCliqueMAT(ug11m)}\}) \quad \text{## in gRbase}
\]

\[
\begin{array}{ccc}
\text{user} & \text{system} & \text{elapsed} \\
0.03 & 0.00 & 0.04
\end{array}
\]
Working with sparse matrices rather than standard matrices slows indexing down:

```r
R> system.time({for (ii in 1:2000) ug11m[2,]})

user  system elapsed
0      0       0

R> system.time({for (ii in 1:2000) ug11M[2,]})

user  system elapsed
0.77  0.00    0.77
```

However, `gRbase` has some functionality for indexing sparse matrices quickly:

```r
R> system.time({for (ii in 1:2000) sp_getXj(ug11M,2)})

user  system elapsed
0.04  0.00    0.04
```

### 4.2 Space

The `graphNEL` representation is – at least – in principle more economic in terms of space requirements than the adjacency matrix representation (because the adjacency matrix representation uses a 0 to represent a “missing edge”. The sparse matrix representation is clearly only superior to the standard matrix representation if the graph is sparse:

```r
R> V <- 1:100
R> M <- 1:10
R> ## Sparse graph
R> ##
R> g1 <- randomGraph(V, M, 0.05)
R> length(edgeList(g1))

[1] 147

R> c(NEL=object.size(g1),
   mat=object.size(as(g1, "matrix")),
   Mat=object.size(as.adjMAT(g1, "Matrix")))

   NEL    mat   Mat
157336  51648  16632

R> ## More dense graph
R> ##
R> g1 <- randomGraph(V, M, 0.5)
R> length(edgeList(g1))
```
5 Graph queries

The graph and RBGL packages implement various graph operations for graphNEL objects. See the documentation for these packages. The gRbase implements a few additional functions, see Section 1. An additional function in gRbase for graph operations is querygraph(). This function is intended as a wrapper for the various graph operations available in gRbase, graph and RBGL. There are two main virtues of querygraph(): 1) querygraph() operates on any of the three graph representations described above\(^4\) and 2) querygraph() provides a unified interface to the graph operations. The general syntax is

\( \text{R> args(querygraph)} \)

function (object, op, set = NULL, set2 = NULL, set3 = NULL)
NULL

\(^4\)Actually not quite yet, but it will be so in the future.