Pathview: pathway based data integration and visualization

Weijun Luo

June 17, 2013

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 functional) 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.

1 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 4). 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 html 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 5). 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 over 2000 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 analysis tools. In Section 6, we will show an integrated analysis using *pathview* with anothr the Bioconductor *gage* package (Luo et al., 2009), available from the Bioconductor website.

The 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.

2 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)/Bioconductor (2.12) 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/Biocondutor. *Rgraphviz* installation is a bit complicate with R 2.5 (Biocondutor 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)
```

3 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
```

4 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.

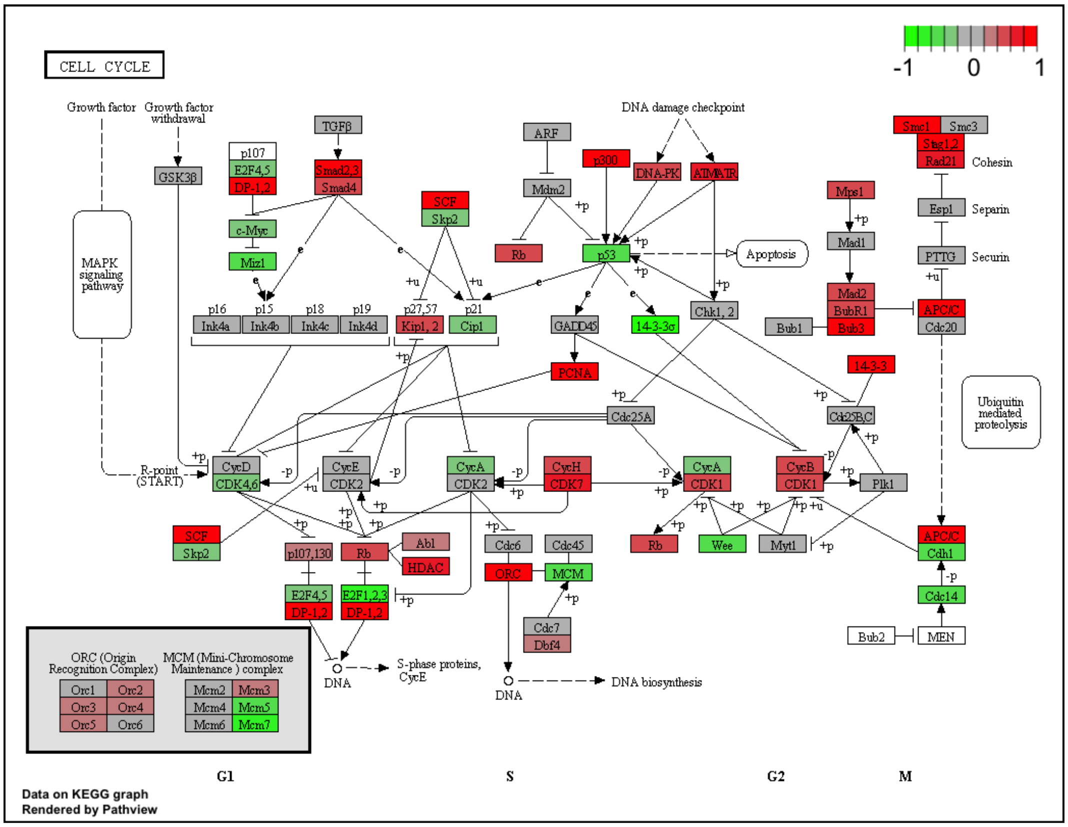
```
> data(gse16873.d)
```

We also load the demo pathway related data.

