Pathview: pathway based data integration and visualization

Weijun Luo (luo_weijun AT yahoo.com)

April 16, 2015

Abstract

In this vignette, we demonstrate the *pathview* package as a tool set for pathway based data integration and visualization. It maps and renders user data on relevant pathway graphs. All users need is to supply their gene or compound data and specify the target pathway. *Pathview* automatically downloads the pathway graph data, parses the data file, maps user data to the pathway, and renders pathway graph with the mapped data. Although built as a stand-alone program, *pathview* may seamlessly integrate with pathway and gene set (enrichment) analysis tools for a large-scale and fully automated analysis pipeline. In this vignette, we introduce common and advanced uses of *pathview*. We also cover package installation, data preparation, other useful features and common application errors. In *gage* package, vignette "RNA-Seq Data Pathway and Gene-set Analysis Workflows" demonstrates GAGE/Pathview workflows on RNA-seq (and microarray) pathway analysis.

1 Cite our work

Weijun Luo and Cory Brouwer. Pathview: an R/Bioconductor package for pathway-based data integration and visualization. Bioinformatics, 29(14):1830-1831, 2013. doi: 10.1093/bioinformatics/btt285.

2 Quick start with demo data

This is the most basic use of *pathview*, please check the full description below for details. Here we assume that the input data are already read in R as in the demo examples. If you are not sure how to do so, you may check Section Common uses for data visualization or *gage* secondary vignette, "Gene set and data preparation".

```
> library(pathview)
> data(gse16873.d)
> pv.out <- pathview(gene.data = gse16873.d[, 1], pathway.id = "04110",
+                    species = "hsa", out.suffix = "gse16873")
```

3 New features

Pathview ($\geq$ 1.5.4) provides paths.hsa data, the full list of human pathway ID/names from KEGG, as to help user specify target pathway IDs when calling `pathview`. Please check Section Common uses for details.

Pathview ($\geq$ 1.5.4) adjust the definitions of 7 arguments for `pathview` function: `discrete, limit, bins, both.dirs, trans.fun, low, mid, high`. These arguments now accept 1- or 2-element vectors beside of 2-element lists. For example, limit=1 is equivalent to limit=list(gene=1, cpd=1), and bins=c(3, 10) is equivalent to bins=list(gene=3, cpd=10) etc. This would makes pathview easier to use.

Pathview ($\geq$ 1.1.6) can plot/integrate/compare multiple states or samples in the same graph (Subsection 8.2).

Pathview ($\geq$ 1.2.4) work with all KEGG species (about 3000) plus KEGG Orthology (with `species="ko"`) (Subsection 8.5).

4 Overview

Pathview (Luo and Brouwer, 2013) is a stand-alone software package for pathway based data integration and visualization. This package can be divided into four functional modules: the Downloader, Parser, Mapper and Viewer. Mostly importantly, *pathview* maps and renders user data on relevant pathway graphs.

Pathview generates both native KEGG view (like Figure 1 in PNG format) and Graphviz view (like Figure 2 in PDF format) for pathways (Section 7). KEGG view keeps all the meta-data on pathways, spacial and temporal information, tissue/cell types, inputs, outputs and connections. This is important for human reading and interpretation of pathway biology. Graphviz view provides better control of node and edge attributes, better view of pathway topology, better understanding of the pathway analysis statistics. Currently only KEGG pathways are implemented. Hopefully, pathways from Reactome, NCI and other databases will be supported in the future. Notice that KEGG requires subscription for FTP access since May 2011. However, *Pathview* downloads individual pathway graphs and data files through API or HTTP access, which is freely available (for academic and non-commerical uses). *Pathview* uses *KEGGgraph* (Zhang and Wiemann, 2009) when parsing KEGG xml data files.

Pathview provides strong support for data integration (Section 8). It works with: 1) essentially all types of biological data mappable to pathways, 2) over 10 types of gene or protein IDs, and 20 types of compound or metabolite IDs, 3) pathways for about 3000 species as well as KEGG orthology, 4) varoius data attributes and formats, i.e. continuous/discrete data, matrices/vectors, single/multiple samples etc.

Pathview is open source, fully automated and error-resistant. Therefore, it seamlessly integrates with pathway or gene set (enrichment) analysis tools. In Section 9, we will show an integrated analysis using *pathview* with anothr the Bioconductor *gage* package (Luo et al., 2009), available from the Bioconductor website.

Note that although we use microarray data as example gene data in this vignette, Pathview is equally applicable to RNA-Seq data and other types of gene/protein centered high throughput data. The secondary vignette in *gage* package, "RNA-Seq Data Pathway and Gene-set Analysis Workflows", demonstrates such applications.

This vignette is written by assuming the user has minimal R/Bioconductor knowledge. Some descriptions and code chunks cover very basic usage of R. The more experienced users may simply omit these parts.

5 Installation

Assume R and Bioconductor have been correctly installed and accessible under current directory. Otherwise, please contact your system admin or follow the instructions on R website and Bioconductor website. Here I would strongly recommend users to install or upgrade to the latest verison of R (3.0.2+)/Bioconductor (2.14+) for simpler installation and better use of *Pathview*. You may need to update your `biocLite` too if you upgrade R/Biocondutor under Windows.

Start R: from Linux/Unix command line, type `R` (Enter); for Mac or Windows GUI, double click the R application icon to enter R console.

End R: type in `q()` when you are finished with the analysis using R, but not now.

Two options:

Simple way: install with Bioconductor installation script `biocLite` directly (this included all dependencies automatically too):

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("pathview")
```

Or a bit more complexer: install through R-forge or manually, but require dependence packages to be installed using Bioconductor first:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite(c("Rgraphviz", "png", "KEGGgraph", "org.Hs.eg.db"))
```

Then install *pathview* through R-forge.

```
> install.packages("pathview",repos="http://R-Forge.R-project.org")
```

Or install manually: download *pathview* package (from R-forge or Bioconductor, make sure with proper version number and zip format) and save to `/your/local/directory/`.

```
> install.packages("/your/local/directory/pathview_1.0.0.tar.gz",
+     repos = NULL, type = "source")
```

Note that there might be problems when installing *Rgraphviz* or *XML* (*KEGGgraph* dependency) package with outdated R/Bioconductor. *Rgraphviz* installation is a bit complicate with R 2.5 (Bioconductor 2.10) or earlier versions. Please check this Readme file on *Rgraphviz*. On Windows systems,*XML* frequently needs to be installed manually. Its windows binary can be downloaded from CRAN and then:

```
> install.packages("/your/local/directory/XML_3.95-0.2.zip", repos = NULL)
```

6 Get Started

Under R, first we load the *pathview* package:

```
> library(pathview)
```

To see a brief overview of the package:

```
> library(help=pathview)
```

To get help on any function (say the main function, `pathview`), use the `help` command in either one of the following two forms:

```
> help(pathview)
> ?pathview
```

7 Common uses for data visualization

Pathview is primarily used for visualizing data on pathway graphs. *pathview* generates both native KEGG view (like Figure 1) and Graphviz view (like Figure 2). The former render user data on native KEGG pathway graphs, hence is natural and more readable for human. The latter layouts pathway graph using Graphviz engine, hence provides better control of node or edge attributes and pathway topology.

We load and look at the demo microarray data first. This is a breast cancer dataset. Here we would like to view the pair-wise gene expression changes between DCIS (disease) and HN (control) samples. Note that the microarray data are log2 transformed. Hence expression changes are log2 ratios.

```
> data(gse16873.d)
```

Here we assume that the input data are already read in R. If not, you may use R functions `read.delim`, `read.table` etc to read in your data. For example, you may read in a truncated version of gse16873 and process it as below.

```
> filename=system.file("extdata/gse16873.demo", package = "pathview")
> gse16873=read.delim(filename, row.names=1)
> gse16873.d=gse16873[,2*(1:6)]-gse16873[,2*(1:6)-1]
```

We also load the demo pathway related data, which includes 3 pathway ids and related plotting parameters.

```
> data(demo.paths)
```

We may also check the full list of KEGG pathway ID/names if needed. We provide human pathway IDs (in the form of hsa+5 digits) mapping to pathway names. It is almost the same for other species, excpet for the 3 or 4 letter species code. Please check Subsection 8.5 for KEGG species code.

```
> data(paths.hsa)
> head(paths.hsa,3)
```

```
                     hsa00010                       hsa00020
"Glycolysis / Gluconeogenesis"    "Citrate cycle (TCA cycle)"
                     hsa00030
   "Pentose phosphate pathway"
```

First, we view the exprssion changes of a single sample (pair) on a typical signaling pathway, "Cell Cycle", by specifying the `gene.data` and `pathway.id` (Figure 1a). The microarray was done on human tissue, hence `species = "hsa"`. Note that such native KEGG view was outupt as a raster image in a PNG file in your working directory.

```
> i <- 1
> pv.out <- pathview(gene.data = gse16873.d[, 1], pathway.id = demo.paths$sel.paths[i],
+                    species = "hsa", out.suffix = "gse16873", kegg.native = T)
> list.files(pattern="hsa04110", full.names=T)

[1] "./hsa04110.gse16873.png" "./hsa04110.png"
[3] "./hsa04110.xml"

> str(pv.out)

List of 2
 $ plot.data.gene:'data.frame':        92 obs. of  9 variables:
  ..$ kegg.names: chr [1:92] "1029" "51343" "4171" "4998" ...
  ..$ labels    : chr [1:92] "CDKN2A" "FZR1" "MCM2" "ORC1" ...
  ..$ type      : chr [1:92] "gene" "gene" "gene" "gene" ...
  ..$ x         : num [1:92] 532 919 553 494 919 919 188 432 123 77 ...
  ..$ y         : num [1:92] 124 536 556 556 297 519 519 191 704 687 ...
  ..$ width     : num [1:92] 46 46 46 46 46 46 46 46 46 46 ...
  ..$ height    : num [1:92] 17 17 17 17 17 17 17 17 17 17 ...
  ..$ mol.data  : num [1:92] 0.129 -0.404 -0.42 0.986 1.181 ...
  ..$ mol.col   : Factor w/ 10 levels "#00FF00","#30EF30",..: 5 3 3 9 9 9 9 9 5 6 ...
 $ plot.data.cpd : NULL

> head(pv.out$plot.data.gene)

  kegg.names labels type   x   y width height   mol.data mol.col
1       1029 CDKN2A gene 532 124    46     17  0.1291987 #BEBEBE
2      51343   FZR1 gene 919 536    46     17 -0.4043256 #5FDF5F
```

```
3       4171   MCM2 gene 553 556    46     17 -0.4202181 #5FDF5F
4       4998   ORC1 gene 494 556    46     17  0.9864873 #FF0000
5        996  CDC27 gene 919 297    46     17  1.1811525 #FF0000
6        996  CDC27 gene 919 519    46     17  1.1811525 #FF0000
```

Graph from the first example above has a single layer. Node colors were modified on the original graph layer, and original KEGG node labels (node names) were kept intact. This way the output file size is as small as the original KEGG PNG file, but the computing time is relative long. If we want a fast view and do not mind doubling the output file size, we may do a two-layer graph with `same.layer = F` (Figure 1b). This way node colors and labels are added on an extra layer above the original KEGG graph. Notice that the original KEGG gene labels (or EC numbers) were replaced by official gene symbols.

