

# Graph workshop

- Graphs in R/bioconductor
- VJ Carey <stvjc@channing.harvard.edu>
- tools mainly on [www.bioconductor.org](http://www.bioconductor.org)
- contributions from R Gentleman, E Whalen, J Gentry, J Zhang gratefully acknowledged

# Overview

- graph class and methods (package *graph*)
  - scribbling a graph for general use: *dotty*, *dot2gxl*
  - validating GXL reading and writing
  - sparse encoding of adjacency matrix (package *sparseM*)
- *Rgraphviz* for layouts
  - working with a layout for a richly annotated graph
- *RBGL* for graph algorithms
- *Rggobi* application
- other packages and future directions –  
RDF/XML, network inference

# notes

- what is to be described is created mostly out of the labors of others
- such reuse is generally advantageous
- requires obedience to (modest) compatibility constraints
- aided by adoption of formal APIs
- require in general portability/licensing at the level of R/bioconductor – portability not quite met with *Rgraphviz*

# *graph* package/classes

```
> library(graph)
```

```
> print(slotNames("graph"))
```

```
[1] "edgemode"
```

```
> print(slotNames("graphNEL"))
```

```
[1] "nodes"      "edgeL"      "edgemode"
```

```
> print(class(dd))
```

```
[1] "graphNEL"
```

```
> print(dd)
```

```
A graph with directed edges
```

```
Number of Nodes = 5
```

```
Number of Edges = 9
```

```
> print(nodes(dd))
```

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

# edgeL structure

```
> print(names(slot(dd, "edgeL")))
```

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

```
> print(slot(dd, "edgeL")[1:2])
```

```
$A
```

```
$A$edges
```

```
[1] 3
```

```
$A$weights
```

```
[1] 1
```

```
$B
```

```
$B$edges
```

```
[1] 2 4 5
```

```
$B$weights
```

```
[1] 2 1 2
```

## S4/PWD (green book) formal classes/methods

	f	parmNames	sigClasses
1	acc	object	graph
2	acc	object	clusterGraph
3	addEdge	from, to, graph, weights	character, character, graphNEL, numeric
4	addNode	node, object	character, graphNEL
5	adj	object	graphNEL
6	adj	object	distGraph
7	adj	object	clusterGraph
8	bNode	object	gEdge
9	clearNode	node, object	character, graphNEL
10	combineNodes	nodes, graph, newName	character, graphNEL, character
11	complement	x	graph
12	connComp	object	graph
13	connComp	object	clusterGraph

# all generics

	1	2	3	4	5
X1	acc	addEdge	addNode	adj	bNode
X2	clearNode	combineNodes	complement	connComp	contents
X3	degree	dfs	Dist	dumpGXL	edgeID
X4	edgeL	edgemode	edgeOrder	edges	edgeWeights
X5	eNode	fromEdges	fromEdges<—	fromGXL	hash
X6	inEdges	intersection	isConnected	nodeID	nodes
X7	nodes<—	nodeType	numNodes	removeEdge	removeNode
X8	subGraph	threshold	toEdges	toEdges<—	toGXL
X9	union	validateGXL			

# scribbling a graph

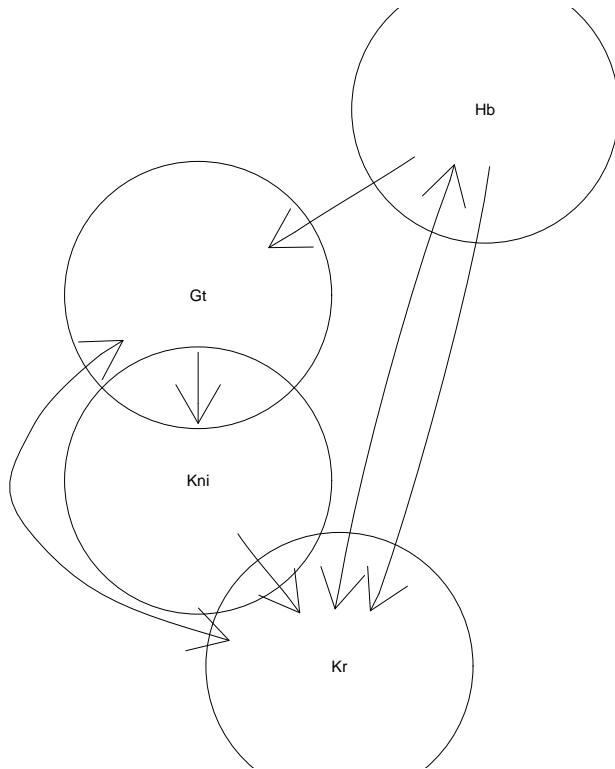
- dotty is a reasonable start for modestly complex graph with minimal annotation
- save as dot and use dot2gxl
- add edgeid attributes to all edges and graph::fromGXL will compute the corresponding R graphNEL (**news flash: dot2gxl now adds edge ids! thanks Emden!**)
- it is sometimes feasible to write GXL for simple graphs directly