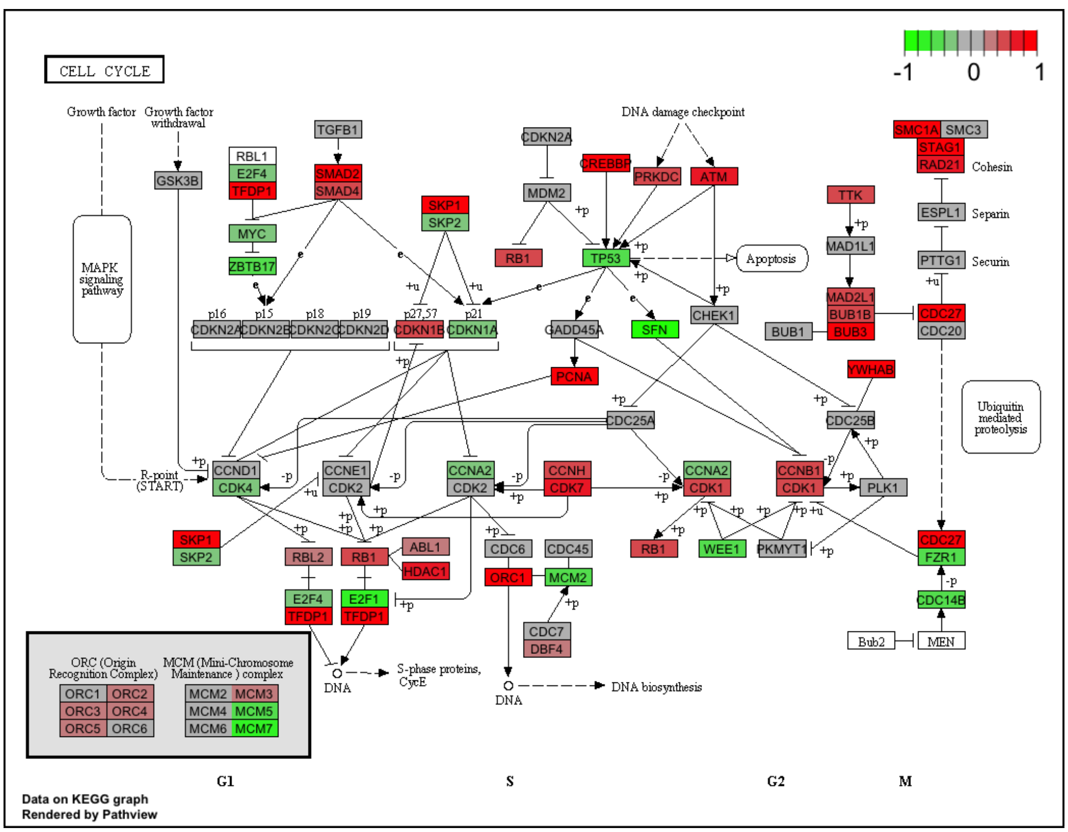
```
> data(demo.paths)
```

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)

[1] "Downloading xml files for hsa04110, 1/1 pathways.."
[1] "Downloading png files for hsa04110, 1/1 pathways.."

> 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
```



(a)



(b)

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

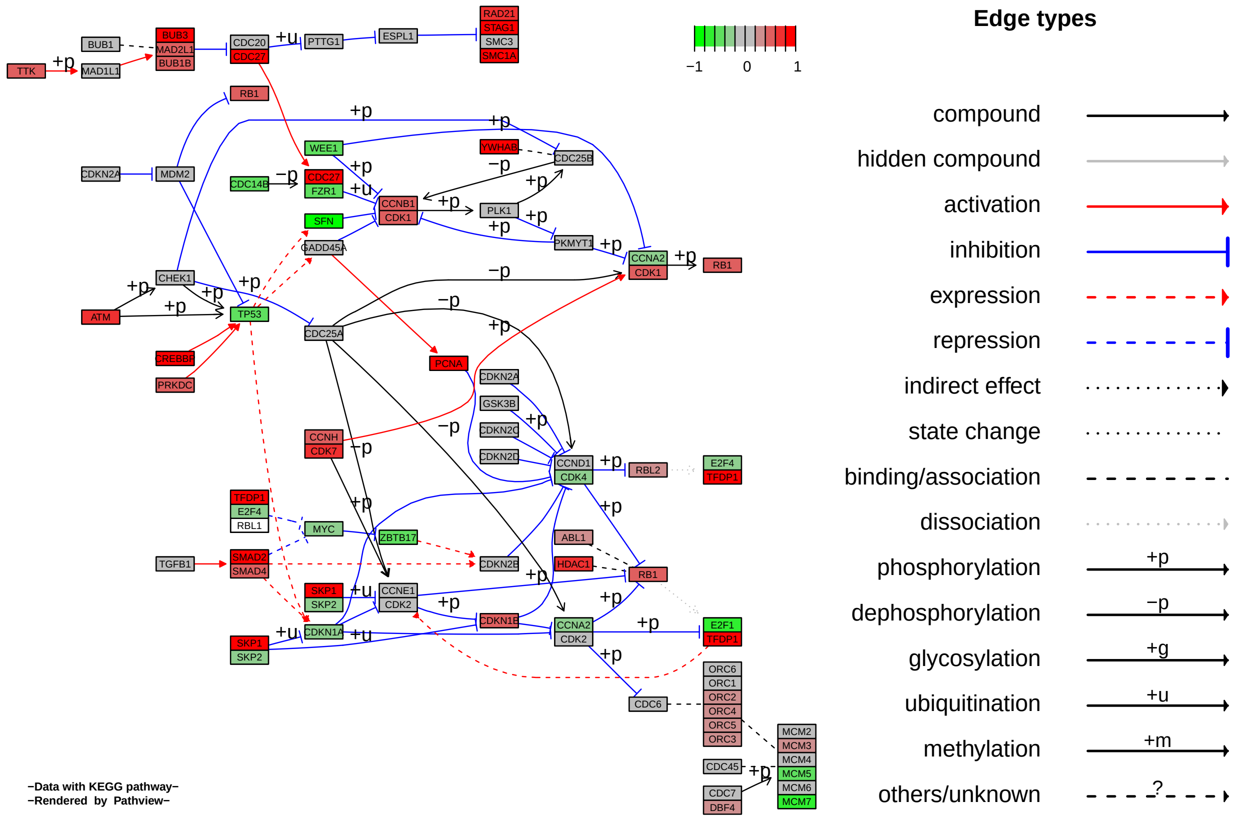


Figure 2: Example Graphviz view on gene data with default settings. Note that legend is put on the same page as main graph.