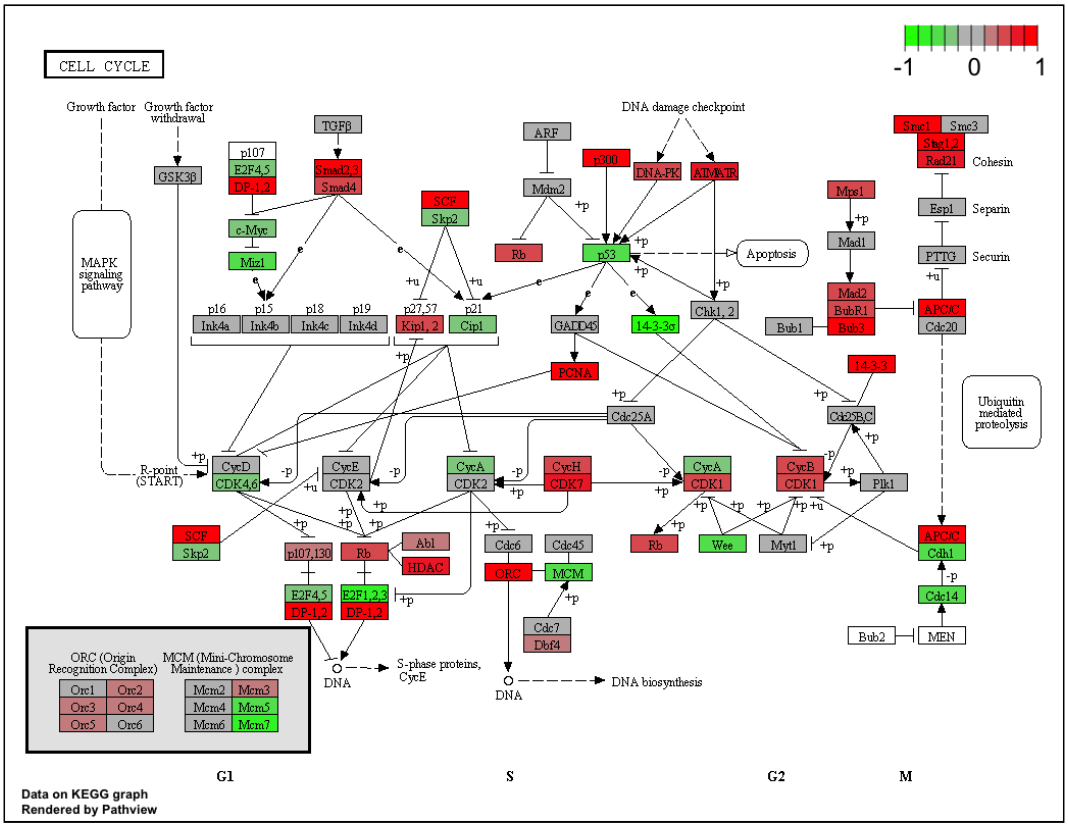
```
> pv.out <- pathview(gene.data = gse16873.d[, 1], pathway.id = demo.paths$sel.paths[i],
+                    species = "hsa", out.suffix = "gse16873.2layer", kegg.native = T,
+                    same.layer = F)
```

In the above two examples, we view the data on native KEGG pathway graph. This view we get all notes and meta-data on the KEGG graphs, hence the data is more readable and interpretable. However, the output graph is a raster image in PNG format. We may also view the data with a *de novo* pathway graph layout using Graphviz engine (Figure 2). The graph has the same set of nodes and edges, but with a different layout. We get more controls over the nodes and edge attributes and look. Importantly, the graph is a vector image in PDF format in your working directory.

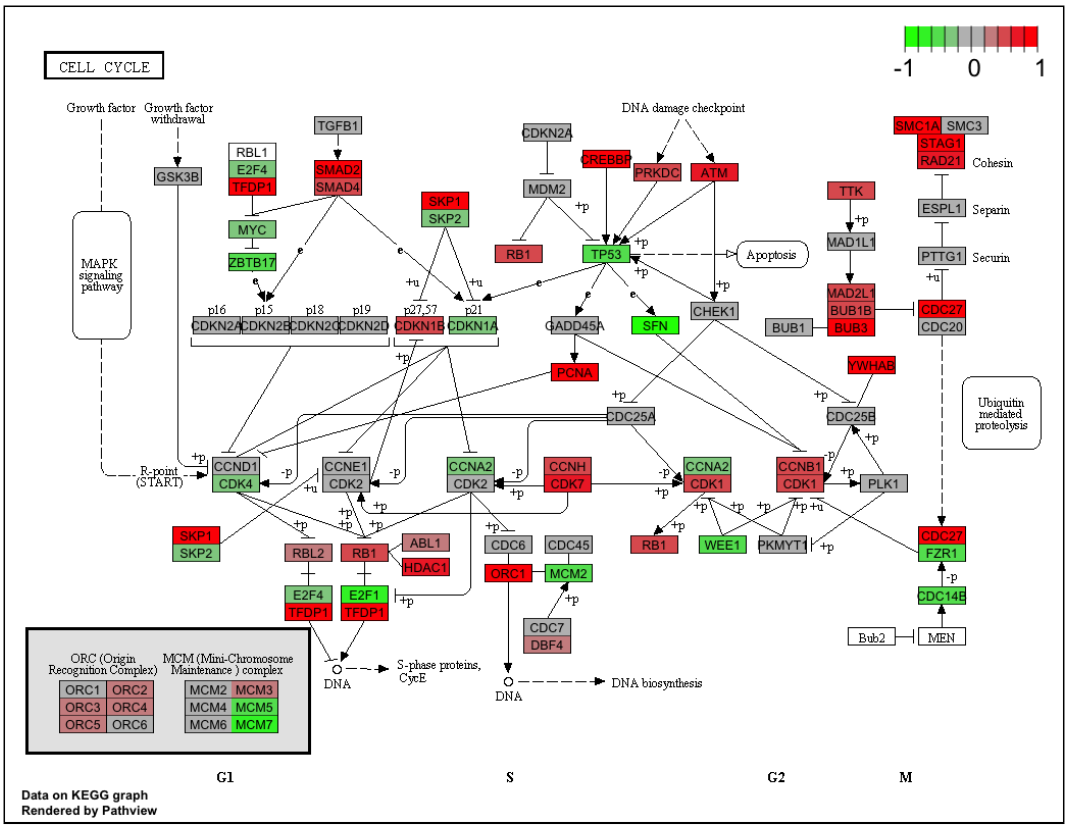
```
> pv.out <- pathview(gene.data = gse16873.d[, 1], pathway.id = demo.paths$sel.paths[i],
+                    species = "hsa", out.suffix = "gse16873", kegg.native = F,
+                    sign.pos = demo.paths$spos[i])
> #pv.out remains the same
> dim(pv.out$plot.data.gene)

[1] 92  9

> head(pv.out$plot.data.gene)

  kegg.names labels type   x   y width height   mol.data mol.col
1       1029 CDKN2A gene 532 124    46     17  0.1291987 #BEBEBE
2      51343   FZR1 gene 919 536    46     17 -0.4043256 #5FDF5F
3       4171   MCM2 gene 553 556    46     17 -0.4202181 #5FDF5F
4       4998   ORC1 gene 494 556    46     17  0.9864873 #FF0000
5        996  CDC27 gene 919 297    46     17  1.1811525 #FF0000
6        996  CDC27 gene 919 519    46     17  1.1811525 #FF0000
```

In the example above, both main graph and legend were put in one layer (or page). We just list KEGG edge types and ignore node types in legend as to save space. If we want the complete legend, we can do a Graphviz view with two layers (Figure 3): page 1 is the main graph, page 2 is the legend. Note that for Graphviz view (PDF file), the concept of "layer" is slightly different from native KEGG view (PNG file). In both cases, we set argument `same.layer=F` for two-layer graph.

```
> pv.out <- pathview(gene.data = gse16873.d[, 1], pathway.id = demo.paths$sel.paths[i],
+     species = "hsa", out.suffix = "gse16873.2layer", kegg.native = F,
+     sign.pos = demo.paths$spos[i], same.layer = F)
```



(a)



(b)

Figure 1: Example native KEGG view on gene data with the (a) default settings; or (b) `same.layer=F`.

