Package ‘ggip’

April 4, 2023

Title  Data Visualization for IP Addresses and Networks

Version  0.3.2

Description  A ‘ggplot2’ extension that enables visualization of IP
             (Internet Protocol) addresses and networks. The address space is
             mapped onto the Cartesian coordinate system using a space-filling
             curve. Offers full support for both IPv4 and IPv6 (Internet Protocol
             versions 4 and 6) address spaces.

License  MIT + file LICENSE

     https://github.com/davidchall/ggip

BugReports  https://github.com/davidchall/ggip/issues

Depends  ggplot2 (>= 3.4.0), ipaddress (>= 1.0.0)

Imports  cli, dplyr, Rcpp, rlang (>= 1.0.0), tidyr, vctrs

Suggests  knitr, rmarkdown, testthat

LinkingTo  ipaddress, Rcpp

VignetteBuilder  knitr

Config/testthat/edition  3

Encoding  UTF-8

RoxygenNote  7.2.3

NeedsCompilation  yes

Author  David Hall [aut, cre] (<https://orcid.org/0000-0002-2193-0480>)

Maintainer  David Hall <david.hall.physics@gmail.com>

Repository  CRAN

Date/Publication  2023-04-04 04:10:02 UTC
## coord_ip

**Coordinate system for IP data**

A `ggplot2` coordinate system that maps a range of IP address space onto a two-dimensional grid using a space-filling curve.

`coord_ip()` forms the foundation of any ggip plot. It translates all `ip_address` and `ip_network` vectors to Cartesian coordinates, ready for use by `ggplot2` layers (see Accessing Coordinates). This ensures all layers use a common mapping.

### Usage

```r
coord_ip(
  canvas_network = ip_network("0.0.0.0/0"),
  pixel_prefix = 16,
  curve = c("hilbert", "morton"),
  expand = FALSE
)
```

### Arguments

- **canvas_network**: An `ip_network` scalar that determines the region of IP space visualized by the entire 2D grid. The default shows the entire IPv4 address space.
- **pixel_prefix**: An integer scalar that sets the prefix length of the network represented by a single pixel. The default value is 16. Increasing this effectively improves the resolution of the plot.
- **curve**: A string to choose the space-filling curve. Choices are "hilbert" (default) and "morton".
- **expand**: If TRUE, adds a small expanded margin around the data grid. The default is FALSE.

### Accessing Coordinates

`coord_ip()` stores the result of the mapping in a nested data frame column. This means each `ip_address` or `ip_network` column in the original data set is converted to a data frame column. When specifying `ggplot2` aesthetics, you'll need to use `$` to access the nested data (see Examples). Each `ip_address` column will be replaced with a data frame containing the following columns:
coord_ip

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip</td>
<td>ip_address</td>
<td>Original IP data</td>
</tr>
<tr>
<td>x</td>
<td>integer</td>
<td>Pixel x</td>
</tr>
<tr>
<td>y</td>
<td>integer</td>
<td>Pixel y</td>
</tr>
</tbody>
</table>

Each ip_network column will be replaced with a data frame containing the following columns:

<table>
<thead>
<tr>
<th>Column name</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip</td>
<td>ip_network</td>
<td>Original IP data</td>
</tr>
<tr>
<td>xmin</td>
<td>integer</td>
<td>Bounding box xmin</td>
</tr>
<tr>
<td>ymin</td>
<td>integer</td>
<td>Bounding box ymin</td>
</tr>
<tr>
<td>xmax</td>
<td>integer</td>
<td>Bounding box xmax</td>
</tr>
<tr>
<td>ymax</td>
<td>integer</td>
<td>Bounding box ymax</td>
</tr>
</tbody>
</table>

See Also

vignette("visualizing-ip-data") describes the mapping in more detail.

