

Package: greenAlgoR (via r-universe)

October 5, 2026

Title Carbon Footprint Estimation for 'R' Computations

Version 0.2.0.9000

Description Computes the carbon footprint and ecological impact of computational tasks in 'R'. Based on the 'Green Algorithms' framework (Lannelongue et al. 2021, <https://calculator.green-algorithms.org/>), this package provides functions to estimate energy consumption and CO2 emissions from 'R' computations. It includes specialized support for 'targets' pipelines and provides visualization tools for carbon footprint analysis. The package helps researchers and data scientists to measure the environmental impact of their computational work.

License AGPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

URL <https://github.com/adrientaudiere/greenAlgoR>,
<https://adrientaudiere.github.io/greenAlgoR/>

BugReports <https://github.com/adrientaudiere/greenAlgoR/issues>

Depends R (>= 3.5.0)

Imports benchmarkme, dplyr, targets, ggplot2 (>= 3.5.0), utils

Suggests autometric, ggnetwork, here, knitr, network, RCurl,
rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libglpk-dev libicu-dev libxml2-dev libssl-dev

Repository <https://adrientaudiere.r-universe.dev>

Date/Publication 2026-07-07 20:00:00 UTC

RemoteUrl <https://github.com/adrientaudiere/greenAlgoR>

RemoteRef HEAD

RemoteSha b8dcf1a2d3fa154db002cc8bc37e9b7f18c7a822

Contents

greenAlgoR-package	2
csv_from_url_ga	3
ga_autometric_history	4
ga_autometric_lastrun	6
ga_autometric_plot	7
ga_footprint	8
ga_targets	13
ga_targets_meta_plot	16
ga_targets_network	18
round_conditionaly	20
session_runtime	21
Index	22

greenAlgoR-package	greenAlgoR <i>package</i>
--------------------	---------------------------

Description

Carbon Footprint Estimation for R Computations

The greenAlgoR package provides tools to estimate the carbon footprint and energy consumption of computational tasks in R. Based on the Green Algorithms framework developed by Lannelongue et al. (2021), this package helps researchers and data scientists understand and minimize the environmental impact of their work.

Internal Data

The package includes some internal datasets to work outline. You can access them directly after loading the package (TDP_cpu_internal, carbon_intensity_internal and ref_value_internal). You can also replace default data by overwriting them. By default, data are loaded using functions:

```
TDP_cpu_internal <- csv_from_url_ga("https://raw.githubusercontent.com/GreenAlgorithms/GA-
data/5266caba6601dae0ffc93af8971e758f55292e08/v3.0/CPU.csv")

carbon_intensity_internal <- csv_from_url_ga("https://raw.githubusercontent.com/GreenAlgorithms/GA-
data/5266caba6601dae0ffc93af8971e758f55292e08/v3.0/CI_aggregated.csv")

ref_value_internal <- csv_from_url_ga("https://raw.githubusercontent.com/GreenAlgorithms/GA-
data/5266caba6601dae0ffc93af8971e758f55292e08/v3.0/referenceValues.csv")
```

Main Functions

- `ga_footprint`: Calculate carbon footprint for individual computations
- `ga_targets`: Calculate carbon footprint for targets pipelines
- `session_runtime`: Compute session runtime and memory usage

Key Features

- Estimate CO2 emissions based on runtime, CPU usage, and memory consumption
- Support for different geographical locations with varying carbon intensities
- Integration with the targets package for pipeline analysis
- Visualization tools for carbon footprint comparisons
- Configurable hardware specifications (CPU models, memory, storage)

Getting Started

To get started with greenAlgoR, try:

```
# Basic usage - estimate footprint of a 12-hour computation
result <- ga_footprint(runtime_h = 12, location_code = "WORLD")

# For your current R session
session_footprint <- ga_footprint(runtime_h = "session")

# For targets pipelines (in a targets project)
targets_footprint <- ga_targets()
```

Author(s)

Adrien Taudière <adrien.taudiere@zaclys.net>

References

Lannelongue, L., Grealey, J., Inouye, M. (2021). Green Algorithms: Quantifying the Carbon Footprint of Computation. *Advanced Science*, 8(12), 2100707. doi:[10.1002/advs.202100707](https://doi.org/10.1002/advs.202100707)

csv_from_url_ga

Load CSV files from Green Algorithms GitHub repositories

Description

Helper function to download and parse CSV data from the Green Algorithms project repositories. This function handles the specific format used by Green Algorithms data files, which often have headers in specific rows.