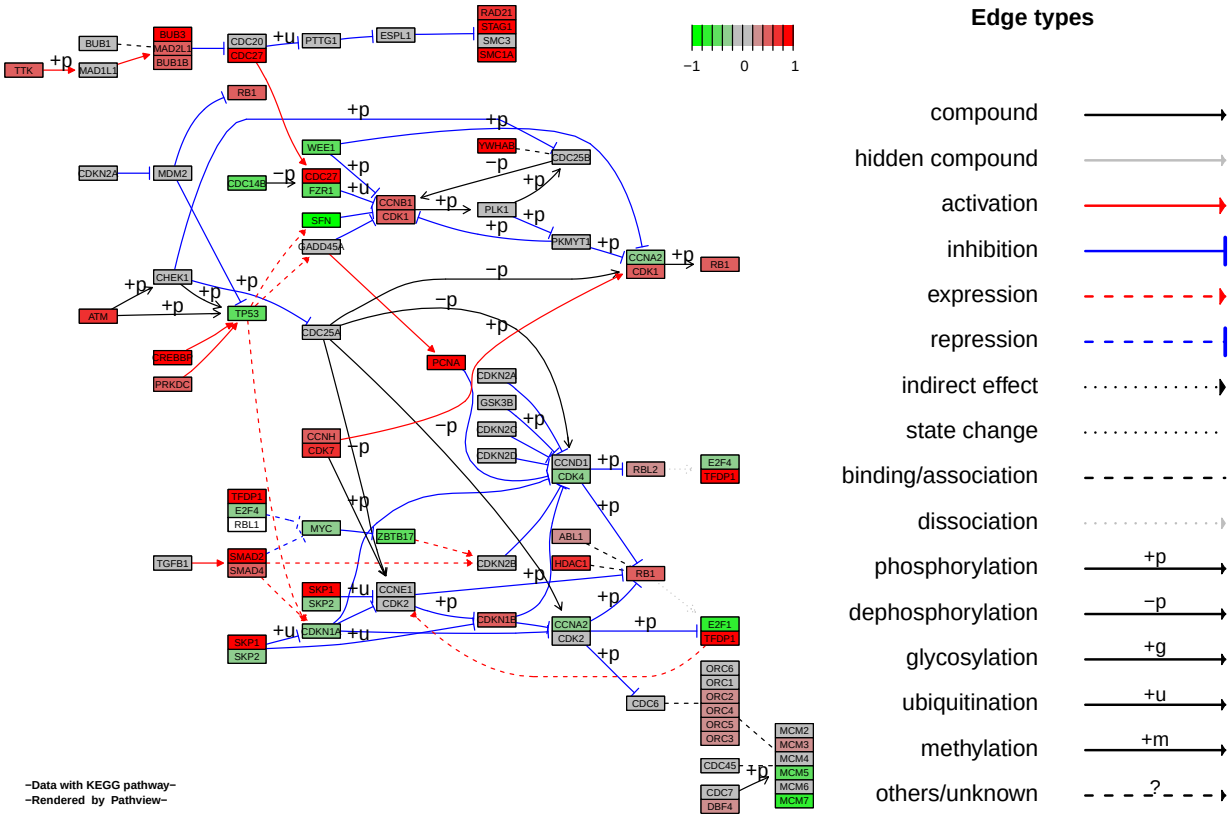
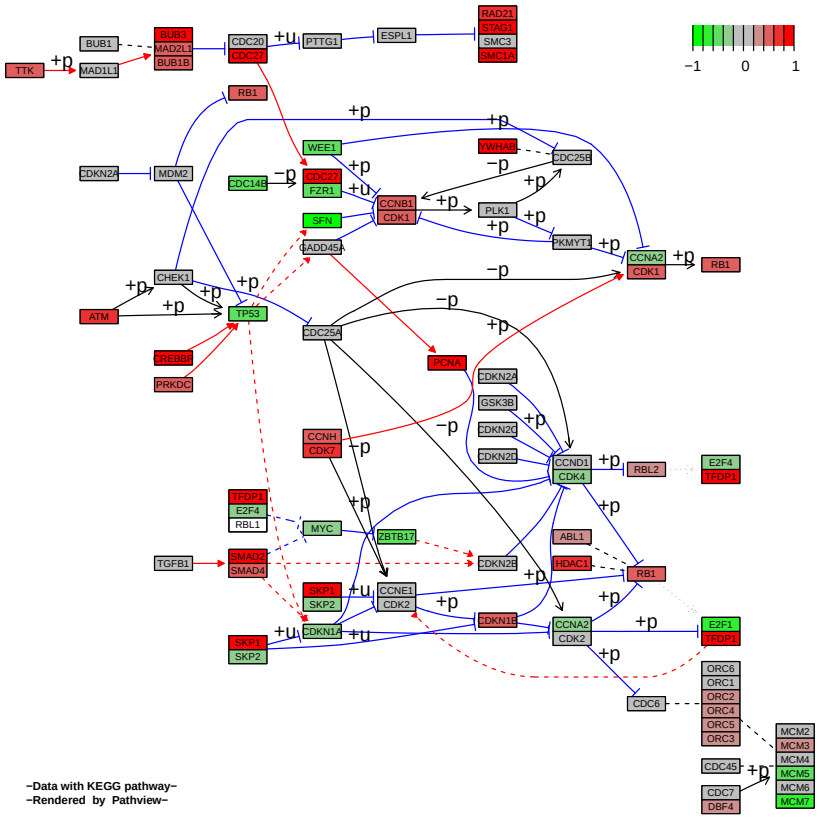
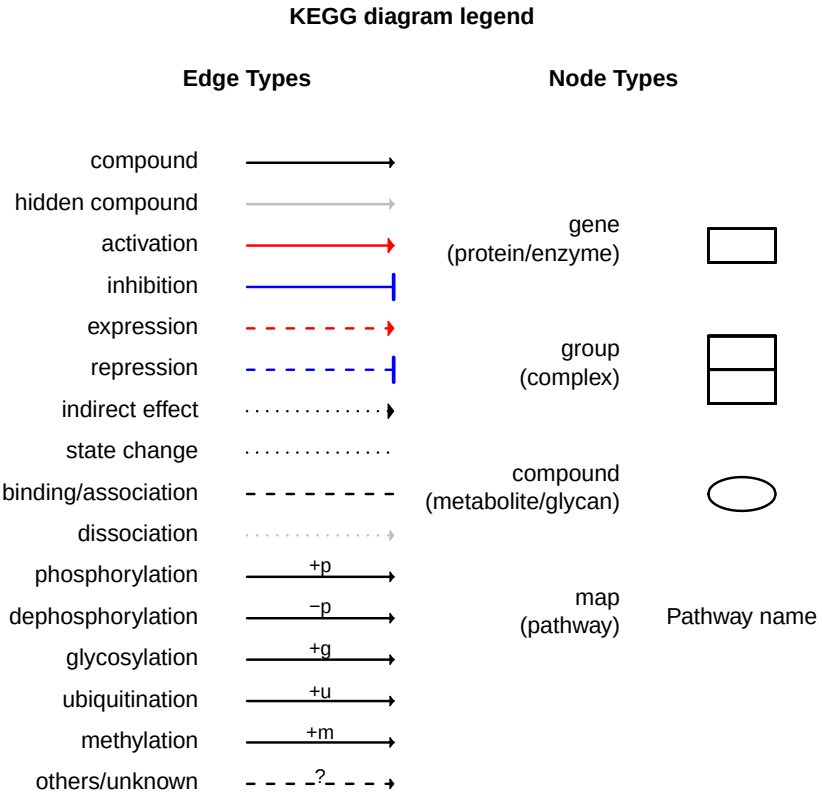


Figure 2: Example Graphviz view on gene data with default settings. Note that legend is put on the same page as main graph.