```
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.

```
> 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,
```

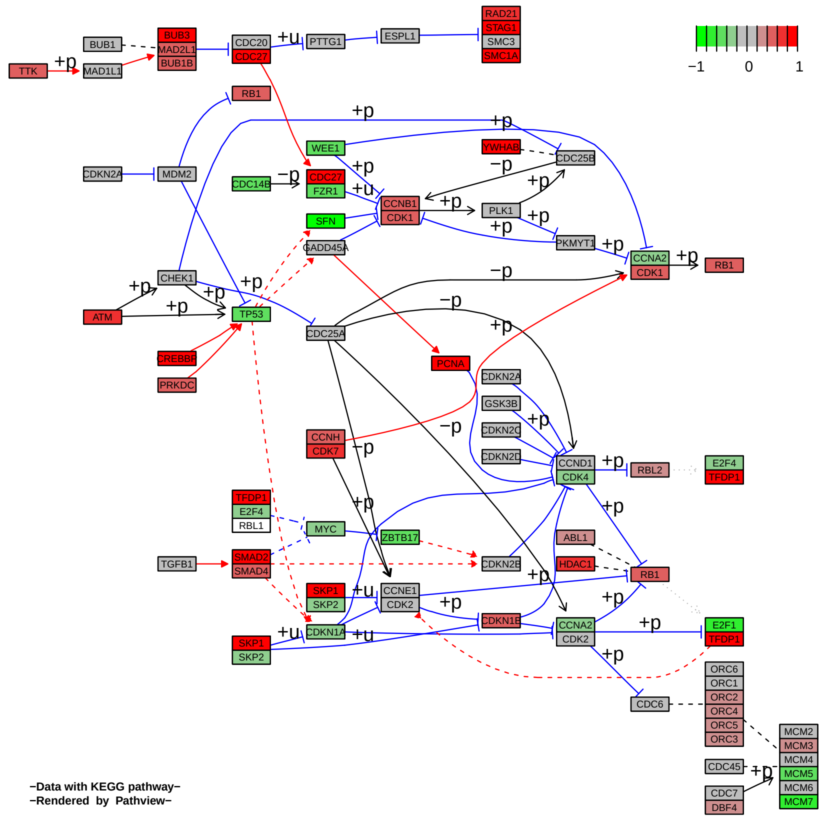
```
+                     sign.pos = demo.paths$spos[i])
> #pv.out remains the same
> dim(pv.out$plot.data.gene)

[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.

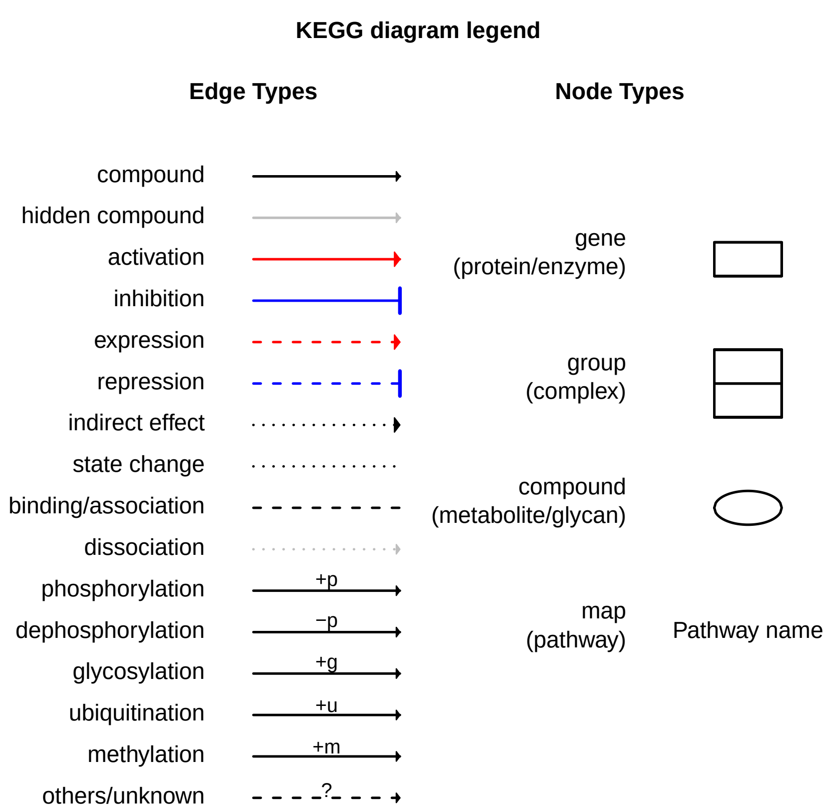
```
> pv.out <- pathview(gene.data = gse16873.d[, 1], pathway.id = demo.paths$sel.paths[i],
+     species = "hsa", out.suffix = "gse16873.2layer", kegg.native = F,
+     sign.pos = demo.paths$spos[i], same.layer = F)
```

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
```

