other software systems. Biological metadata (for example, genome annotations) need to be tightly integrated with the analysis of primary data, and the Bioconductor project has invested a lot of effort in the provision of high quality metadata packages. The experimental data in functional genomics require more structured formats than the basic data types of R, and one of the main products of the Bioconductor core is the provision of common data structures that allow the efficient exchange of data and computational results between different packages. One example is the `ExpressionSet`, an S4 class for the storage of the essential data and information on a microarray experiment.

The articles in this issue span a wide range of topics. Common themes are *preprocessing* (data import, quality assessment, standardization, error modeling and summarization), *pattern discovery and detection*, and higher level statistical models with which we aim to gain insight into the underlying biological processes. Sometimes, the questions that we encounter in bioinformatics result in methods that have potentially a wider applicability; this is true in particular for the first three articles with which we start this issue.

Wolfgang Huber
European Bioinformatics Insitute (EBI)
European Molecular Biology Laboratory (EMBL) Cambridge, UK
huber@ebi.ac.uk

Graphs and Networks: Tools in Bioconductor

by Li Long and Vince Carey

Introduction

Network structures are fundamental components for scientific modeling in a number of substantive domains, including sociology, ecology, and computational biology. The mathematical theory of graphs comes immediately into play when the entities and processes being modeled are clearly organized into objects (modeled as graph nodes) and relationships (modeled as graph edges).

Graph theory addresses the taxonomy of graph structures, the measurement of general features of connectedness, complexity of traversals, and many other combinatorial and algebraic concepts. An important generalization of the basic concept of graph (traditionally defined as a set of nodes N and a binary relation E on N defining edges) is the hypergraph (in which edges are general subsets of the node set of cardinality at least 2).

The basic architecture of the Bioconductor toolkit for graphs and networks has the following structure:

- Representation infrastructure: packages **graph**, **hypergraph**;
- Algorithms for traversal and measurement: packages **RBGL**, **graphPart**
- Algorithms for layout and visualization: package **Rgraphviz**; **RBGL** also includes some layout algorithms;
- Packages for addressing substantive problems in bioinformatics: packages **pathRender**, **GraphAT**, **ScISI**, **GOstats**, **ontoTools**, and others.

In this article we survey aspects of working with network structures with some of these Bioconductor tools.

The basics: package graph

The **graph** package provides a variety of S4 classes representing graphs. A virtual class `graph` defines the basic structure:

```
> library(graph)
> getClass("graph")
Virtual Class

Slots:

Name:   edgemode  edgeData  nodeData
Class: character  attrData  attrData

Known Subclasses: "graphNEL", "graphAM",
   "graphH", "distGraph", "clusterGraph",
   "generalGraph"
```

A widely used concrete extension of this class is `graphNEL`, denoting the "node and edge list" representation:

```
> getClass("graphNEL")

Slots:

Name:      nodes     edgeL  edgemode  edgeData
Class:    vector      list character  attrData
```

```
        attrData
        nodeData

Extends: "graph"
```

An example graph is supplied, representing the "integrin mediated cell adhesion" pathway as defined in a previous version of KEGG (Kyoto Encyclopedia of Genes and Genomes, Kanehisa and Goto, 2000; the pathway is now called "focal adhesion").

```
> data(integrinMediatedCellAdhesion)
> IMCAGraph
A graphNEL graph with directed edges
Number of Nodes = 52
Number of Edges = 91
> nodes(IMCAGraph)[1:5]
[1] "ITGB" "ITGA" "ILK"  "CAV"  "SHC"
> edges(IMCAGraph)[1:5]
$ITGB
[1] "ITGA" "ILK"  "CAV"  "SHC"  "ACTN" "TLN"

$ITGA
[1] "ITGB"

$ILK
[1] "ITGB"

$CAV
[1] "ITGB"

$SHC
[1] "FYN"  "GRB2" "ITGB"
```

Methods that can be used to work with graph instances include:

- The `edgemode` method returns a character token indicating whether the graph is directed or undirected.
- The `nodes` method returns information on graph nodes; in the case of a `graphNEL` instance it returns a vector, intended to represent the (unordered) node set through character names of entities described by the graph. (Note that there are more general representations available, in which objects other than strings can constitute graph nodes, see remarks on `nodeData` below.)
- The `edges` method returns a named list. The *i*th element of the list bears as its name the name of the *i*th node, and contains a vector of node names to which the *i*th node is directly linked by a graph edge.

```
> all.equal(names(edges(IMCAGraph)),
     nodes(IMCAGraph))
[1] TRUE
```

The network may be visualized using the following code:

```
library(Rgraphviz)
plot(IMCAGraph,
    attrs = IMCAAttrs$defAttrs,
    nodeAttrs = IMCAAttrs$nodeAttrs,
    subGList = IMCAAttrs$subGList);
```

See Figure 1.