# what is GXL?

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE gxl SYSTEM "file:/Users/chaouiya/GIN_sim/XML_Format/Oxygen/GINML.dtd">
<gxl xmlns:xlink="http://www.w3.org/1999/xlink">
  <graph id="gap">
    <node id="Gt" basevalue="1">
      <int>1</int>
      <parameter idActiveInteractions="KrGt" val="1"/>
    </node>
    <node id="Kr">
      <int>1</int>
      <parameter idActiveInteractions="HbKr1" val="1"/>
      ...
    </node>
    <edge from="Hb" to="Gt" id="HbGt" sign="negative">
      <int>1</int>
    </edge>
    <edge from="Hb" to="Kr" id="HbKr1" sign="positive">
      ...
    </edge>
  </graph>
</gxl>
```

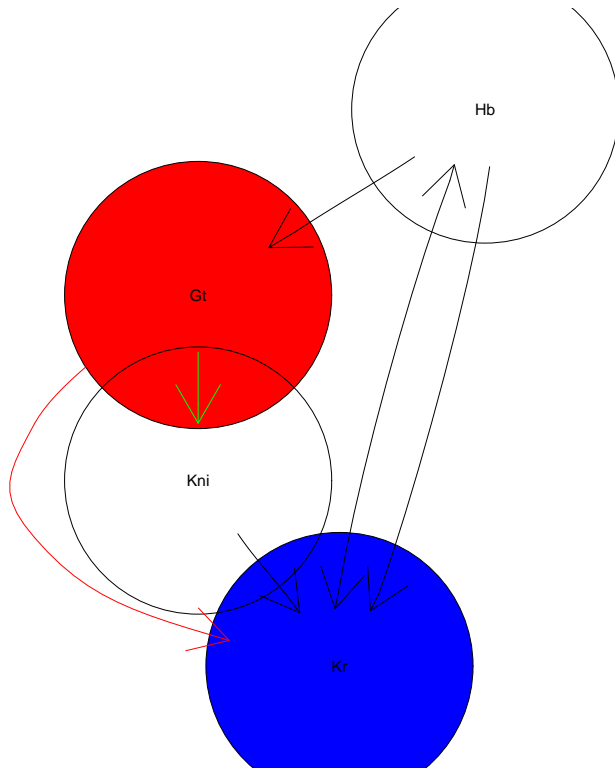
## result with Rgraphviz defaults (dot layout)