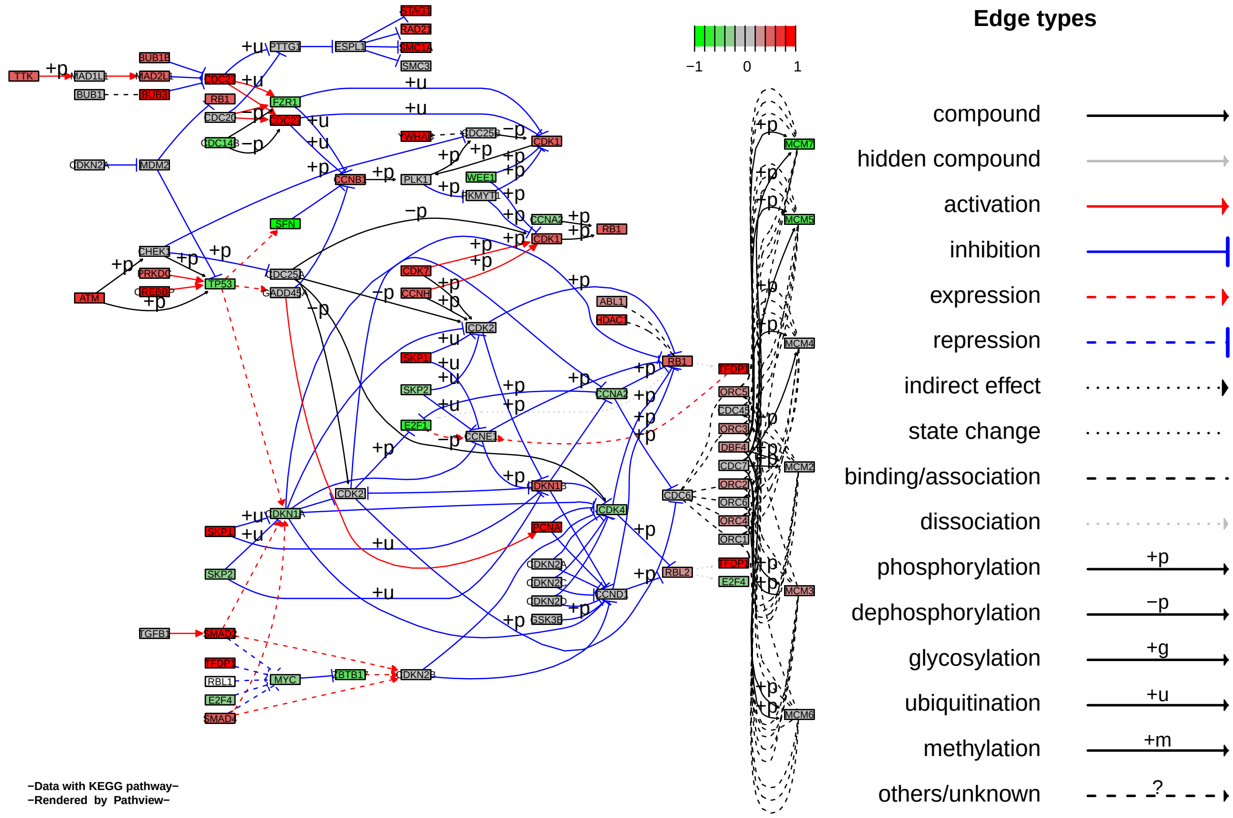


(a) page 1

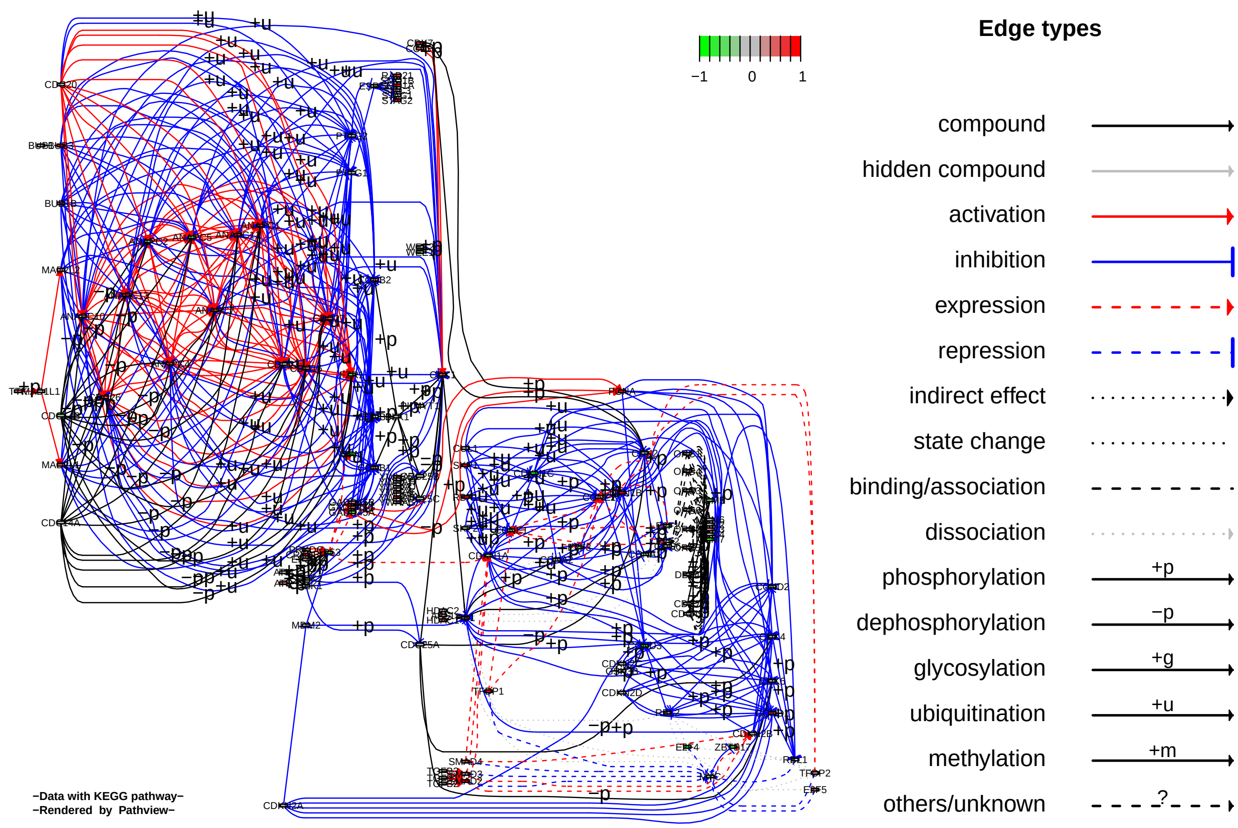


(b) page 2

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`.

```
> 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
```

5 Data integration

Pathview provides strong support for data Integration. It can be used to integrate, analyze and visualize a wide variety of biological data: 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 data when the data are mapped to KEGG ortholog pathways. The integrated Mapper module maps a variety of gene/protein IDs and compound/metabolite IDs to standard KEGG gene or compound IDs. User data named with any of these different ID types get accurately mapped to target KEGG pathways. Currently, *pathview* covers KEGG pathways for over 2000 species, 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, either in matrix or vector format, with single or multiple samples/experiments etc.

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])
```

```
[1] "Downloading xml files for hsa00640, 1/1 pathways.."
[1] "Downloading png files for hsa00640, 1/1 pathways.."
```