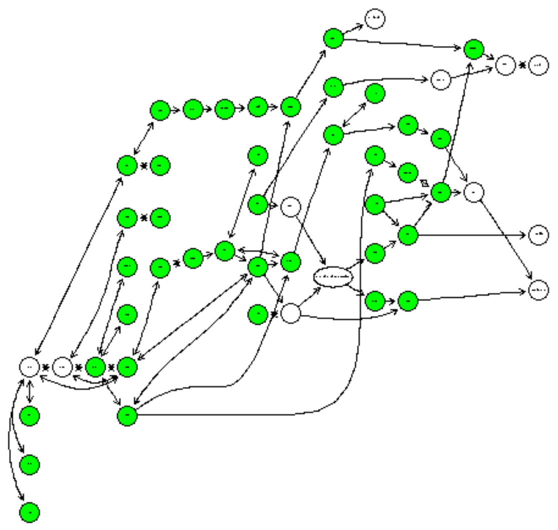

Figure 1: Rendering of the IMCA pathway.

We will illustrate aspects of other parts of the system using this example graph structure.

The Rgraphviz package

Rgraphviz provides an interface to graphviz[1], an open source library for graph visualization. Graphviz performs several types of graph layout:

- dot: draws directed graphs in a hierarchical way
- neato: draws graphs using Kamada-Kawai algorithm
- fdp: draws graphs using Fruchterman-Reingold heuristic
- twopi: draws radial layout
- circo: draws circular layout

Many aspects of graph rendering are covered, with general programmatic control over colors, fonts, shapes, and line styles employed for various aspects of the display and annotation of the graph.

The Bioconductor **Rgraphviz** interface package lets you choose the layout engine for your graph object among the options `dot` (the default), `neato` and `twopi`.

[1] http://www.graphviz.org

We generate a graph and plot it using different layout engines:

```
> library("Rgraphviz")
> set.seed(123)
> V <- letters[1:10]
> M <- 1:4
> g1 <- randomGraph(V, M, 0.2)
A graphNEL graph with undirected edges
Number of Nodes = 10
Number of Edges = 16

> plot(g1)
```

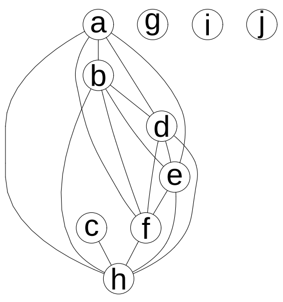

Figure 2: Rendering with dot.

```
> plot(g1, "neato")
```

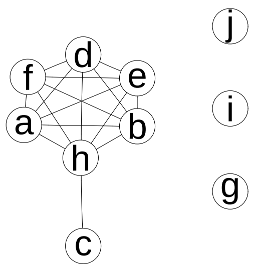

Figure 3: Rendering with neato.

```
> plot(g1, "twopi")
```

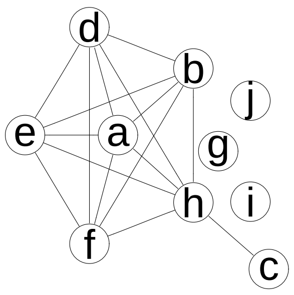

Figure 4: Rendering with twopi.

The RBGL package

Researchers study different kinds of biological networks: protein interaction networks, genetic regulatory systems, metabolic networks are a few examples. Signaling pathway identification could be modeled using shortest paths in a protein interaction network. The task of identification of protein complexes can be conducted by finding cohesive subgroups in protein interaction networks. The protein interaction network under consideration is built from the experimental data, which contains incomplete information, false positives and false negatives. Extensible representations that can manage such complications are at a premium in this process.

RBGL provides a set of graph algorithms to analyse the structures and properties of graphs and networks.

There are three categories of algorithms in this package:

- interfaces to graph algorithms from the boost[2] graph library (BGL[3]);
- algorithms built on BGL; and
- other algorithms.