Usage

```
csv_from_url_ga(url, remove_first_line = TRUE)
```

Arguments

url Character string with URL to a raw CSV file from Green Algorithms repository

remove_first_line Logical (default TRUE). Whether to remove the first line from the CSV file (often contains metadata rather than column headers).

Value

A data.frame with properly formatted column names and data

Author(s)

Adrien Taudière

Examples

```
## Not run:
# Download carbon intensity data
carbon_intensity <- csv_from_url_ga(
  paste0(
    "https://raw.githubusercontent.com/GreenAlgorithms/GA-data",
    "/5266caba6601dae0ffc93af8971e758f55292e08/v3.0/CI_aggregated.csv"
  )
)
head(carbon_intensity)

## End(Not run)
```

ga_autometric_history *Plot the full history of every logged run, including reruns*

Description

Draw a Gantt-style timeline of **every** process run recorded in an autometric log, faceted by pipeline session. Each (pid, phase) execution becomes a horizontal segment from its start to its end time, colored by mean CPU usage.

A log usually accumulates several pipeline executions that can be days or weeks apart, so plotting them on one absolute time axis would squeeze every run into an invisible sliver. Runs are therefore grouped into **sessions** (consecutive runs separated by a gap larger than session_gap) and drawn in their own facet with a session-relative time axis. A target rebuilt in several sessions appears in several facets, which exposes the trial-and-error and re-run cost that the "last run only" views ([ga_targets_meta_plot\(\)](#), [ga_autometric_lastrun\(\)](#)) hide by design. This full-history view is the closest to greenAlgoR's original motivation of accounting for the *whole* development cost of a pipeline, not just its final build.

Usage

```
ga_autometric_history(
  log,
  units_time = "hours",
  units_memory = "gigabytes",
  session_gap = 1,
  min_duration = 0,
  ...
)
```

Arguments

log	Either a path to an autometric log file (read with <code>autometric::log_read()</code>) or a data frame already returned by <code>autometric::log_read()</code> (with at least the columns <code>pid</code> , <code>phase</code> , <code>time</code> , <code>cpu</code>).
units_time, units_memory	Passed to <code>autometric::log_read()</code> when log is a file path (defaults "hours" and "gigabytes").
session_gap	(numeric, default 1) Gap, expressed in <code>units_time</code> , above which two consecutive runs are considered to belong to different pipeline sessions (facets). Increase it to merge runs into fewer sessions, decrease it to split more finely. Set to <code>Inf</code> to force a single session.
min_duration	(numeric, default 0) Drop runs shorter than this duration (in <code>units_time</code>). Useful to hide near-instant phases and keep the figure legible.
...	Additional arguments forwarded to <code>autometric::log_read()</code> .

Value

A [ggplot](#) object: one facet per session, each with one segment per logged run.

Author(s)

Adrien Taudière

See Also

[ga_autometric_lastrun\(\)](#), [ga_targets_meta_plot\(\)](#), [ga_autometric_plot\(\)](#)

Examples

```
if (requireNamespace("autometric", quietly = TRUE)) {
  log_df <- data.frame(
    version = "0.1.2",
    phase = c(rep("raw", 4), rep("scaled", 4), rep("raw", 4)),
    pid = c(rep(1L, 4), rep(2L, 4), rep(3L, 4)),
    name = "local",
    status = 0L,
    # Two sessions: the first at t ~ 0 h, a "raw" rerun ~ 5 h later.
    time = c(0, 0.1, 0.2, 0.3, 0.3, 0.35, 0.4, 0.45, 5.0, 5.1, 5.2, 5.3),
    core = runif(12, 0, 40),
    cpu = runif(12, 0, 90),
    resident = runif(12, 70, 90),
    virtual = runif(12, 900, 1000)
  )
  ga_autometric_history(log_df)
}
```

ga_autometric_lastrun *Plot the last-run duration of every stored target from an autometric log*

Description

Reconstruct the resource impact of the **last** pipeline run from an autometric log. Each (pid, phase) execution is collapsed to its duration and mean CPU usage, the log is restricted to phases that are still present in the targets store (targets::tar_objects()), and for each target only the single longest run is kept. The result is one bar per stored target, mapping duration to bar length and mean CPU usage to fill.