```
> str(pv.out)
```

```
List of 2
 $ plot.data.gene:'data.frame':       22 obs. of  9 variables:
  ..$ kegg.names: chr [1:22] "4329" "4329" "84693" "5095" ...
  ..$ labels    : chr [1:22] "ALDH6A1" "ALDH6A1" "MCEE" "PCCA" ...
  ..$ type      : chr [1:22] "gene" "gene" "gene" "gene" ...
  ..$ x         : num [1:22] 936 890 773 852 796 796 746 746 713 599 ...
  ..$ y         : num [1:22] 430 609 525 430 309 223 309 223 556 566 ...
  ..$ width     : num [1:22] 46 46 46 46 46 46 46 46 46 46 ...
  ..$ height    : num [1:22] 17 17 17 17 17 17 17 17 17 17 ...
  ..$ mol.data  : num [1:22] 0.7469 0.7469 NA 1.1903 0.0733 ...
  ..$ mol.col   : Factor w/ 8 levels "#30EF30","#5FDF5F",..: 6 6 8 7 4 4 8 8 5 7 ...
 $ plot.data.cpd :'data.frame':       36 obs. of  9 variables:
  ..$ kegg.names: chr [1:36] "C04225" "C02614" "C00109" "C02876" ...
  ..$ labels    : chr [1:36] "C04225" "C02614" "C00109" "C02876" ...
  ..$ type      : chr [1:36] "compound" "compound" "compound" "compound" ...
  ..$ x         : num [1:36] 646 551 646 771 771 551 545 545 545 771 ...
  ..$ y         : num [1:36] 495 143 118 115 184 90 184 255 354 351 ...
  ..$ width     : num [1:36] 8 8 8 8 8 8 8 8 8 8 ...