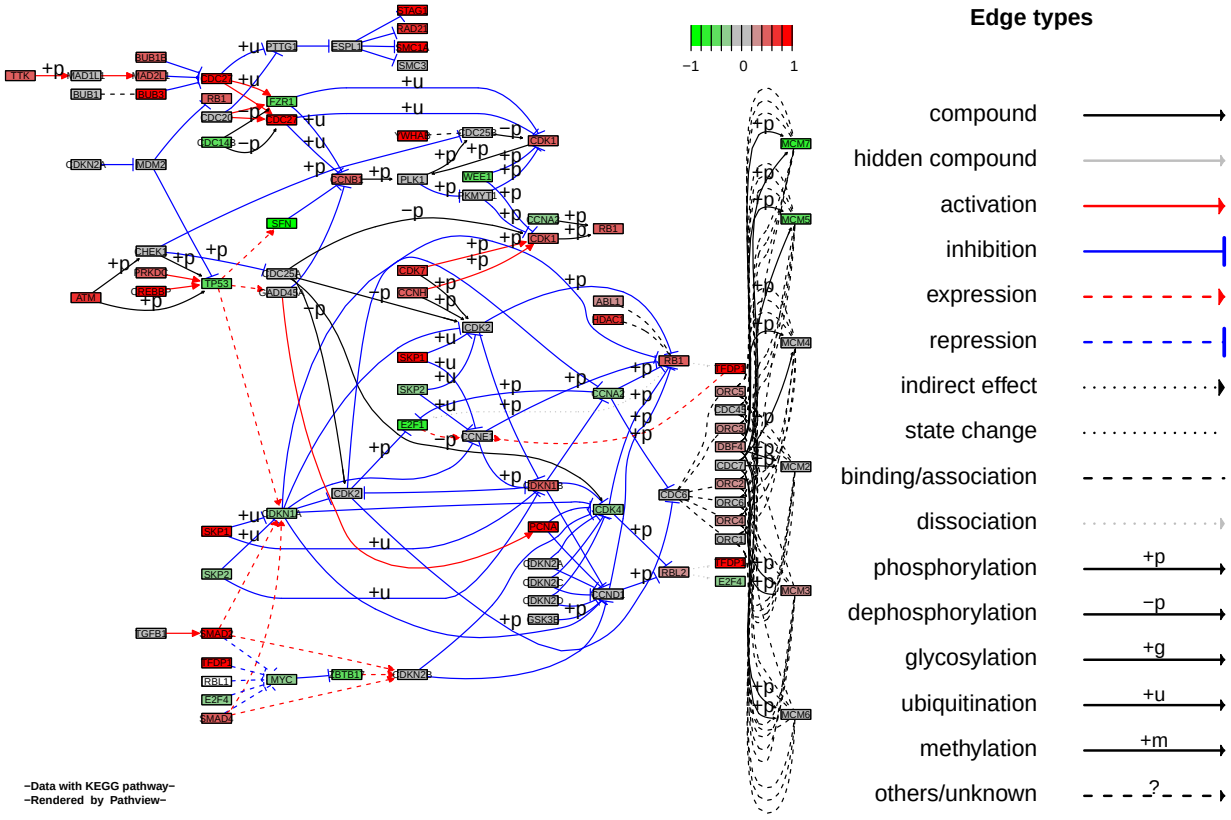


(a) page 1

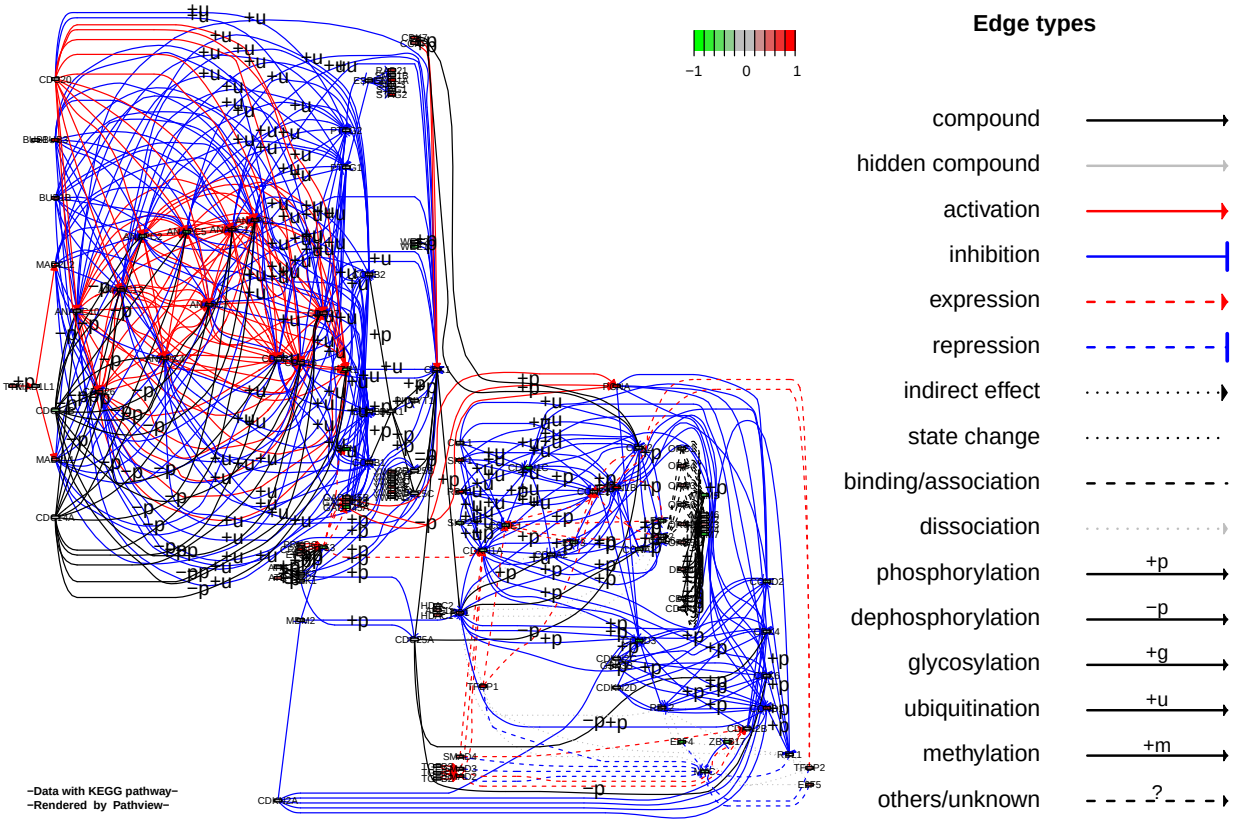


(b) page 2

Figure 3: Example Graphviz view on gene data with `same.layer=F`. Note that legend is put on a different page than main graph.



(a)



(b)

Figure 4: Example Graphviz view on gene data with (a) `split.group = T`; or (b) `expand.node = T`.

In Graphviz view, we have more control over the graph layout. We may split the node groups into individual detached nodes (Figure 4a). We may even expand the multiple-gene nodes into individual genes (Figure 4b). The split nodes or expanded genes may inherit the edges from the unsplit group or unexpanded nodes. This way we tend to get a gene/protein-gene/protein interaction network. And we may better view the network characteristics (modularity etc) and gene-wise (instead of node-wise) data. Note in native KEGG view, a gene node may represent multiple genes/proteins with similar or redundant functional role. The number of member genes range from 1 up to several tens. They are intentionally put together as a single node on pathway graphs for better clarity and readability. Therefore, we do not split node and mark each member genes separately by default. But rather we visualize the node-wise data by summarize gene-wise data, users may specify the summarization method using `node.sum` arguement.

```
> pv.out <- pathview(gene.data = gse16873.d[, 1], pathway.id = demo.paths$sel.paths[i],
+     species = "hsa", out.suffix = "gse16873.split", kegg.native = F,
+     sign.pos = demo.paths$spos[i], split.group = T)
> dim(pv.out$plot.data.gene)

[1] 92  9

> head(pv.out$plot.data.gene)

  kegg.names labels type   x   y width height   mol.data mol.col
1       1029 CDKN2A gene 532 124    46     17  0.1291987 #BEBEBE
2      51343   FZR1 gene 919 536    46     17 -0.4043256 #5FDF5F
3       4171   MCM2 gene 553 556    46     17 -0.4202181 #5FDF5F
4       4998   ORC1 gene 494 556    46     17  0.9864873 #FF0000
5        996  CDC27 gene 919 297    46     17  1.1811525 #FF0000
6        996  CDC27 gene 919 519    46     17  1.1811525 #FF0000

> pv.out <- pathview(gene.data = gse16873.d[, 1], pathway.id = demo.paths$sel.paths[i],
+     species = "hsa", out.suffix = "gse16873.split.expanded", kegg.native = F,
+     sign.pos = demo.paths$spos[i], split.group = T, expand.node = T)
> dim(pv.out$plot.data.gene)

[1] 124   9

> head(pv.out$plot.data.gene)

          kegg.names labels type   x   y width height    mol.data mol.col
hsa:1029        1029 CDKN2A gene 532 124    46     17  0.12919874 #BEBEBE
hsa:51343      51343   FZR1 gene 919 536    46     17 -0.40432563 #5FDF5F
hsa:4171        4171   MCM2 gene 553 556    46     17  0.17968149 #BEBEBE
hsa:4172        4172   MCM3 gene 553 556    46     17  0.33149955 #CE8F8F
hsa:4173        4173   MCM4 gene 553 556    46     17  0.06996779 #BEBEBE
hsa:4174        4174   MCM5 gene 553 556    46     17 -0.42874682 #5FDF5F
```

8 Data integration

Pathview provides strong support for data integration. It can be used to integrate, analyze and visualize a wide variety of biological data (Subsection 8.1): gene expression, protein expression, genetic association, metabolite, genomic data, literature, and other data types mappable to pathways. Notebaly, it can be directly used for metagenomic, microbiome or unknown species data when the data are mapped to KEGG ortholog pathways

(Subsection 8.5). The integrated Mapper module maps a variety of gene/protein IDs and compound/metabolite IDs to standard KEGG gene or compound IDs (Subsection 8.4). User data named with any of these different ID types get accurately mapped to target KEGG pathways. Currently, *pathview* covers KEGG pathways for about 3000 species (Subsection 8.5), and species can be specified either as KEGG code, scientific name or comon name. In addition, *pathview* works with different data attributes and formats, both continuous and discrete data (Subsection 8.3), either in matrix or vector format, with single or multiple samples/experiments etc. Partcullary, *Pathview* can now integrate and compare multiple samples or states into one graph (Subsection 8.2).

8.1 Compound and gene data

In examples above, we viewed gene data with canonical signaling pathways. We frequently want to look at metabolic pathways too. Besides gene nodes, these pathways also have compound nodes. Therefore, we may integrate or visualize both gene data and compound data with metabolic pathways. Here gene data is a broad concept including genes, transcripts, protein , enzymes and their expression, modifications and any measurable attributes. Same is compound data, including metabolites, drugs, their measurements and attributes. Here we still use the breast cancer microarray dataset as gene data. We then generate simulated compound or metabolomic data, and load proper compound ID types (with sufficient number of unique entries) for demonstration.

```
> sim.cpd.data=sim.mol.data(mol.type="cpd", nmol=3000)
> data(cpd.simtypes)
```

We generate a native KEGG view graph with both gene data and compound data (Figure 5a). Such metabolic pathway graphs generated by *pathview* is the same as the original KEGG graphs, except that the compound nodes are magnified for better view of the colors.

```
> i <- 3
> print(demo.paths$sel.paths[i])

[1] "00640"

> pv.out <- pathview(gene.data = gse16873.d[, 1], cpd.data = sim.cpd.data,
+     pathway.id = demo.paths$sel.paths[i], species = "hsa", out.suffix = "gse16873.cpd",
+     keys.align = "y", kegg.native = T, key.pos = demo.paths$kpos1[i])
> str(pv.out)

List of 2
 $ plot.data.gene:'data.frame':        21 obs. of  9 variables:
  ..$ kegg.names: chr [1:21] "4329" "31" "23417" "18" ...
  ..$ labels    : chr [1:21] "ALDH6A1" "ACACA" "MLYCD" "ABAT" ...
  ..$ type      : chr [1:21] "gene" "gene" "gene" "gene" ...
  ..$ x         : num [1:21] 202 202 202 319 418 921 857 778 685 801 ...
  ..$ y         : num [1:21] 325 237 196 378 327 390 538 494 390 390 ...
  ..$ width     : num [1:21] 46 46 46 46 46 46 46 46 46 46 ...
  ..$ height    : num [1:21] 17 17 17 17 17 17 17 17 17 17 ...
  ..$ mol.data  : num [1:21] 0.747 -0.483 -0.251 2.785 0.77 ...
  ..$ mol.col   : Factor w/ 8 levels "#30EF30","#5FDF5F",..: 6 2 3 7 6 6 6 8 7 4 ...
 $ plot.data.cpd :'data.frame':        44 obs. of  9 variables:
  ..$ kegg.names: chr [1:44] "C00222" "C00804" "C01013" "C00099" ...
  ..$ labels    : chr [1:44] "C00222" "C00804" "C01013" "C00099" ...
  ..$ type      : chr [1:44] "compound" "compound" "compound" "compound" ...
```

```
  ..$ x         : num [1:44] 268 265 367 368 265 555 650 775 775 555 ...
  ..$ y         : num [1:44] 327 449 327 388 228 116 91 88 157 63 ...
  ..$ width     : num [1:44] 8 8 8 8 8 8 8 8 8 8 ...
  ..$ height    : num [1:44] 8 8 8 8 8 8 8 8 8 8 ...
  ..$ mol.data  : num [1:44] 0.14 0.143 NA -0.638 1.053 ...
  ..$ mol.col   : Factor w/ 8 levels "#0000FF","#3030EF",..: 5 5 8 2 7 5 7 4 1 8 ...
```

```
> head(pv.out$plot.data.cpd)
```

```
   kegg.names labels     type   x   y width height      mol.data mol.col
25     C00222 C00222 compound 268 327     8      8  0.1397585262 #BEBEBE
26     C00804 C00804 compound 265 449     8      8  0.1429100287 #BEBEBE
27     C01013 C01013 compound 367 327     8      8            NA #FFFFFF
28     C00099 C00099 compound 368 388     8      8 -0.6382880325 #3030EF
30     C00083 C00083 compound 265 228     8      8  1.0532858787 #FFFF00
84     C02614 C02614 compound 555 116     8      8  0.0003758095 #BEBEBE
```

We also generate Graphviz view of the same pathway and data (Figure 5b). Graphviz view better shows the hierarchical structure. For metabolic pathways, we need to parse the reaction entries from xml files and convert it to relationships between gene and compound nodes. We use ellipses for compound nodes. The labels are standard compound names, which are retrieved from CHEMBL database. KEGG does not provide it in the pathway database files. Chemical names are long strings, we need to do word wrap to fit them to specified width on the graph.

```
> pv.out <- pathview(gene.data = gse16873.d[, 1], cpd.data = sim.cpd.data,
+     pathway.id = demo.paths$sel.paths[i], species = "hsa", out.suffix = "gse16873.cpd",
+     keys.align = "y", kegg.native = F, key.pos = demo.paths$kpos2[i],
+     sign.pos = demo.paths$spos[i], cpd.lab.offset = demo.paths$offs[i])
```