Examples

```r
suppressPackageStartupMessages(library(dplyr))

tibble(address = ip_address(c("0.0.0.0", "128.0.0.0", "192.168.0.1"))) %>%
  ggplot(aes(x = address$x, y = address$y, label = address$ip)) +
  geom_point() +
  geom_label(nudge_x = c(10, 0, -10), nudge_y = -10) +
  coord_ip(expand = TRUE) +
  theme_ip_light()

tibble(network = ip_network(c("0.0.0.0/8", "224.0.0.0/4"))) %>%
  mutate(
    start = network_address(network),
    end = broadcast_address(network)
  ) %>%
  ggplot() +
  geom_point(aes(x = start$x, y = start$y), color = "blue") +
  geom_point(aes(x = end$x, y = end$y), color = "red") +
  geom_rect(
    aes(xmin = network$xmin, xmax = network$xmax, ymin = network$ymin, ymax = network$ymax),
    alpha = 0.5, fill = "grey"
  ) +
  coord_ip(curve = "morton", expand = TRUE) +
  theme_ip_light()
```
Description

Computes and draws the outline of the Hilbert curve used to map IP data to the Cartesian plane. By superimposing this outline on top of a ggplot plot, it guides the eye to regions that are close in IP address space.

Usage

```r
geom_hilbert_outline(
  mapping = NULL,
  data = NULL,
  ..., 
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)
```

Arguments

- **mapping**: Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply `mapping` if there is no plot mapping.

- **data**: The data to be displayed in this layer. There are three options:
  - If `NULL`, the default, the data is inherited from the plot data as specified in the call to `ggplot()`.
  - A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.
  - A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).

- **...**: Other arguments passed on to `layer()`. These are often aesthetics, used to set an aesthetic to a fixed value, like `colour = "red"` or `size = 3`. They may also be parameters to the paired geom/stat.

- **na.rm**: If `FALSE`, the default, missing values are removed with a warning. If `TRUE`, missing values are silently removed.

- **show.legend**: logical. Should this layer be included in the legends? `NA`, the default, includes if any aesthetics are mapped. `FALSE` never includes, and `TRUE` always includes. It can also be a named logical vector to finely select the aesthetics to display.

- **inherit.aes**: If `FALSE`, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.
Aesthetics

geom_curve_outline() understands the following aesthetics:

- **ip**: An ip_network column. By default, the entire Hilbert curve is shown.
- **curve_order**: How nested is the curve? (default: 3).
- **closed**: Should the curve outline have closed ends? (default: FALSE).
- **alpha**
- **colour**
- **linetype**
- **linewidth**

Computed variables

- **x**, **y** The start coordinates for the segment
- **xend**, **yend** The end coordinates for the segment

Examples

```r
p <- ggplot() + coord_ip() + theme_ip_light()

# default shows curve across entire canvas
p + geom_hilbert_outline()

# only show subnetwork
p + geom_hilbert_outline(ip = ip_network("128.0.0.0/2"))

# increased nesting
p + geom_hilbert_outline(curve_order = 4)

# show multiple networks
df <- data.frame(
    ip = ip_network(c("0.0.0.0/2", "128.0.0.0/4")),
    curve_order = c(4, 5),
    closed = c(FALSE, TRUE)
)

p + geom_hilbert_outline(
    aes(ip = ip, curve_order = curve_order, closed = closed),
    data = df
)
```

**Description**

These functions are used internally by coord_ip() to map ip_address and ip_network vectors to Cartesian coordinates. They are exposed externally to support use of these coordinates outside of ggplot2.
Usage

address_to_cartesian(
  address,
  canvas_network = ip_network("0.0.0.0/0"),
  pixel_prefix = 16,
  curve = c("hilbert", "morton")
)

network_to_cartesian(
  network,
  canvas_network = ip_network("0.0.0.0/0"),
  pixel_prefix = 16,
  curve = c("hilbert", "morton")
)

Arguments

address An ip_address vector

canvas_network An ip_network scalar that determines the region of IP space visualized by the entire 2D grid. The default shows the entire IPv4 address space.

pixel_prefix An integer scalar that sets the prefix length of the network represented by a single pixel. The default value is 16. Increasing this effectively improves the resolution of the plot.

curve A string to choose the space-filling curve. Choices are "hilbert" (default) and "morton".

network An ip_network vector

Value

A data.frame containing columns:

• address_to_cartesian(): x and y

• network_to_cartesian(): xmin, ymin, xmax and ymax

Examples

address_to_cartesian(ip_address("192.168.0.1"))

network_to_cartesian(ip_network("224.0.0.0/4"))
stat_summary_address  Summarize IP addresses on a heatmap

Description

Addresses are grouped into networks determined by the pixel_prefix argument of coord_ip(). Then the z values are summarized with summary function fun.