  ..$ height    : num [1:36] 8 8 8 8 8 8 8 8 8 8 ...
  ..$ mol.data  : num [1:36] NA 0.000376 0.825575 -0.35501 -1.551161 ...
  ..$ mol.col   : Factor w/ 8 levels "#0000FF","#3030EF",..: 8 5 7 4 1 8 8 8 8 7 ...
```

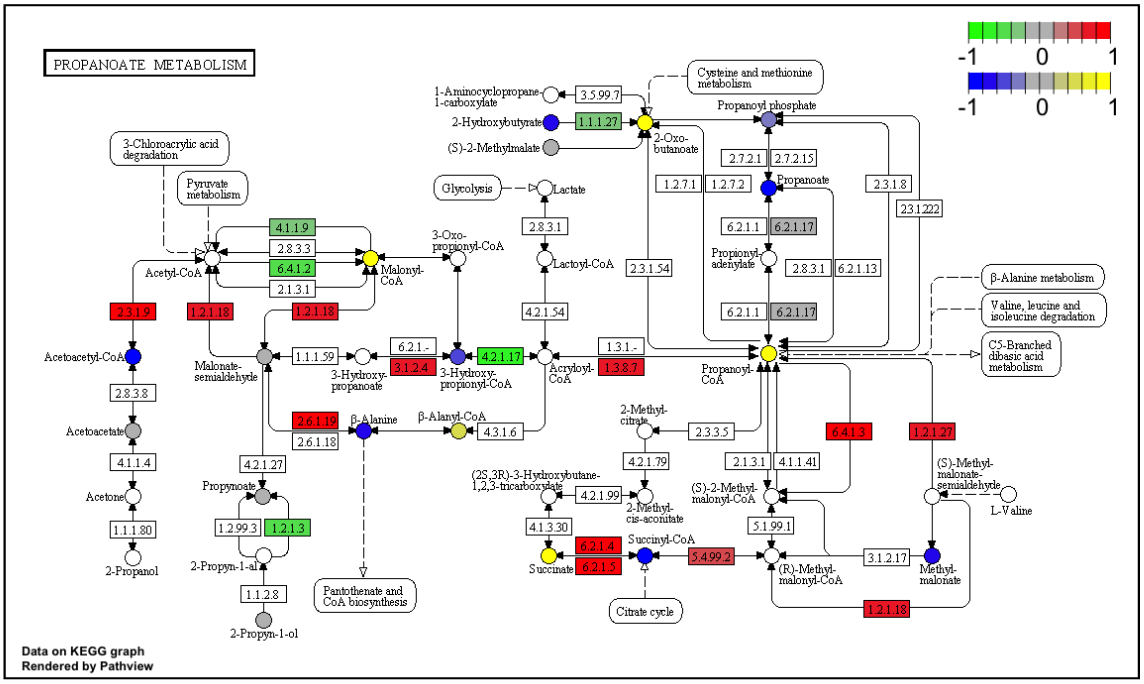
```
> head(pv.out$plot.data.cpd)
```

```
    kegg.names labels     type   x   y width height      mol.data mol.col
52      C04225 C04225 compound 646 495     8      8            NA #FFFFFF
110     C02614 C02614 compound 551 143     8      8  0.0003758095 #BEBEBE
111     C00109 C00109 compound 646 118     8      8  0.8255746672 #FFFF00
112     C02876 C02876 compound 771 115     8      8 -0.3550097157 #8F8FCE
113     C00163 C00163 compound 771 184     8      8 -1.5511608864 #0000FF
114     C01234 C01234 compound 551  90     8      8            NA #FFFFFF
```

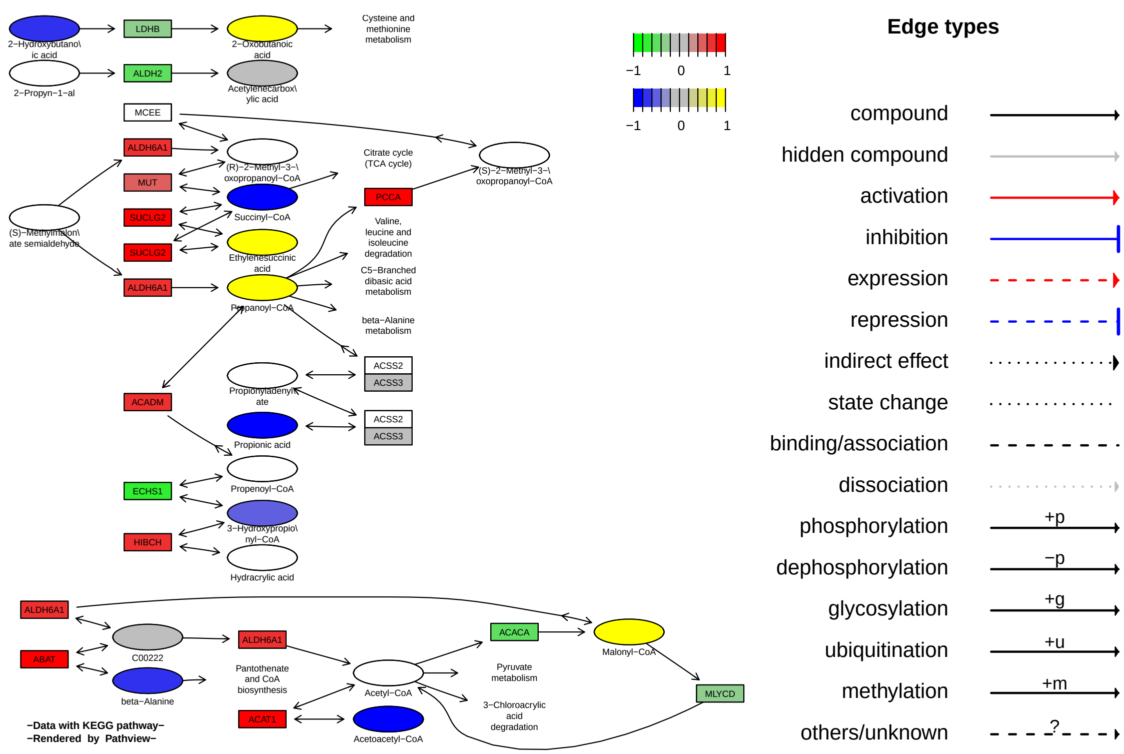
We also generate Graphviz view of the same pathway and data (Figure 5b). Graphviz view better shows the hierarchical structure. For metabolic pathways, we need to parse the reaction entries from xml files and convert it to relationships between gene and compound nodes. We use ellipses for compound nodes. 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])
```