```
g <- fromGXL(file("gapA.gxl"))  
plot(g)
```



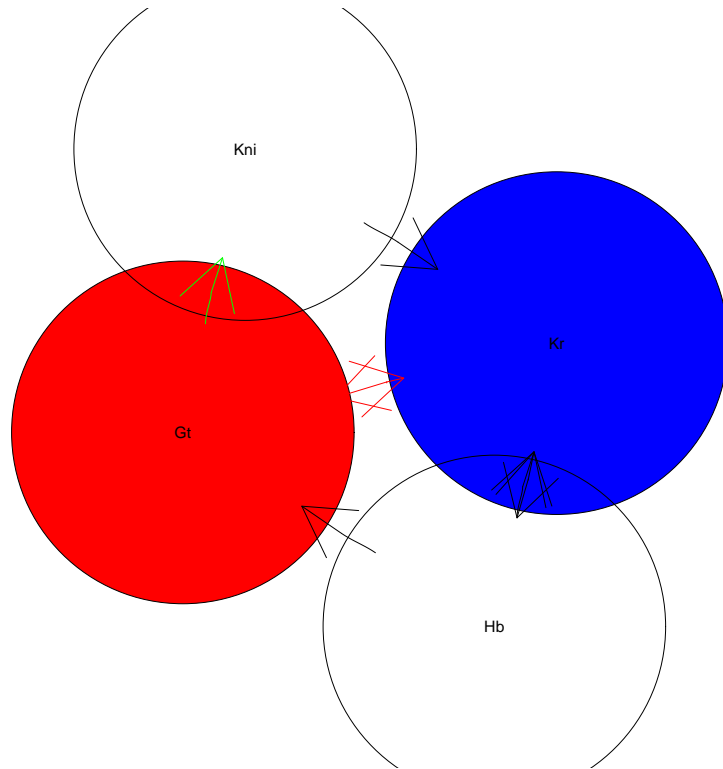
# colors