The Boost C++ Library provides peer-reviewed C++ libraries, with the objective of creating highly portable open source resources for C++ programming analogous to the proprietary standard template library (STL).

The Boost Graph Library (BGL) is one of the Boost libraries. BGL provides a set of generic data structures to represent graphs, and a collection of C++ modules (all defined as templates) constituting algorithms on graphs.

Package **RBGL** provides a direct interface to many of the graph algorithms available in BGL. Table 1 lists the main functionalities currently provided.

In the second category, we implemented minimum-cut algorithm `minCut`, based on one of the maximum-flow algorithms from BGL.

In the third category, we implemented

- highly connected subgraphs (function `highlyConnSG`), a clustering algorithm proposed by Hartuv and Shamir (2000).
- a number of algorithms from social network analysis: k-cliques (function `kCliques`), k-cores (function `kCores`), maximum cliques (function `maxClique`), lambda-sets (function `LambdaSets`), and
- predicate functions: to decide if a graph is triangulated (function `is.triangulated`), to test if a subset of nodes separates two other subsets of nodes (function `separates`).

[2]http://www.boost.org
[3]http://www.boost.org/libs/graph

RBGL functions	Comments
Traversals	
`bfs`	breadth-first search
`dfs`	depth-first search
Shortest paths	
`dijkstra.sp`	Single-source, nonnegative weights
`bellman.ford.sp`	Single-source, general weights
`dag.sp`	Single-source, DAG
`floyd.warshall.-all.pairs.sp`	Returns distance matrix
`johnson.all.pairs.sp`	Returns distance matrix
Minimal spanning trees	
`mstree.kruskal`	Returns edge list and weights
`prim.minST`	As above
Connectivity	
`connectedComp`	Returns list of node-sets
`strongComp`	As above
`articulationPoints`	As above
`biConnComp`	As above
`edgeConnectivity`	Returns index and minimum disconnecting set
`init.incremental.-components`	Special processing for evolving graphs
`incremental.components`	
`same.component`	Boolean in the incremental setting
Maximum flow algorithms	
`edmunds.karp.max.flow`	List of max flow, and edge-specific flows
`push.relabel.max.flow`	
Vertex ordering	
`tsort`	Topological sort
`cuthill.mckee.ordering`	Reduces bandwidth
`sloan.ordering`	Reduces wavefront
`min.degree.ordering`	Heuristic
Other functions	
`transitive.closure`	Returns from-to matrix
`isomorphism`	Boolean
`sequential.vertex.coloring`	Returns color no for nodes
`brandes.betweenness.-centrality`	Indices and dominance measure
`circle.layout`	Returns vertex coordinates
`kamada.kawai.spring.layout`	Returns vertex coordinates

Table 1: Names of key functions in **RBGL**. Working examples for all functions are provided in the package manual pages.

To illustrate an application, we use the sp.between function to determine the shortest path in the network between the SRC gene and the cell proliferation process:

```
> sp.between(IMCAGraph, "SRC",
          "cell proliferation")
$‘SRC:cell proliferation‘
$‘SRC:cell proliferation‘$path
[1] "SRC"                "FAK"
[3] "MEK"                "ERK"
[5] "cell proliferation"

$‘SRC:cell proliferation‘$length
[1] 4

$‘SRC:cell proliferation‘$pweights
               SRC->FAK
                      1
               FAK->MEK
                      1
               MEK->ERK
                      1
ERK->cell proliferation
                      1
```

The resulting list includes a vector encoding the shortest path, its length, and the weights of individual steps through the graph.

Representations and general attributes

As noted in the class report for getClass("graph"), a variety of graph representations are available as S4 classes. It is particularly simple to work with graphAM on the basis of a binary adjacency matrix:

```
> am = diag(4)
> am = cbind(1,am)
> am = rbind(am,0)
> am[5,3] = am[3,5] = 1
> am[1,1] = 0
> dimnames(am) = list(letters[1:5],
                      letters[1:5])
> am
  a b c d e
a 0 1 0 0 0
b 1 0 1 0 0
c 1 0 0 1 1
d 1 0 0 0 1
e 0 0 1 0 0
> amg = new("graphAM", am,
          edgemode="directed")
> amg
A graphAM graph with directed edges
Number of Nodes = 5
Number of Edges = 9
> nodes(amg)
[1] "a" "b" "c" "d" "e"
> edges(amg)[2]
$b
[1] "a" "c"
```

Other classes can be investigated through examples in the manual pages.