In all previous examples, we looked at single sample data, which are either vector or single-column matrix. *Pathview* also handles multiple sample data, generates graph for each sample. It automatically match samples by recycling over the smaller sample size when sample sizes are different for gene and compound data. In the



(a)



(b)

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

following example (Figure not shown), `gene.data` has two samples while `cpd.data` has one. Hence `cpd.data` is recycled to match the second `gene.data` sample in the second graph. To prevent such data matching/recycling, you need to set argument `match.data=F`, which is passed to secondary function `keggview.native` or `keggview.graph` by `pathview`.

```
> head(gse16873.d[, 1:2])

               DCIS_1      DCIS_2
10000     -0.30764480 -0.14722769
10001      0.41586805 -0.33477259
10002      0.19854925  0.03789588
10003     -0.23155297 -0.09659311
100048912 -0.04490724 -0.05203146
10004     -0.08756237 -0.05027725

> pv.out <- pathview(gene.data = gse16873.d[, 1:2], 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])
> head(pv.out$plot.data.gene)

   kegg.names  labels type   x   y width height     DCIS_1      DCIS_2
54       4329 ALDH6A1 gene 936 430    46     17 0.74686683  0.05287812
55       4329 ALDH6A1 gene 890 609    46     17 0.74686683  0.05287812
57      84693    MCEE gene 773 525    46     17         NA          NA
58       5095    PCCA gene 852 430    46     17 1.19029289 -0.30087442
62      79611   ACSS3 gene 796 309    46     17 0.07325135 -0.21532204
63      79611   ACSS3 gene 796 223    46     17 0.07325135 -0.21532204
   DCIS_1.col DCIS_2.col
54    #EF3030    #BEBEBE
55    #EF3030    #BEBEBE
57    #FFFFFF    #FFFFFF
58    #FF0000    #8FCE8F
62    #BEBEBE    #8FCE8F
63    #BEBEBE    #8FCE8F
```

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 6) 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),
```

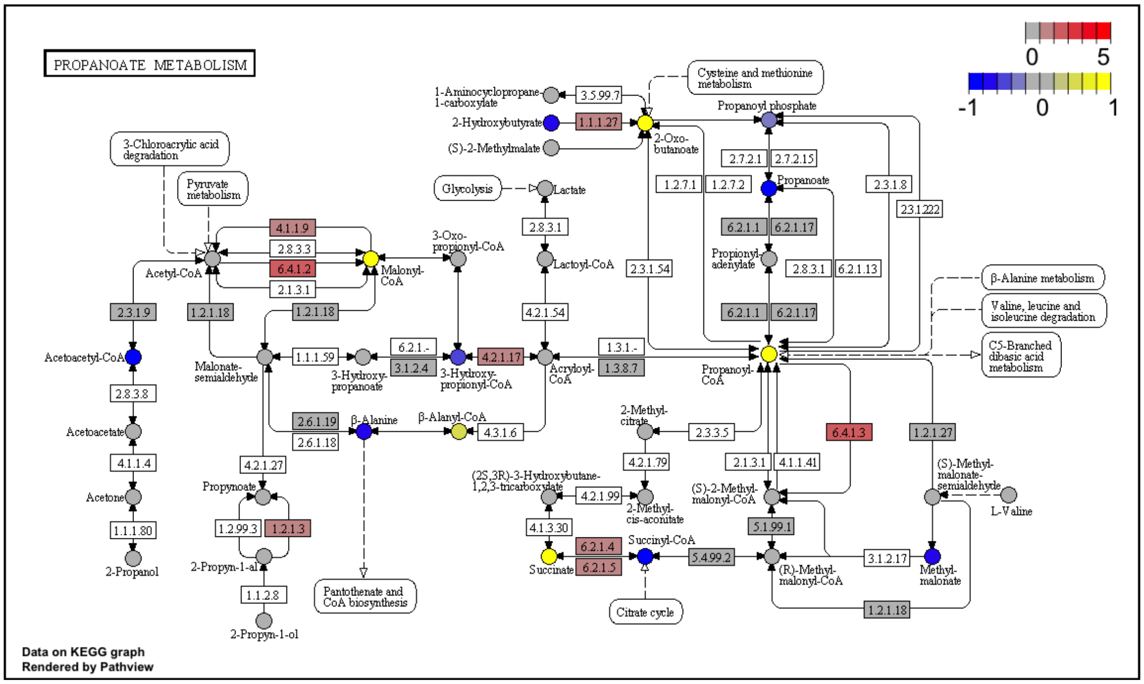


Figure 6: Example native KEGG view on discrete gene data and continuous compound data simultaneously.