```
plot(g, nodeCols=c(Gr="red" ...,  
edgeCols=list(Gr=list(Kr="red" ...))
```



# spring layout

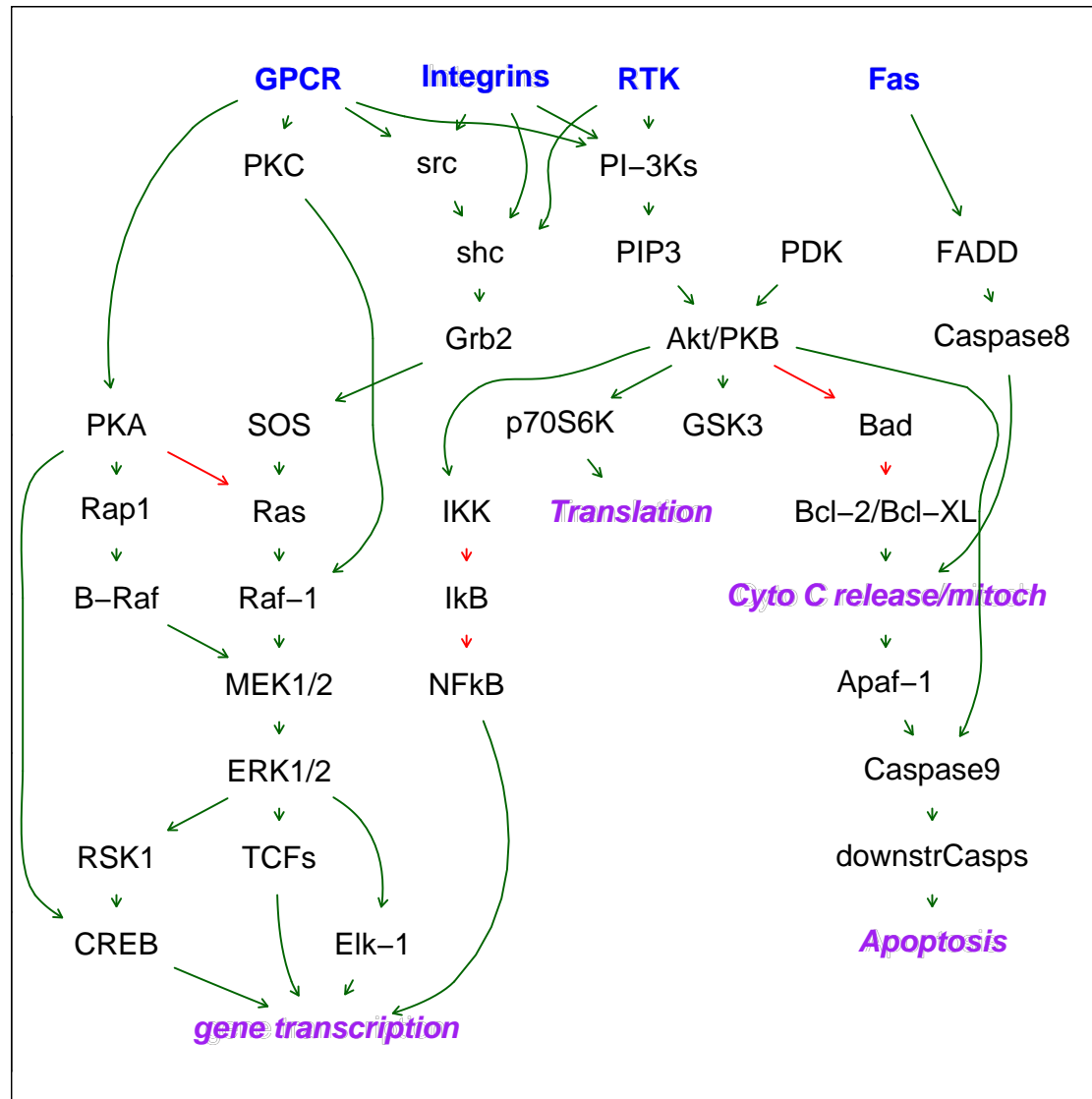
```
plot(g, "neato", nodeCols=c(Gr="red" ... ,  
    edgeCols=list(Gr=list(Kr="red" ...))
```



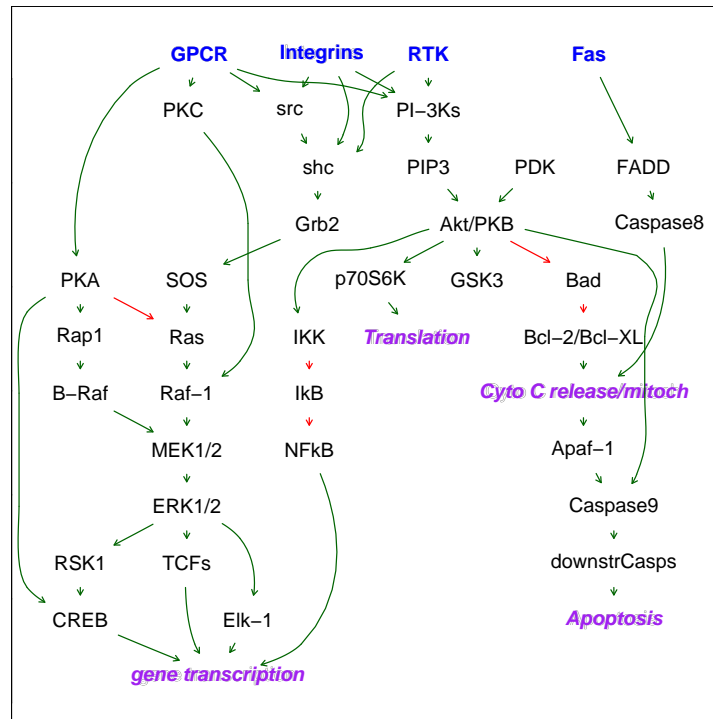
# going beyond defaults

- `Rgraphviz::plot` returns a structure that includes much of the internal graphviz layout information
- there are `text`, `points` and `lines` methods in `Rgraphviz` that can be used to control rendering based on the graphviz structure
- an example of the results follows
- key objective is to increase interactivity of R rendering for linked browsing between metadata and observational data; `ggobi` will be used in this connection later

# beyond defaults



## beyond defaults



- this particular graph
  - started in Sui Huang's contribution to Hofest ader/Collado-Vides monogr.
  - was manually translated to GXL (VC),
  - was read into R (Duncan),
  - plotted by Rgraphviz (Gviz, Jeff)
  - and the returned structure was used for text, points and lines commands in R

# Rgraphviz future plans

- exploit subgraph concept (introduce `clGraphNEL`)
- improve propagation of weights, edge labeling to `plot` method
- improve interactivity with plotted images
- catalog themes for graph visualizations, to support manufacture of large collections of similar visualizations

# exporting GXL

