# Package 'paircompviz'

October 24, 2025

**Type** Package

**Title** Multiple comparison test visualization

**Version** 1.47.0

**Date** 2013-06-01

**Author** Michal Burda

**Maintainer** Michal Burda <michal.burda@osu.cz>

**Description** This package provides visualization of the results from the multiple (i.e. pairwise) comparison tests such as pairwise.t.test, pairwise.prop.test or pairwise.wilcox.test. The groups being compared are visualized as nodes in Hasse diagram. Such approach enables very clear and vivid depiction of which group is significantly greater than which others, especially if comparing a large number of groups.

**Imports** Rgraphviz

**Depends** R (>= 2.10), Rgraphviz

**Suggests** multcomp, reshape, rpart, plyr, xtable

**License** GPL (>=3.0)

**biocViews** GraphAndNetwork

**git_url** https://git.bioconductor.org/packages/paircompviz

**git_branch** devel

**git_last_commit** df4863e

**git_last_commit_date** 2025-04-15

**Repository** Bioconductor 3.23

**Date/Publication** 2025-10-24

## Contents

---

`paircompviz-package` *Multiple comparison test visualization*

---

## Description

This package provides visualization of the results from the multiple (i.e. pairwise) comparison tests such as pairwise.t.test, pairwise.prop.test or pairwise.wilcox.test. The groups being compared are visualized as nodes in Hasse diagram. Such approach enables very clear and vivid depiction of which group is significantly greater than which others, especially if comparing a large number of groups.

## Author(s)

Michal Burda <michal.burda@osu.cz>

## Examples

```
paircomp(InsectSprays$count, InsectSprays$spray, test="t")
```

---

`brokentrans` *Artificial dataset that suffers with broken transitivity of the pairwise t-test comparisons*

---

## Description

This is a dataset of artificial data created to demonstrate that there exists a data sample such that pairwise comparisons using t-test break transitivity of the results, i.e. that if treatment 1 is significantly lower than 2 and treatment 2 is lower than 3, it is not always the case that also treatment 1 is significantly lower than 3.

## Usage

```
data(brokentrans)
```

## Format

A data frame of two columns:

1. x is the measured value,
2. g is the treatment group.

## Examples

```
data(brokentrans)

# For \alpha = 10^{-9}, we obtain significant difference
# between 1-2, 2-3, but not 1-3.
tapply(brokentrans$x, brokentrans$g, mean)
pairwise.t.test(brokentrans$x, brokentrans$g, pool.sd=FALSE)
```

---

hasse *Visualization of Hasse diagram specified by an adjacency matrix*

---

## Description

Given an adjacency matrix, this function displays the corresponding Hasse diagram. This is a wrapper function for graph creation using the **Rgraphviz** package.

## Usage

```
hasse(e,
      v=NULL,
      elab="",
      ecol="black",
      ebg="gray",
      vcol="black",
      vbg="white",
      vsize=1,
      fvlab=".",
      fvcol="black",
      fvbg="white",
      fvsize=1,
      febg="black",
      fesize=1,
      main=paste("Hasse Diagram of", deparse(substitute(e))),
      compress=FALSE)
```

## Arguments

| | |
|---|---|
| e | An adjacency matrix, with $e_{i,j}$ indicating the edge size between vertices $i$ and $j$ ($e_{i,j} = 0$ means no edge between $i$ and $j$). The matrix must be rectangular with non-negative non-missing values. |
| v | Vector of names of the vertices. If null, the vertex names will be obtained from column names of adjacency matrix $e$. |
| elab | Labels of the edges. If it is a scalar value, all edges would have the same label. Otherwise, $elab$ must be a rectangular matrix (similar to adjacency matrix $e$). A value on $i$-th row and $j$-th column is a label of the edge between vertex $i$ and vertex $j$. |

| | |
|---|---|
| ecol | Edge label color. If scalar, all edge labels have the same color. Otherwise, $ecol$ must be in the form of adjacency matrix: a value on $i$-th row and $j$-th column is a color of the label of the edge between vertex $i$ and vertex $j$. |
| ebg | Edge line color. If scalar, all edges have the same color. Otherwise, $ebg$ must be in the form of adjacency matrix: a value on $i$-th row and $j$-th column is a color of the edge between vertex $i$ and vertex $j$. |
| vcol | Vertex label color. If scalar, all vertices have the same label color. Otherwise, $vcol$ must be a vector of the size corresponding to the number of vertices. |
| vbg | Vertex background color. If scalar, all vertices have the same background color. Otherwise, $vcol$ must be a vector of the size corresponding to the number of vertices. |
| vsize | Vertex sizes. If scalar, all vertices have the same size in the image. Otherwise, $vsize$ must be a vector of the size corresponding to the number of vertices. |
| fvlab | Labels of "dot" vertices. Must be scalar. |
| fvcol | "dot" vertex label color. Must be scalar. |
| fvbg | "dot" vertex background color. Must be scalar. |
| fvsize | "dot" vertex size. Must be scalar. |
| febg | Color of edges introduced by edge compression. Must be scalar. |
| fesize | Thickness of edges introduced by edge compression. Must be scalar and non-negative. |
| main | Main title of the diagram. |
| compress | TRUE if the edges should be compressed, i.e. if the maximum bi-cliques have to be found in the graph and replaced with a "dot" vertex. (See examples.) |