8.2 Multiple states or samples

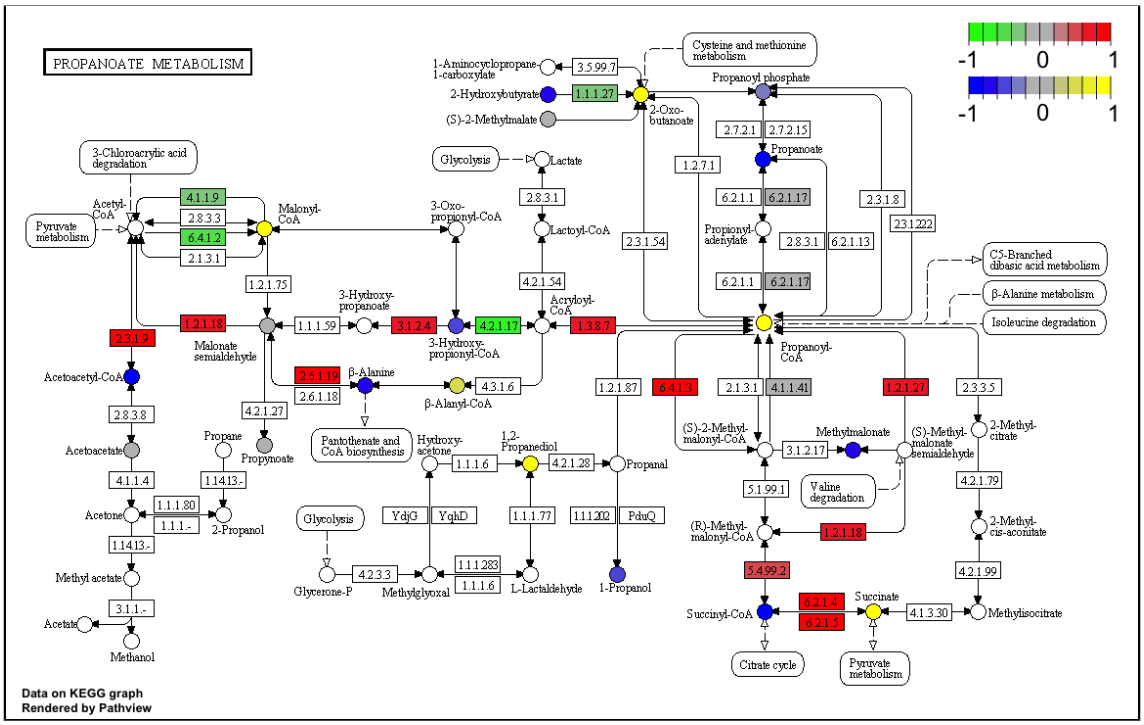
In all previous examples, we looked at single sample data, which are either vector or single-column matrix. *Pathview* also handles multiple sample data, it used to generate graph for each sample. Since version 1.1.6, *Pathview* can integrate and plot multiple samples or states into one graph (Figure 6 - 7).

Let's simulate compound data with multiple replicate samples first.

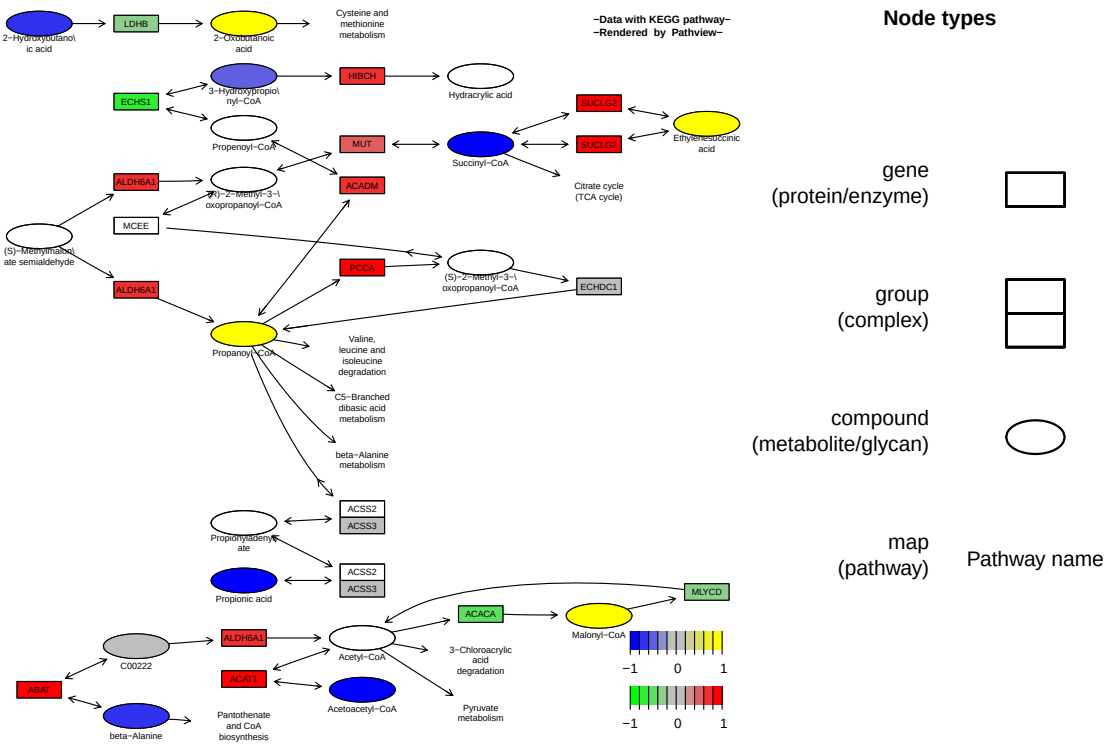
```
> set.seed(10)
> sim.cpd.data2 = matrix(sample(sim.cpd.data, 18000,
+     replace = T), ncol = 6)
> rownames(sim.cpd.data2) = names(sim.cpd.data)
> colnames(sim.cpd.data2) = paste("exp", 1:6, sep = "")
> head(sim.cpd.data2, 3)
```

```
             exp1       exp2      exp3       exp4        exp5      exp6
C02787  0.62355826 -0.1108793  1.069398 -0.9595403 1.653444849  1.360614
C08521 -1.23737070  0.4676360 -2.064253 -0.6593838 0.004274093  0.512765
C01043 -0.01768295  0.5472769 -0.592388 -0.1190882 0.950917578 -1.130288
```

In the following examples, `gene.data` has three samples while `cpd.data` has two. We may include all these samples in one graph. We can do either native KEGG view (Figure 6) or Graphviz view (Figure 7)on such multiple-sample data. In these graphs, we see that the gene nodes and compound nodes are sliced into multiple



(a)



(b)

Figure 5: Example (a) KEGG view or (b) Graphviz view on both gene data and compound data simultaneously.

pieces corresponding to different states or samples. Since the sample sizes are different for `gene.data` and `cpd.data`, we can choose to match the data if samples in the two data types are actually paired, i.e. first columns of for `gene.data` and `cpd.data` come from the same experiment/sample, and so on.

```
> #KEGG view
> pv.out <- pathview(gene.data = gse16873.d[, 1:3],
+     cpd.data = sim.cpd.data2[, 1:2], pathway.id = demo.paths$sel.paths[i],
+     species = "hsa", out.suffix = "gse16873.cpd.3-2s", keys.align = "y",
+     kegg.native = T, match.data = F, multi.state = T, same.layer = T)
> head(pv.out$plot.data.cpd)
```

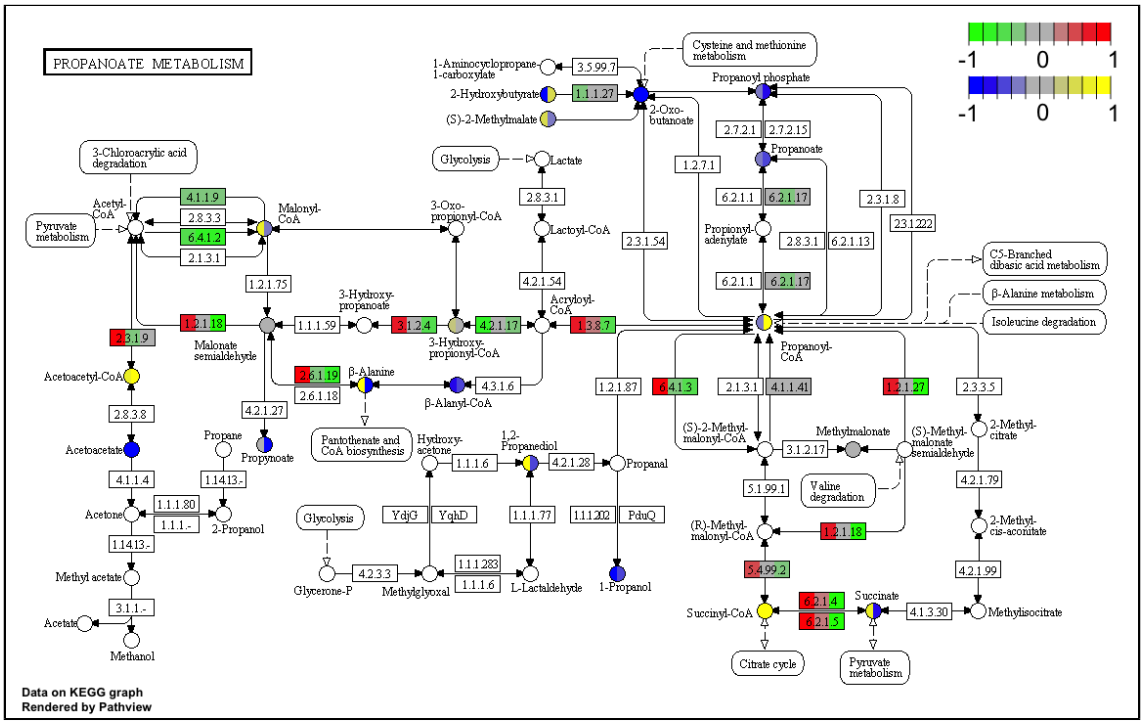
```
   kegg.names labels     type   x   y width height       exp1       exp2
25     C00222 C00222 compound 268 327     8      8 -0.1322799  0.1614143
26     C00804 C00804 compound 265 449     8      8 -0.1996916 -1.0967526
27     C01013 C01013 compound 367 327     8      8         NA         NA
28     C00099 C00099 compound 368 388     8      8  1.4532987 -0.9155783
30     C00083 C00083 compound 265 228     8      8  0.6580861 -0.3964395
84     C02614 C02614 compound 555 116     8      8  0.5832773 -0.2103209
   exp1.col exp2.col
25  #BEBEBE  #BEBEBE
26  #BEBEBE  #0000FF
27  #FFFFFF  #FFFFFF
28  #FFFF00  #0000FF
30  #EFEF30  #8F8FCE
84  #DFDF5F  #8F8FCE
```

```
> #KEGG view with data match
> pv.out <- pathview(gene.data = gse16873.d[, 1:3],
+     cpd.data = sim.cpd.data2[, 1:2], pathway.id = demo.paths$sel.paths[i],
+     species = "hsa", out.suffix = "gse16873.cpd.3-2s.match",
+     keys.align = "y", kegg.native = T, match.data = T, multi.state = T,
+     same.layer = T)
> #graphviz view
> pv.out <- pathview(gene.data = gse16873.d[, 1:3],
+     cpd.data = sim.cpd.data2[, 1:2], pathway.id = demo.paths$sel.paths[i],
+     species = "hsa", out.suffix = "gse16873.cpd.3-2s", keys.align = "y",
+     kegg.native = F, match.data = F, multi.state = T, same.layer = T,
+     key.pos = demo.paths$kpos2[i], sign.pos = demo.paths$spos[i])
```