```
> cat(saveXML(toGXL(g)), sep = "\n")
```

```
<?xml version="1.0"?>  
<!DOCTYPE SYSTEM "http://www.gupro.de/GXL/gxl-1.0.1.dtd">  
<gxl>  
  <graph id="graphNEL">  
    <node id="Gt"/>  
    <node id="Kr"/>  
    <node id="Hb"/>  
    <node id="Kni"/>  
    <edge id="e1" from="Gt" to="Kr"/>  
    <edge id="e2" from="Gt" to="Kni"/>  
    <edge id="e3" from="Hb" to="Gt"/>  
    <edge id="e4" from="Hb" to="Kr"/>  
    <edge id="e5" from="Kni" to="Kr"/>  
    <edge id="e6" from="Kr" to="Gt"/>  
    <edge id="e7" from="Kr" to="Hb"/>  
  </graph>  
</gxl>
```

# validating GXL

- `graph::validateGXL` uses *XML* `libxml2`-based methods to validate against the GXL DTD
- not frequently exploited, examples are in the *graph* package
- XML schema support in R *XML* package (or `libxml2`) not well-established

# graphSadj – coming

- large graphs are efficiently encoded by sparse representation of adjacency matrix
- *sparseM* package on CRAN (Koenker, Ng) serves reasonably well
- lacks dimnames, efficient subscripting, tricks can be added
- not clear if we can inherit graph algorithms based on sparse encoding – Boost, LEDA?

# RBGL

- open source STL-like utility library at [www.boost.org](http://www.boost.org) (or <file:///C:/cygwin/home/stvjc/bioC/Docs/Papers/Graph.2pp/boostTOC.htm>)
- we currently can convert `graph::graphNEL` objects to boost structures and execute `tsort`, `bfs`, `dfs`, `Dijkstra SP`

```
> print(dijkstra.sp(dd))
```

```
$distances
```

```
[1] 0 6 1 4 5
```

```
$penult
```

```
[1] 1 5 1 3 4
```

```
$start
```

```
[1] 1
```

# current RBGL functions

```
> print(objects("package:RBGL")[-bado])
```

```
[1] "adjListBGL"      "adjMat"          "Am2A1"           "bfs"  
[5] "dijkstra.sp"    "mstree.kruskal" "tsort"
```

```
> print(adjMat(dd))
```

```
      A B C D E  
[1,] 0 0 0 0 1  
[2,] 0 1 1 0 1  
[3,] 1 0 0 0 0  
[4,] 0 1 1 0 0  
[5,] 0 1 0 1 0
```

```
> print(adjListBGL(dd))
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]  
[1,]    0    1    1    1    2    2    3    4    4  
[2,]    2    1    3    4    1    3    4    0    1
```

# RBGL status

- until recently, quite a mess
- now quite a bit cleaner: all interfaces use `.Call`, mild name mangling keeps R/Boost graph types in sync
- incantation for creating a Boost graph (adjlist) instance `g`

```
SEXP BGL_KMST_D( SEXP num_verts_in, SEXP num_edges_in,
                SEXP R_edges_in, SEXP R_weights_in)
{
using namespace boost;