This is the log-based companion of [ga_targets_meta_plot\(\)](#): the former reads the metadata, the latter re-derives the same "last run" picture from the finer-grained resource samples. To visualize *every* run instead of only the last, use [ga_autometric_history\(\)](#).

Usage

```
ga_autometric_lastrun(
  log,
  store = targets::tar_config_get("store"),
  object_names = NULL,
  units_time = "hours",
  units_memory = "gigabytes",
  ...
)
```

Arguments

log	Either a path to an autometric log file (read with autometric::log_read()) or a data frame already returned by autometric::log_read() (with at least the columns pid, phase, time, cpu).
store	(character, default targets::tar_config_get("store")) Path to the targets data store, used to list the objects still present via targets::tar_objects().
object_names	(optional character) Vector of target names to keep. When supplied, store is not queried. Useful for tests or to focus on a subset.
units_time, units_memory	Passed to autometric::log_read() when log is a file path (defaults "hours" and "gigabytes").
...	Additional arguments forwarded to autometric::log_read().

Value

A [ggplot](#) object with one bar per stored target.

Author(s)

Adrien Taudière

See Also

[ga_targets_meta_plot\(\)](#), [ga_autometric_history\(\)](#), [ga_autometric_plot\(\)](#)

Examples

```
if (requireNamespace("autometric", quietly = TRUE)) {
  log_df <- data.frame(
    version = "0.1.2",
    phase = rep(c("raw", "scaled"), each = 4),
    pid = rep(c(1L, 2L), each = 4),
    name = "local",
    status = 0L,
    time = c(0, 0.1, 0.2, 0.3, 0.3, 0.35, 0.4, 0.45),
    core = runif(8, 0, 40),
    cpu = runif(8, 0, 90),
    resident = runif(8, 70, 90),
    virtual = runif(8, 900, 1000)
  )
  ga_autometric_lastrun(log_df, object_names = c("raw", "scaled"))
}
```

ga_autometric_plot	<i>Plot resource-usage logs recorded by the autometric package</i>
--------------------	--

Description

Render a [ggplot](#) of the CPU and memory usage recorded by `autometric::log_start()` / `autometric::log_stop()`. Each selected metric is drawn as a line over time, faceted so that percentage metrics (CPU) and megabyte metrics (memory) keep their own y-axis scale. This turns an autometric log into a publication-ready resource-usage figure and pairs naturally with the carbon-footprint estimates of [ga_footprint\(\)](#).

Usage

```
ga_autometric_plot(
  log,
  metrics = c("cpu", "resident"),
  color_by = "phase",
  ...
)
```

Arguments

log	Either a path to an autometric log file (read with <code>autometric::log_read()</code>) or a data frame already returned by <code>autometric::log_read()</code> (with at least the columns <code>time</code> , <code>cpu</code> , <code>core</code> , <code>resident</code> , <code>virtual</code>).
-----	--

metrics	(character, default c("cpu", "resident")) Which columns to plot. Any subset of "cpu" (\ "resident" (resident memory, MB) and "virtual" (virtual memory, MB).
color_by	(character, default "phase") Column used to color the lines, typically "phase", "name" or "pid" so several logged phases or processes can be compared on one plot.
...	Additional arguments passed on to autometric::log_read() when log is a file path.

Value

A `ggplot` object with one facet per metric.

Author(s)

Adrien Taudière

See Also

`ga_targets_network()`, `ga_footprint()`

Examples

```
if (requireNamespace("autometric", quietly = TRUE)) {
  log_file <- tempfile()
  autometric::log_start(path = log_file, seconds = 0.05)
  x <- numeric(0)
  for (i in 1:5) {
    x <- c(x, rnorm(2e5))
    Sys.sleep(0.1)
  }
  autometric::log_stop()
  ga_autometric_plot(log_file)
}
```

ga_footprint	<i>Compute footprint in grams of CO2 using R</i> <i>https://doi.org/10.1002/advs.202100707</i> <i>Lannelongue et al. 2021 algorithm</i>
--------------	---

Description

Please cite Lannelongue, L., Grealey, J., Inouye, M., Green Algorithms: Quantifying the Carbon Footprint of Computation. Adv. Sci. 2021, 2100707. [doi:10.1002/advs.202100707](https://doi.org/10.1002/advs.202100707)