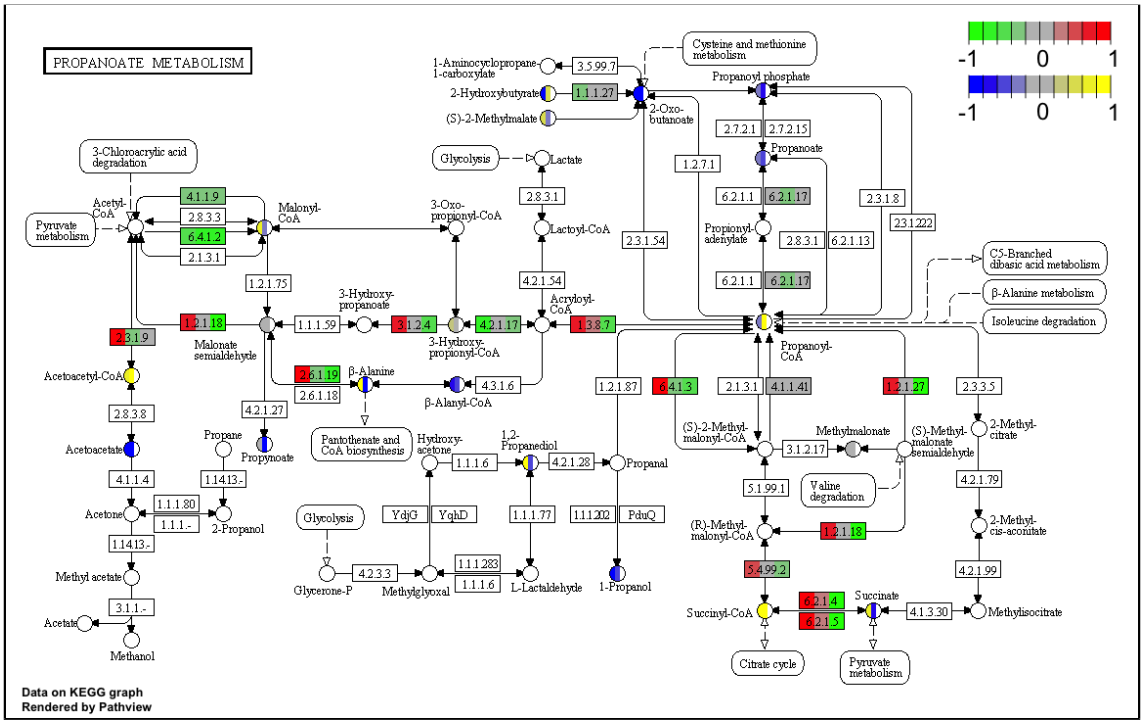
Again, we may choose to plot the samples separately, i.e. one sample per graph. Note that in this case, the samples in `gene.data` and `cpd.data` has to be matched as to be assigned to the same graph. Hence, argument `match.data` isn't really effective here.

```
> #plot samples/states separately
> pv.out <- pathview(gene.data = gse16873.d[, 1:3],
+     cpd.data = sim.cpd.data2[, 1:2], pathway.id = demo.paths$sel.paths[i],
+     species = "hsa", out.suffix = "gse16873.cpd.3-2s", keys.align = "y",
+     kegg.native = T, match.data = F, multi.state = F, same.layer = T)
```

As described above, KEGG views on the same layer may takes some time. Again, we can choose to do KEGG view with two layers as to speed up the process if we don't mind losing the original KEGG gene labels (or EC numbers).



(a)



(b)

Figure 6: Example KEGG view on multiple states of both gene data and compound data simultaneously (a) without or (b) with matching the samples.

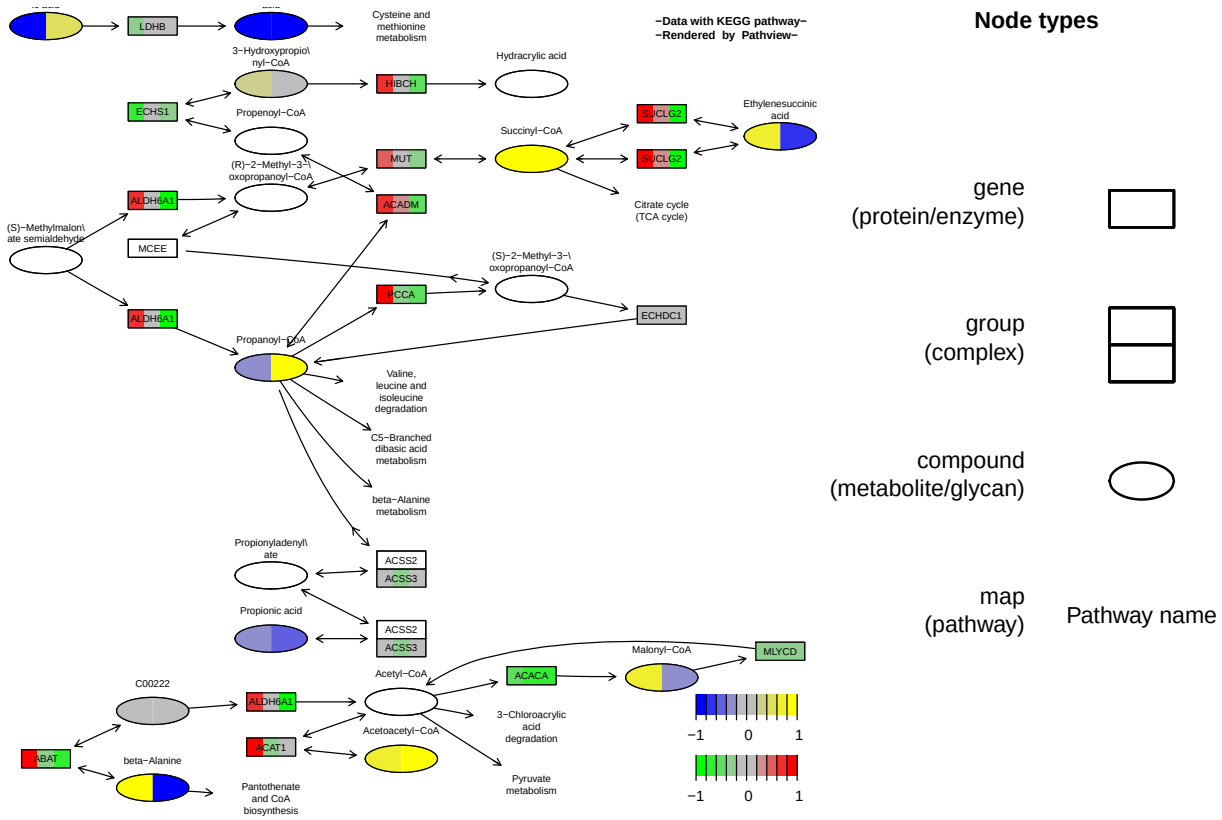


Figure 7: Example Graphviz view on multiple states of both gene data and compound data simultaneously.

```
> pv.out <- pathview(gene.data = gse16873.d[, 1:3],
+     cpd.data = sim.cpd.data2[, 1:2], pathway.id = demo.paths$sel.paths[i],
+     species = "hsa", out.suffix = "gse16873.cpd.3-2s.2layer",
+     keys.align = "y", kegg.native = T, match.data = F, multi.state = T,
+     same.layer = F)
```

8.3 Discrete data

So far, we have been dealing with continuous data. But we often work with discrete data too. For instance, we select list of signficant genes or compound based on some statistics (p-value, fold change etc). The input data can be named vector of two levels, either 1 or 0 (signficant or not), or it can be a shorter list of signficant gene/compound names. In the next two examples, we made both `gene.data` and `cpd.data` or `gene.data` only (Figure 8) discrete.

```
> require(org.Hs.eg.db)
> gse16873.t <- apply(gse16873.d, 1, function(x) t.test(x,
+     alternative = "two.sided")$p.value)
> sel.genes <- names(gse16873.t)[gse16873.t < 0.1]
> sel.cpds <- names(sim.cpd.data)[abs(sim.cpd.data) > 0.5]
> pv.out <- pathview(gene.data = sel.genes, cpd.data = sel.cpds,
+     pathway.id = demo.paths$sel.paths[i], species = "hsa", out.suffix = "sel.genes.sel.cpd",
+     keys.align = "y", kegg.native = T, key.pos = demo.paths$kpos1[i],
+     limit = list(gene = 5, cpd = 2), bins = list(gene = 5, cpd = 2),
+     na.col = "gray", discrete = list(gene = T, cpd = T))
> pv.out <- pathview(gene.data = sel.genes, cpd.data = sim.cpd.data,
+     pathway.id = demo.paths$sel.paths[i], species = "hsa", out.suffix = "sel.genes.cpd",
+     keys.align = "y", kegg.native = T, key.pos = demo.paths$kpos1[i],
+     limit = list(gene = 5, cpd = 1), bins = list(gene = 5, cpd = 10),
+     na.col = "gray", discrete = list(gene = T, cpd = F))
```

8.4 ID mapping

A distinguished feature of *pathview* is its strong ID mapping capability. The integrated Mapper module maps over 10 types of gene or protein IDs, and 20 types of compound or metabolite IDs to standard KEGG gene or compound IDs, and also maps between these external IDs. In other words, user data named with any of these different ID types get accurately mapped to target KEGG pathways. *Pathview* applies to pathways for about 3000 species, and species can be specified in multiple formats: KEGG code, scientific name or comon name.