To illustrate extensibility of graph component representations, we return to the IMCAGraph example. As noted above, the node set for a graphNEL instance is a character vector. We may wish to have additional attributes characterizing the nodes. In this example, we show how to allow a “long gene name” attribute to be defined on this instance.

First, we establish the name of the node attribute and give its default value for all nodes.

```
> nodeDataDefaults(IMCAGraph,
      attr="longname") <- as.character(NA)
```

Now we take a specific node and assign the new attribute value.

```
> nodeData(IMCAGraph, "SHC",
      attr="longname" ) =
+ paste("src homology 2 domain",
   "containing transforming protein")
```

To retrieve the attribute value, the nodeData accessor is used:

```
> nodeData(IMCAGraph, "SHC")
$SHC
$SHC$longname
[1] "src homology 2 domain containing
        transforming protein"
```

The hypergraph and graphPart packages

A hypergraph is a generalised graph, where a hyperedge is defined to represent a *subset* of nodes (in contrast to the edge in a basic graph, which is defined by a *pair* of nodes). This structure can be used to model one-to-many and many-to-many relationships. The package **hypergraph** provides an R class to represent hypergraphs.

A protein interaction network could be considered as a hypergraph: nodes are proteins, hyperedges are protein complexes, a node is connected to a hyperedge when the corresponding protein is a member of the corresponding protein complex.

A k-way partition of a hypergraph assigns the nodes to k disjoint non-empty sets so that a given cost function is minimum. A typical cost function is the weight sum of hyperedges that span over more than one such disjoint sets. Computing k-way partitions of hypergraphs is NP-hard.

There are several libraries available that provide approximate solutions to the above partition problem. The package **graphPart** provides interfaces to two such libraries:

- hMETIS[4]: a set of algorithms based on the multilevel hypergraph partitioning schemes developed in Karypis Lab at Univ. of Minnesota. These algorithms have shown to produce good results and are scalable.
- PaToH[5]: a multilevel hypergraph partitioning tool developed by U. Catalyurek in Ohio State Univ. It is fast and stable, and can handle partitioning with fixed cells and multi-contraints.

Note that these libraries are not open source, and must be obtained independently of the Bioconductor tools described here.

The pathRender package

The Cancer Molecular Analysis Project (cMAP) from NCICB[6] maintains a collection of genes organized by pathways and by ontology. The pathway interaction database is assembled from publicly available sources, represented in a formal, ontology-based manner. The pathways are modeled as directed graphs.

The data package **cMAP** contains annotated data from cMAP for use in bioconductor. For a given pathway, the package **pathRender** provides a tool to render this pathway as a whole or just a part of it.

pathRender does the following:

- it takes data from the cMAP database via package **cMAP**;
- it builds a graph where nodes represent molecules/proteins, edges represent interactions between them;
- it assigns different properties to the nodes and edges according to what they represent; then
- it uses package **Rgraphviz** to render such a graph on graphical display.

Outlook

For **RBGL**, we're planning to implement additional algorithms, such as algorithms for calculating clustering coefficient and transitivity of graphs, and computing *k*-cores of hypergraphs. Further experience is needed to establish more effective use of these algorithms in bioinformatic work flows.

Since bipartite graphs and hypergraphs are likely to play a substantial role, we will also introduce more specialized algorithms for these structures.

Acknowledgement

L.L. was supported by a grant from Intel Corp. to the Vital-IT Center.

Bibliography

E. Hartuv and R. Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(4–6):175–181, 2000. URL `citeseer.ist.psu.edu/hartuv99clustering.html`.

M. Kanehisa and S. Goto. KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Res*, 28: 27–30, 2000.

Li Long
Vital-IT Center
Swiss Institute of Bioinformatics
CH 1015 Lausanne
`Li.Long@isb-sib.ch`

Vincent J. Carey
Channing Laboratory
Brigham and Women's Hospital
Harvard Medical School
181 Longwood Ave.
Boston MA 02115, USA
`stvjc@channing.harvard.edu`

[4] `http://glaros.dtc.umn.edu/gkhome/views/metis`
[5] `http://bmi.osu.edu/~umit/software.html`
[6] `http://cmap.nci.nih.gov`