Usage

stat_summary_address(
  mapping = NULL,
  data = NULL,
  ..., #
  fun = NULL,
  fun.args = list(),
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE
)

Arguments

mapping  Set of aesthetic mappings created by aes(). If specified and inherit.aes = TRUE (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.

data  The data to be displayed in this layer. There are three options:

  If NULL, the default, the data is inherited from the plot data as specified in the call to ggplot(). A data.frame, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.

  A function will be called with a single argument, the plot data. The return value must be a data.frame, and will be used as the layer data. A function can be created from a formula (e.g. ~ head(.x, 10)).

...  Other arguments passed on to layer(). These are often aesthetics, used to set an aesthetic to a fixed value, like colour = "red" or size = 3. They may also be parameters to the paired geom/stat.

fun  Summary function (see section below for details). If NULL (the default), the observations are simply counted.

fun.args  A list of extra arguments to pass to fun.

na.rm  If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
show.legend logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.

inherit.aes If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn’t inherit behaviour from the default plot specification, e.g. `borders()`.

Aesthetics

`stat_summary_address()` understands the following aesthetics (required aesthetics are in bold):

- ip: An `ip_address` column
- z: Value passed to the summary function (required if fun is used)
- fill: Default is `after_stat(value)`
- alpha

Computed variables

The following variables are available to `after_stat()`:

- value: Value of summary statistic
- count: Number of observations

Summary function

The data might contain multiple rows per pixel of the heatmap, so a summary function reduces this information to a single value to display. This function receives the data column specified by the z aesthetic and also receives arguments specified by `fun.args`.

The `fun` argument can be specified in multiple ways:

- **NULL** If no summary function is provided, the number of observations is computed. In this case, you don’t need to specify the z aesthetic, and the computed variables value and count will be equal.
- **string** The name of an existing function (e.g. `fun = "mean"`).
- **function** Either provide an existing function (e.g. `fun = mean`) or define a new function (e.g. `fun = function(x) sum(x^2)`).
- **formula** A function can also be created from a formula. This uses .x as the summarized variable (e.g. `fun = ~ sum(.x^2)`).

Examples

```r
dat <- data.frame(
  ip = sample_ipv4(10000),
  weight = runif(10000)
)
p <- ggplot(dat, aes(ip = ip)) +
  coord_ip() +
```
theme_ip

theme_ip_light()

# simple count of observations
p +
  stat_summary_address() +
  scale_fill_viridis_c(trans = "log2", na.value = "black", guide = "none")

# compute mean weight
p +
  stat_summary_address(aes(z = weight), fun = ~ mean(.x)) +
  scale_fill_viridis_c(na.value = "black", guide = "none")

_________________________________________________________________

<table>
<thead>
<tr>
<th>theme_ip</th>
<th>Themes for IP data</th>
</tr>
</thead>
</table>

Description

These set sensible defaults for plots generated by ggip. Use `ggplot2::theme()` if you want to tweak the results.

Usage

theme_ip_light(base_size = 11, base_family = "")

theme_ip_dark(
  background_color = "black",
  text_color = "white",
  base_size = 11,
  base_family = ""
)

Arguments

- base_size: base font size, given in pts.
- base_family: base font family
- background_color: Background color
- text_color: Text color

Examples

p <- ggplot(data.frame(ip = ip_address("128.0.0.0"))) +
  geom_point(aes(x = ip$x, y = ip$y), color = "grey") +
  coord_ip()

p + theme_ip_light()

p + theme_ip_dark()
Index

address_to_cartesian(ip_to_cartesian), 5
aes(), 4, 7
after_stat(), 8
borders(), 4, 8
coord_ip, 2
coord_ip(), 5
fortify(), 4, 7
geom_hilbert_outline, 4
ggplot(), 4, 7
ggplot2::theme(), 9
ip_address, 2, 5, 6, 8
ip_network, 2, 3, 5, 6
ip_to_cartesian, 5
layer(), 4, 7
network_to_cartesian(ip_to_cartesian), 5
stat_summary_address, 7
theme_ip, 9
theme_ip_dark(theme_ip), 9
theme_ip_light(theme_ip), 9