The following example makes use of the integrated mapper to map external ID types to standard KEGG IDs automatically (Figure 9). We only need to specify the external ID types using `gene.idtype` and `cpd.idtype` arguments. Note that automatic mapping is limited to certain ID types. For details check: `gene.idtype.list` and `data(rn.list); names(rn.list)`.

```
> cpd.cas <- sim.mol.data(mol.type = "cpd", id.type = cpd.simtypes[2],
+     nmol = 10000)
> gene.ensprot <- sim.mol.data(mol.type = "gene", id.type = gene.idtype.list[4],
+     nmol = 50000)
> pv.out <- pathview(gene.data = gene.ensprot, cpd.data = cpd.cas,
+     gene.idtype = gene.idtype.list[4], cpd.idtype = cpd.simtypes[2],
+     pathway.id = demo.paths$sel.paths[i], species = "hsa", same.layer = T,
+     out.suffix = "gene.ensprot.cpd.cas", keys.align = "y", kegg.native = T,
```

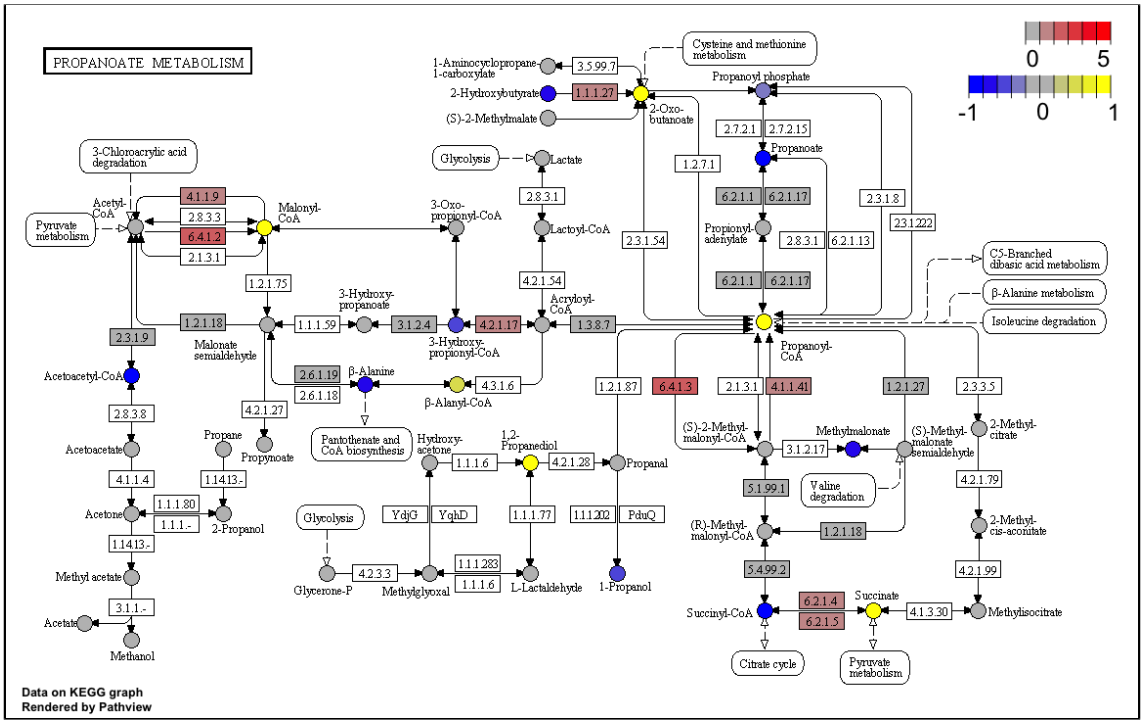


Figure 8: Example native KEGG view on discrete gene data and continuous compound data simultaneously.

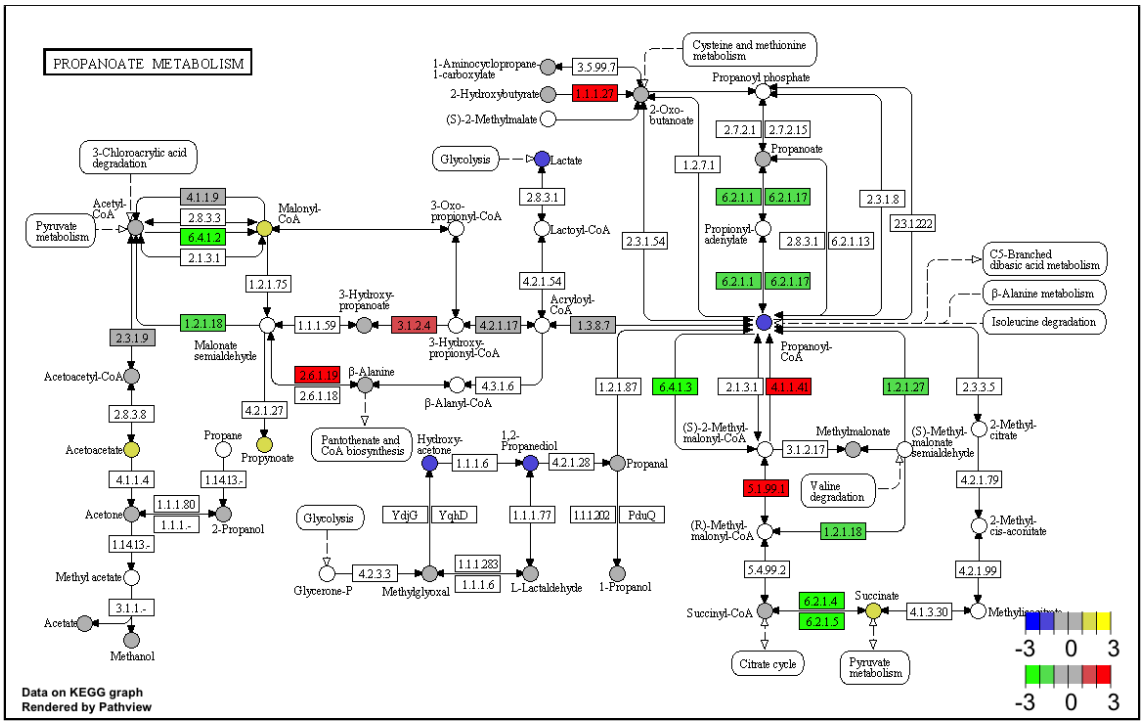


Figure 9: Example native KEGG view on gene data and compound data with other ID types.

```
+     key.pos = demo.paths$kpos2[i], sign.pos = demo.paths$spos[i],
+     limit = list(gene = 3, cpd = 3), bins = list(gene = 6, cpd = 6))
```

For external IDs not in the auto-mapping lists, we may make use of the `mol.sum` function (also part of the Mapper module) to do the ID and data mapping explicitly. Here we need to provide `id.map`, the mapping matrix between external ID and KEGG standard ID. We use ID mapping functions including `id2eg` and `cpdidmap` etc to get `id.map` matrix. Note that these ID mapping functions can be used independent of `pathview` main function. The following example use this route with the simulated `gene.ensprot` and `cpd.kc` data above, and we get the same results (Figure not shown).

```
> id.map.cas <- cpdidmap(in.ids = names(cpd.cas), in.type = cpd.simtypes[2],
+     out.type = "KEGG COMPOUND accession")
> cpd.kc <- mol.sum(mol.data = cpd.cas, id.map = id.map.cas)
> id.map.ensprot <- id2eg(ids = names(gene.ensprot),
+     category = gene.idtype.list[4], org = "Hs")
> gene.entrez <- mol.sum(mol.data = gene.ensprot, id.map = id.map.ensprot)
> pv.out <- pathview(gene.data = gene.entrez, cpd.data = cpd.kc,
+     pathway.id = demo.paths$sel.paths[i], species = "hsa", same.layer = T,
+     out.suffix = "gene.entrez.cpd.kc", keys.align = "y", kegg.native = T,
+     key.pos = demo.paths$kpos2[i], sign.pos = demo.paths$spos[i],
+     limit = list(gene = 3, cpd = 3), bins = list(gene = 6, cpd = 6))
```

8.5 Working with species

Species is a tricky yet important issue when working with KEGG. KEGG has its own dedicated nomenclature and database for species, which includes about 3000 organisms with complete genomes. In other words, gene data for any of these organisms can be mapped, visualized and analyzed using *pathview*. This comprehensive species coverage is a prominent feature of *pathview*'s data integration capacity. However, KEGG does not treat all of these organisms/genomes the same way. KEGG use NCBI GeneID (or Entrez Gene) as the default ID for the most common model animals, including human, mouse, rat etc and a few crops, e.g. soybean, wine grape and maize. On the other hand, KEGG uses Locus tag and other IDs for all others organisms, including animals, plants, fungi, protists, as well as bacteria and archaea.