setupGraphTypes
setTraits( Graph_dd ) # dd = directed, double-value
setWeightedDoubleEdges( Graph_dd )
```

# Rggobi application

- aim: link a metadata graph to observational data to support EDA and concept-directed browsing of observations
- tools: ggobi, Rggobi, XML
- now switch to active ggobi

## RDF and inference on network structure

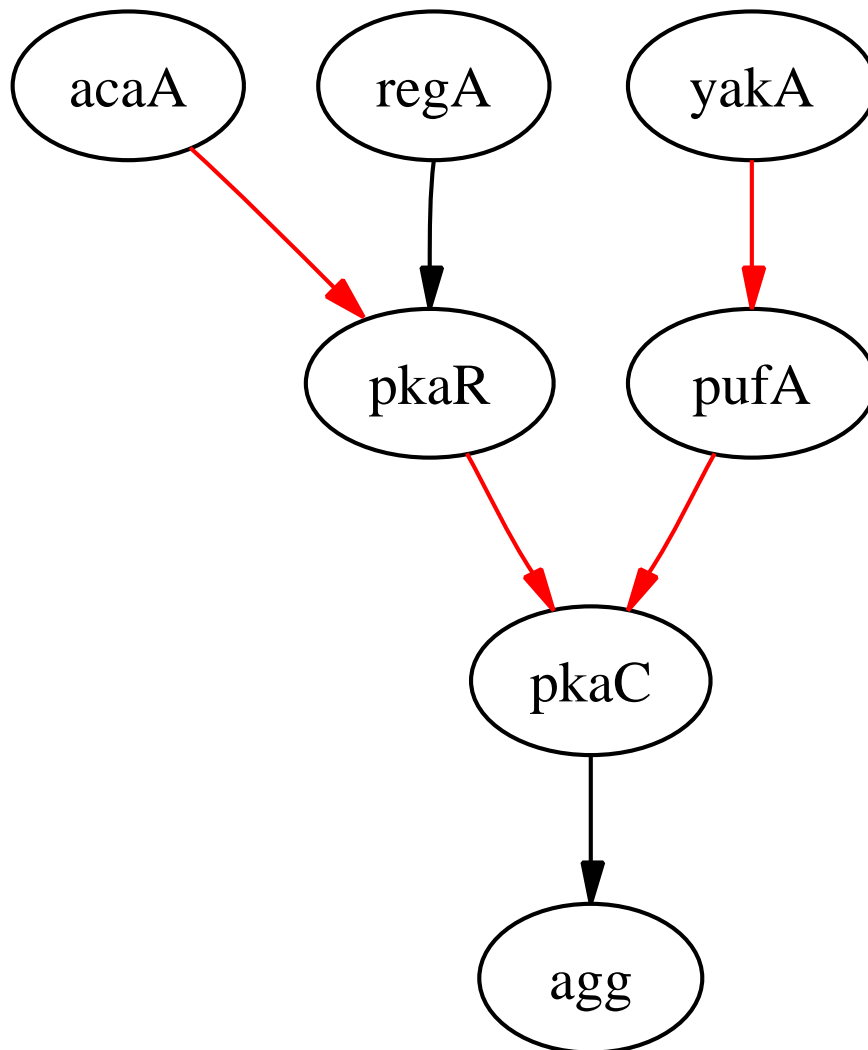
- Given information on gene-phenotype associations obtained through a series of knockout experiments

Table 1. Experimental data on *Dicoryosentium* aggregation

Exp No.	Genotype	Aggregation [-, ±, +, ++]
1	wild-type	+
2	<i>yslA</i> <sup>-</sup>	-
3	<i>pufA</i> <sup>-</sup>	++
4	<i>pldR</i> <sup>-</sup>	++
5	<i>pldC</i> <sup>-</sup>	-
6	<i>acaA</i> <sup>-</sup>	-
7	<i>regA</i> <sup>-</sup>	++
8	<i>acaA</i> <sup>+</sup>	++
9	<i>pldC</i> <sup>+</sup>	++
10	<i>pldC</i> <sup>-</sup> , <i>regA</i> <sup>-</sup>	-
11	<i>yslA</i> <sup>-</sup> , <i>pufA</i> <sup>-</sup>	++
12	<i>yslA</i> <sup>-</sup> , <i>pldR</i> <sup>-</sup>	+
13	<i>yslA</i> <sup>-</sup> , <i>pldC</i> <sup>-</sup>	-
14	<i>pldC</i> <sup>-</sup> , <i>yslA</i> <sup>+</sup>	-
15	<i>yslA</i> <sup>-</sup> , <i>pldC</i> <sup>+</sup>	++

## RDF and inference on network structure

- Derive, using biological rules, the gene interaction network



## an approach to network structure inference

- database on knockout experiments encoded in RDF (a minimal framework for description, with all statements of form subject-verb-object or object-property-value)
- rules written for cwm, an inference engine, rdf processor written in python by T. Berners-Lee
- cwm derives the interactions, exports as RDF/XML
- R parses XML and converts to R graphs or dot
- rules in use extracted from B. Zupan et al, *Bioinformatics* 2003; their GenePath WWW service (Univ. Ljubljana) is based on prolog

# Conclusions

- R/graph/graphviz/XML/BGL/ggobi interoperate quite well
- *sparseM* very promising tool for adjacency matrix operations, needs infrastructure
- RSGB in development, SGB has interesting example graphs and algorithms, software a little hard to port
- *sna* package includes large collection of graph-oriented tools related to social network analysis