## Details

This function depicts a Hasse diagram specified with an adjacency matrix $e$. Hasse diagram is a visualization of partially ordered set, by drawing its transitive reduction as an oriented graph. Each vertex corresponds to an element of the set. There is an edge between vertex $i$ and vertex $j$ iff $i < j$ and there is no $z$ such that $i < z < j$.

The function is also capable of edge compression via introducing the "dot" edges: Let $U, V$ be two disjoint non-empty sets of edges, such that for each $u$ from $U$ and $v$ from $V$, there exists an edge from $u$ to $v$. (The number of such edges equals $|U| \cdot |V|$.) Starting from $|U| > 2$ and $|V| > 2$, the Hasse diagram may become too complicated and hence confusing. Therefore a `compress` argument exists in this function that enables "compression" of the edges in such a way that a new "dot" node $w$ is introduced and $|U| \cdot |V|$ edges between sets $U$ and $V$ are replaced with $|U| + |V|$ edges from set $U$ to node $w$ and from node $w$ to set $V$.

## Value

Nothing.

## Author(s)

Michal Burda

## See Also

paircomp

## Examples

```
# linear order
e <- matrix(c(0, 1, 1, 0, 0, 1, 0, 0, 0), nrow=3, byrow=TRUE)
hasse(e)

# prepare adjacency matrix
m <- matrix(0, byrow=TRUE, nrow=5, ncol=5)
m[3, 1] <- 1
m[3, 2] <- 1
m[4, 1] <- 9
m[4, 2] <- 1
m[5, 1] <- 1
m[5, 2] <- 1
m

mc <- m
mc[mc > 0] <- "red"
ms <- m
ms[ms > 0] <- "blue"

# view m with default settings
hasse(m, ebg="black")

# view m WITHOUT edge compression and some fancy adjustments
hasse(v=c("a", "b", "c", "d", "e"),
           vcol=c(gray(0.5), gray(1), rep(gray(0), 3)),
           vbg=gray(5:1/5), vsize=1:5, e=m, ecol=mc, ebg=ms, elab=m,
           compress=FALSE)

# view m WITH edge compression and some fancy adjustments
hasse(v=c("a", "b", "c", "d", "e"),
           vcol=c(gray(0.5), gray(1), rep(gray(0), 3)),
           vbg=gray(5:1/5), vsize=1:5, e=m, ecol=mc, ebg=ms, elab=m,
           compress=TRUE)
```

---

paircomp *Visualization of multiple pairwise comparison test results*

---

## Description

This function performs multiple pairwise comparison tests on given data and views the results in the form of Hasse diagram.

## Usage

```
paircomp(obj,
         grouping=NULL,
         test=c("t", "prop", "wilcox"),
         level=0.05,
         main=NULL,
         compress=TRUE,
         visualize=c("position", "size", "pvalue"),
         result=FALSE,
         draw=TRUE,
         ...)
```

## Arguments

| | |
|---|---|
| obj | either a vector or an object of class glht as returned from the glht function of the multcomp package.<br>If obj is an object of class glht, then arguments grouping and test may be arbitrary, because they will be not used. Otherwise, if test equals "t" or "wilcox", obj should be a numeric vector of responses, and if test equals "prop", obj should be a vector of 0's and 1's. |
| grouping | a grouping factor. If obj is a numeric vector, grouping must be a factor. If obj is an object of class glht, grouping should be NULL. |
| test | a name of the test to use. If obj is an object of class glht, the value of test does not have any effect. Otherwise, the values determine the type of the pairwise comparison test procedure. Allowed values "t", "prop" or "wilcox" imply internal call of pairwise.t.test(), pairwise.prop.test() or pairwise.wilcox.test(), respectively. |
| level | the maximum p-value that will be considered as significant; i.e. pairwise test results with p-value lower than the specified level will be represented with an edge in the resulting Hasse diagram. |
| main | main title of the diagram. |
| compress | TRUE if the edges should be compressed, i.e. if the maximum bi-cliques have to be found in the graph and replaced with a "dot" vertex. (See examples.) |
| visualize | vector of additional information to be included in the diagram: "position" enables vertex background color to be derived from the treatment's proportion ("prop" test) or mean value (otherwise); "size" enables vertex size to correspond to the treatment's sample size; "pvalue" sets the edges' line thickness accordingly to p-value (lower p-value corresponds to thicker line). |
| result | whether to return test results as a return value. |
| draw | whether to render the diagram. |
| ... | other arguments that will be passed to the underlying function that performs pairwise comparisons (e.g. pairwise.t.test, pairwise.prop.test or pairwise.wilcox.test. |

## Details

All treatments in a set are compared in pairs using a selected statistical test. If the results form a partially ordered set, they can be viewed in a Hasse diagram.