Pathview carries a data matrix `korg`, which includes a complete list of supported KEGG species and default gene IDs. Let's explore `korg` data matrix as to have some idea on KEGG species and its Gene ID usage.

```
> data(korg)
> head(korg)
```

```
     kegg.code scientific.name           common.name
[1,] "hsa"     "Homo sapiens"            "human"
[2,] "ptr"     "Pan troglodytes"         "chimpanzee"
[3,] "pps"     "Pan paniscus"            "bonobo"
[4,] "ggo"     "Gorilla gorilla gorilla" "western lowland gorilla"
[5,] "pon"     "Pongo abelii"            "Sumatran orangutan"
[6,] "mcc"     "Macaca mulatta"          "rhesus monkey"
     entrez.gnodes kegg.geneid ncbi.geneid
[1,] "1"           "100"       "100"
[2,] "1"           "100533953" "100533953"
[3,] "1"           "100967419" "100967419"
[4,] "1"           "101123859" "101123859"
[5,] "1"           "100169736" "100169736"
[6,] "1"           "100301991" "100301991"
```

```
> #number of species which use Entrez Gene as the default ID
> sum(korg[,"entrez.gnodes"]=="1",na.rm=T)
```

```
[1] 81

> #number of species which use other ID types or none as the default ID
> sum(korg[,"entrez.gnodes"]=="0",na.rm=T)

[1] 2969

> #species which do not have Entrez Gene annotation at all
> na.idx=is.na(korg[,"ncbi.geneid"])
> korg[na.idx,]

     kegg.code scientific.name
[1,] "dosa"    "Oryza sativa japonica"
[2,] "pfh"     "Plasmodium falciparum HB3"
[3,] "pfd"     "Plasmodium falciparum Dd2"
[4,] "send"    "Salmonella enterica subsp. enterica serovar Typhimurium DT104"
[5,] "sens"    "Salmonella enterica subsp. enterica serovar Agona 24249"
[6,] "senb"    "Salmonella enterica subsp. enterica serovar Bovismorbificans"
[7,] "kpr"     "Klebsiella pneumoniae subsp. rhinoscleromatis SB3432"
     common.name     entrez.gnodes kegg.geneid       ncbi.geneid
[1,] "Japanese rice" "0"           "Os01t0183400-00" NA
[2,] ""              "0"           "PFHG_00076"      NA
[3,] ""              "0"           "PFDG_00003"      NA
[4,] ""              "0"           "DT104_00921"     NA
[5,] ""              "0"           "Q786_00430"      NA
[6,] ""              "0"           "BN855_10520"     NA
[7,] ""              "0"           "KPR_0002"        NA
```

From the exploration of `korg` above, we see that out of the 3000 KEGG species, only a few don't have NCBI (Entrez) Gene ID or any gene ID (annotation) at all. Almost all of them have both default KEGG gene ID (often Locus tag) and Entrez Gene ID annotation. Therefore, `pathview` accepts `gene.idtype="kegg"` or `"Entrez"` (case insensitive) for all these species. The users need to make sure the right `gene.idtype` is specified because KEGG and Entrez Gene IDs are not the same except for 35 common species. For 19 species, Bioconductor provides gene annotation packages. The users have the freedom to input `gene.data` with other `gene.idtype`'s. For a list of these Bioconductor annotated species and extra Gene ID types allowed:

```
> data(bods)
> bods

      package             species               kegg code id.type
 [1,] "org.Ag.eg.db"      "Anopheles"           "aga"     "eg"
 [2,] "org.At.tair.db"    "Arabidopsis"         "ath"     "tair"
 [3,] "org.Bt.eg.db"      "Bovine"              "bta"     "eg"
 [4,] "org.Ce.eg.db"      "Worm"                "cel"     "eg"
 [5,] "org.Cf.eg.db"      "Canine"              "cfa"     "eg"
 [6,] "org.Dm.eg.db"      "Fly"                 "dme"     "eg"
 [7,] "org.Dr.eg.db"      "Zebrafish"           "dre"     "eg"
 [8,] "org.EcK12.eg.db"   "E coli strain K12"   "eco"     "eg"
 [9,] "org.EcSakai.eg.db" "E coli strain Sakai" "ecs"     "eg"
[10,] "org.Gg.eg.db"      "Chicken"             "gga"     "eg"
[11,] "org.Hs.eg.db"      "Human"               "hsa"     "eg"
```

```
[12,] "org.Mm.eg.db"      "Mouse"              "mmu"     "eg"
[13,] "org.Mmu.eg.db"     "Rhesus"             "mcc"     "eg"
[14,] "org.Pf.plasmo.db"  "Malaria"            "pfa"     "orf"
[15,] "org.Pt.eg.db"      "Chimp"              "ptr"     "eg"
[16,] "org.Rn.eg.db"      "Rat"                "rno"     "eg"
[17,] "org.Sc.sgd.db"     "Yeast"              "sce"     "orf"
[18,] "org.Ss.eg.db"      "Pig"                "ssc"     "eg"
[19,] "org.Xl.eg.db"      "Xenopus"            "xla"     "eg"
```

```
> data(gene.idtype.list)
> gene.idtype.list
```

```
 [1] "SYMBOL"       "GENENAME"     "ENSEMBL"      "ENSEMBLPROT"  "PROSITE"
 [6] "UNIGENE"      "UNIPROT"      "ACCNUM"       "ENSEMBLTRANS" "REFSEQ"
```

All previous examples show human data, where Entrez Gene is KEGG's default gene ID. *Pathview* now (since version 1.1.5) explicitly handles species which use Locus tag or other gene IDs as the KEGG default ID. Below are an couple of examples with E coli strain K12 data. First, we work on gene data with the default KEGG ID (Locus tag) for E coli K12.

```
> eco.dat.kegg <- sim.mol.data(mol.type="gene",id.type="kegg",species="eco",nmol=3000)
> head(eco.dat.kegg)
```

```
      b1447       b1206       b2579       b0264       b2197       b2267
-1.15259948  0.46416071  0.72893247  0.41061745 -1.46114720 -0.01890809
```

```
> pv.out <- pathview(gene.data = eco.dat.kegg, gene.idtype="kegg",
+     pathway.id = "00640", species = "eco", out.suffix = "eco.kegg",
+     kegg.native = T, same.layer=T)
```

We may also work on gene data with Entrez Gene ID for E coli K12 the same way as for human.

```
> eco.dat.entrez <- sim.mol.data(mol.type="gene",id.type="entrez",species="eco",nmol=3000)
> head(eco.dat.entrez)
```

```
     946008      945770      947068      947698      946697      946750
-1.15259948  0.46416071  0.72893247  0.41061745 -1.46114720 -0.01890809
```

```
> pv.out <- pathview(gene.data = eco.dat.entrez, gene.idtype="entrez",
+     pathway.id = "00640", species = "eco", out.suffix = "eco.entrez",
+     kegg.native = T, same.layer=T)
```

Based on the `bods` data described above, E coli K12 is an Bioconductor annotated species. Hence we may further work on its gene data with other ID types, for example, official gene symbols. When calling `pathview`, such data need to be mapped to Entrez Gene ID first (if not yet), then to default KEGG ID (Locus tag). Therefore, it takes longer time than the last example.

```
> egid.eco=eg2id(names(eco.dat.entrez), category="symbol", pkg="org.EcK12.eg.db")
> eco.dat.symbol <- eco.dat.entrez
> names(eco.dat.symbol) <- egid.eco[,2]
> head(eco.dat.kegg)
> pv.out <- pathview(gene.data = eco.dat.symbol, gene.idtype="symbol",
+     pathway.id = "00640", species = "eco", out.suffix = "eco.symbol.2layer",
+     kegg.native = T, same.layer=F)
```

Importantly, *pathview* can be directly used for metagenomic or microbiome data when the data are mapped to KEGG ortholog pathways. And data from any new species that has not been annotated and included in KEGG (non-KEGG species) can also been analyzed and visualized using *pathview* by mapping to KEGG ortholog pathways the same way. In the next example, we simulate the mapped KEGG ortholog gene data first. Then the data is input as `gene.data` with `species="ko"`. Check `pathview` function for details.

```
> ko.data=sim.mol.data(mol.type="gene.ko", nmol=5000)
> pv.out <- pathview(gene.data = ko.data, pathway.id = "04112",
+                    species = "ko", out.suffix = "ko.data", kegg.native = T)
```

9 Integrated workflow with pathway analysis

Although built as a stand alone program, *Pathview* may seamlessly integrate with pathway and functional analysis tools for large-scale and fully automated analysis pipeline. The next example shows how to connect common pathway analysis to results rendering with *pathview*. The pathway analysis was done using another Bioconductor package *gage* (Luo et al., 2009), and the selected signficant pathways plus the expression data were then piped to *pathview* for auomated results visualization (Figure not shown). In *gage* package, vignette "RNA-Seq Data Pathway and Gene-set Analysis Workflows" demonstrates GAGE/Pathview workflows on RNA-seq (and microarray) pathway analysis.

```
> library(gage)
> data(gse16873)
> cn <- colnames(gse16873)
> hn <- grep('HN',cn, ignore.case =TRUE)
> dcis <- grep('DCIS',cn, ignore.case =TRUE)
> data(kegg.gs)
> #pathway analysis using gage
> gse16873.kegg.p <- gage(gse16873, gsets = kegg.gs,
+     ref = hn, samp = dcis)
> #prepare the differential expression data
> gse16873.d <- gagePrep(gse16873, ref = hn, samp = dcis)
> #equivalently, you can do simple subtraction for paired samples
> gse16873.d <- gse16873[,dcis]-gse16873[,hn]
> #select significant pathways and extract their IDs
> sel <- gse16873.kegg.p$greater[, "q.val"] < 0.1 & !is.na(gse16873.kegg.p$greater[,
+     "q.val"])
> path.ids <- rownames(gse16873.kegg.p$greater)[sel]
> path.ids2 <- substr(path.ids[c(1, 2, 7)], 1, 8)
> #pathview visualization
> pv.out.list <- sapply(path.ids2, function(pid) pathview(gene.data = gse16873.d[,
+     1:2], pathway.id = pid, species = "hsa"))
```

10 Common Errors

- mismatch between the IDs for `gene.data` (or `cpd.data`) and `gene.idtype` (or `cpd.idtype`). For example, `gene.data` or `cpd.data` uses some extern ID types, while `gene.idtype = "entrez"` and `cpd.idtype = "kegg"` (default).
- mismatch between `gene.data` (or `cpd.data`) and `species`. For example, `gene.data` come from "mouse", while `species="hsa"`.

- `pathway.id` wrong or wrong format, right format should be a five digit number, like 04110, 00620 etc.
- any of `limit, bins, both.dir, trans.fun, discrete, low, mid, high` arguments is specified as a vector of length 1 or 2, instead of a list of 2 elements. Correct format should be like `limit = list(gene = 1, cpd = 1)`.
- `key.pos` or `sign.pos` not good, hence the color key or signature overlaps with pathway main graph.
- **Special Note**: some KEGG xml data files are incomplete, inconsistent with corresponding png image or inaccurate/incorrect on some parts. These issues may cause inaccuracy, incosistency, or error messages although *pathview* tries the best to accommodate them. For instance, we may see inconistence between KEGG view and Graphviz view. As in the latter case, the pathway layout is generated based on data from xml file.

References

Weijun Luo and Cory Brouwer. Pathview: an R/Bioconductor package for pathway-based data integration and visualization. *Bioinformatics*, 29(14):1830–1831, 2013. doi: 10.1093/bioinformatics/btt285. URL `http://bioinformatics.oxfordjournals.org/content/29/14/1830.full`.

Weijun Luo, Michael Friedman, Kerby Shedden, Kurt Hankenson, and Peter Woolf. GAGE: generally applicable gene set enrichment for pathway analysis. *BMC Bioinformatics*, 10(1):161, 2009. URL `http://www.biomedcentral.com/1471-2105/10/161`.

Jitao David Zhang and Stefan Wiemann. KEGGgraph: a graph approach to KEGG PATHWAY in R and Bioconductor. *Bioinformatics*, 25(11):1470–1471, 2009.