```
+     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))
```

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 over 2000 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 7). 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$sel.paths[i], species = "hsa", same.layer = T,
+     out.suffix = "gene.ensprot.cpd.cas", keys.align = "y", kegg.native = T,
+     key.pos = demo.paths$kpos2[i], sign.pos = demo.paths$spos[i],
+     limit = list(gene = 3, cpd = 3), bins = list(gene = 6, cpd = 6))
```

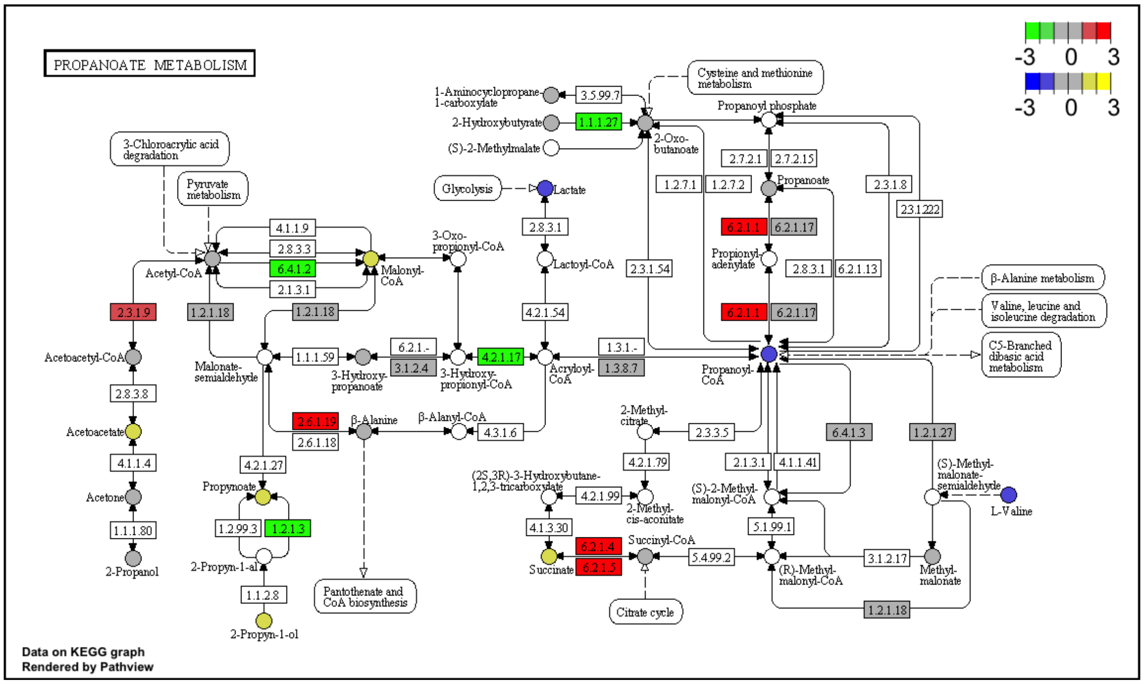


Figure 7: Example native KEGG view on gene data and compound data with other ID types.

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))
```

Importantly, *pathview* can be directly used for metagenomic or microbiome data when the data are mapped to KEGG ortholog pathways. 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)

[1] "Downloading xml files for ko04112, 1/1 pathways.."
[1] "Downloading png files for ko04112, 1/1 pathways.."
```

6 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).

```
> library(gage)
> data(gse16873)
> cn <- colnames(gse16873)
> hn <- grep('HN',cn, ignore.case =TRUE)
> dcis <- grep('DCIS',cn, ignore.case =TRUE)
> kgs.file <- system.file("extdata", "kegg.sigmet.rda", package = "pathview")
> load(kgs.file)
> gse16873.kegg.p <- gage(gse16873, gsets = kegg.sigmet,
+     ref = hn, samp = dcis)
> gse16873.d <- gagePrep(gse16873, ref = hn, samp = dcis)
> 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)
> pv.out.list <- sapply(path.ids2, function(pid) pathview(gene.data = gse16873.d[,
+     1:2], pathway.id = pid, species = "hsa"))
```

7 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*, 2013. doi: 10.1093/bioinformatics/btt285.

Weijun Luo, Michael Friedman, Kerby Shedden, Kurt Hankenson, and Peter Woolf. GAGE: generally applicable gene set enrichment for pathway analysis. *BMC Bioinformatics*, 2009.

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