Hasse diagram is a graph with each treatment being represented as a vertex. An edge is drawn downwards from vertex $y$ to vertex $x$ if and only if treatment $x$ is significantly lower than treatment $y$, and there is no such treatment $z$ that $x$ was lower than $z$ and $z$ lower than $y$. Each edge is connected to exactly two vertices: its two endpoints. If there does not exist a path between some two treatments, it means that these two treatments are incomparable (i.e. the difference among them is not statistically significant).

The function accepts two types of inputs: either an instance of class `glht` or a vector `obj` of measured values and a factor `grouping` of treatments.

The `glht` object may be obtained from function `glht` of the **multcomp** package and set as the `obj` argument. Argument `grouping` must be `NULL`, in that case.

If `obj` is a numeric vector of measured values, `grouping` must not be `NULL` and also a type of statistical test must be selected by setting `test` argument.

Edge compression (introducing "dot" edges):

Sometimes, pairwise comparison tests may yield in such bipartite setting that each pair of nodes of some two node subsets would be inter-connected with an edge (without any edge between nodes in the same subset). More specifically, let $U$, $V$ be two disjoint non-empty sets of edges, such that for each $u$ from $U$ and $v$ from $V$, there exists an edge from $u$ to $v$. (The number of such edges equals $|U| \cdot |V|$.) Starting from $|U| > 2$ and $|V| > 2$, the Hasse diagram may become too complicated and hence confusing. Therefore a `compress` argument exists in this function that enables "compression" of the edges in such a way that a new "dot" node $w$ is introduced and $|U| \cdot |V|$ edges between sets $U$ and $V$ are replaced with $|U| + |V|$ edges from set $U$ to node $w$ and from node $w$ to set $V$.

## Value

If argument `result` is `TRUE`, the function returns everything that is returned by the underlying test function (`pairwise.t.test`, `pairwise.prop.test` or `pairwise.wilcox.test` accordingly to the `test` argument), or a copy of the `obj` argument, if `obj` is an instance of class `glht`.

## Author(s)

Michal Burda

## See Also

`pairwise.t.test`, `pairwise.prop.test`, `pairwise.wilcox.test`, `glht` `hasse`

## Examples

```
# Example of test="prop":
o <- c(rep(1, 10), rep(0, 10), rep(c(0,1), 5))
g <- c(rep(1,10), rep(2, 10), rep(3, 10))
paircomp(o, g, test="prop")

# Example of test="t" and test="wilcox":
```

```
paircomp(InsectSprays$count, InsectSprays$spray, test="t")
paircomp(InsectSprays$count, InsectSprays$spray, test="wilcox")

# Example of t-test with non-pooled SD and Bonferroni adjustment
# for multiple comparisons:
paircomp(InsectSprays$count, InsectSprays$spray, test="t",
         pool.sd=FALSE, p.adjust.method="bonferroni")

# Compare diagrams with and without compressed edges:
paircomp(InsectSprays$count, InsectSprays$spray, test="t",
         compress=FALSE)
paircomp(InsectSprays$count, InsectSprays$spray, test="t",
         compress=TRUE)

# perform Tukey test:
library(rpart) # for car90 dataset
library(multcomp) # for glht() function
aovR <- aov(Price ~ Type, data = car90)
glhtR <- glht(aovR, linfct = mcp(Type = "Tukey"))
paircomp(glhtR)
```

---

`transReduct` *Remove transitive edges from an adjacency matrix*

---

## Description

This function removes transitive edges from an adjacency matrix.

## Usage

```
transReduct(e)
```

## Arguments

| | |
|---|---|
| e | an adjacency matrix, i.e. a rectangular matrix with value $e_{i,j}$ above zero indicating an edge between vertices $i$ and $j$ of the corresponding graph. |

## Details

This function takes an adjacency matrix as the argument e. Both rows and columns correspond to graph vertices, with value $e_{i,j}$ above zero indicating an edge between vertices $i$ and $j$. The function removes all transitive edges, i.e. sets to zero corresponding elements of matrix $e$. The transitive edge is such an edge between vertices $i$ and $j$ that after removing it from the graph, there still exists a path from $i$ to $j$.

## Value

An adjacency matrix $e$ with transitive edges being removed.

## Author(s)

Michal Burda

## See Also

paircomp, hasse

## Examples

```
e <- matrix(c(0, 1, 1, 0, 0, 1, 0, 0, 0), nrow=3, byrow=TRUE)
transReduct(e)
```

# Index