Default value are from <https://github.com/Cambridge-Sustainable-Computing-Lab/Green-Algorithms-calculator>:

- PUE: <https://raw.githubusercontent.com/GreenAlgorithms/GA-data/5266caba6601dae0ffc93af8971e758f55292e08/v3>

- TDP_per_core: <https://raw.githubusercontent.com/GreenAlgorithms/GA-data/5266caba6601dae0ffc93af8971e758f552>
- power_draw_per_gb: <https://onlinelibrary.wiley.com/doi/10.1002/advs.202100707>

Description of the algorithm from the [green-algorithms](#) website:

"""

The carbon footprint is calculated by estimating the energy draw of the algorithm and the carbon intensity of producing this energy at a given location:

$$\text{carbonfootprint} = \text{energynEEDED} * \text{carbonintensity}$$

Where the energy needed is:

$$\text{runtime} * (\text{powerdrawforcores} * \text{usage} + \text{powerdrawformemory}) * \text{PUE} * \text{PSF}$$

The power draw for the computing cores depends on the model and number of cores, while the memory power draw only depends on the size of memory available. The usage factor corrects for the real core usage (default is 1, i.e. full usage). The PUE (Power Usage Effectiveness) measures how much extra energy is needed to operate the data centre (cooling, lighting etc.).

The PSF (Pragmatic Scaling Factor) is used to take into account multiple identical runs (e.g. for testing or optimisation).

The Carbon Intensity depends on the location and the technologies used to produce electricity. But note that the "energy needed" [...] is independent of the location.

"""

Usage

```
ga_footprint(
    runtime_h = NULL,
    location_code = "WORLD",
    PUE = 1.67,
    TDP_per_core = 12,
    n_cores = 1,
    cpu_model = "Any",
    memory_ram = NULL,
    power_draw_per_gb = 0.3725,
    PSF = 1,
    usage_core = 1,
    add_ref_values = TRUE,
    add_storage_estimation = FALSE,
    mass_storage = NULL,
    carbon_intensity = NULL,
    TDP_cpu = NULL,
    ref_value = NULL
)
```

Arguments

<code>runtime_h</code>	Runtime in hours (numeric). Use a positive number for explicit runtime, or "session" to automatically calculate based on current R session time using <code>proc.time()</code> .
<code>location_code</code>	Character string specifying geographical location for carbon intensity. Available options include country codes (e.g., "FR", "US", "CN") or "WORLD" for global average. See the Green Algorithms database for complete list of supported locations.
<code>PUE</code>	Power Usage Effectiveness (numeric, default 1.67). Measures data center efficiency - how much extra energy is needed for cooling, lighting, etc. Use 1.05 for personal computers, 1.2-1.7 for data centers. See https://raw.githubusercontent.com/GreenAlgorithms/GA-data/5266caba6601dae0ffc93af8971e758f55292e08/v3.0/default_PUE.csv
<code>TDP_per_core</code>	Thermal Design Power per core in Watts (numeric, default 12). CPU power consumption per core. Find values at https://www.techpowerup.com/cpu-specs/ or https://calculator.green-algorithms.org/ . Overridden by <code>cpu_model</code> parameter.
<code>n_cores</code>	Number of CPU cores (integer, default 1). Overridden by <code>cpu_model</code> parameter.
<code>cpu_model</code>	Character string specifying exact CPU model. Must match entries in the Green Algorithms database. When specified, automatically sets <code>TDP_per_core</code> and <code>n_cores</code> . Use "Any" for generic calculation.
<code>memory_ram</code>	RAM memory in GB (numeric). If NULL, attempts to detect automatically using <code>benchmarkme::get_ram()</code> .
<code>power_draw_per_gb</code>	Power consumption per GB of RAM in Watts (numeric, default 0.3725).
<code>PSF</code>	Pragmatic Scaling Factor (numeric, default 1). Accounts for multiple runs of the same computation. As noted by Lannelongue et al. (2021): "computations are rarely performed only once" - use values > 1 to account for repeated runs, parameter sweeps, or iterative development. GHG emissions are multiplied."
<code>usage_core</code>	(int, default 1). The usage factor corrects for the real core usage (default is 1, i.e. full usage).
<code>add_ref_values</code>	(logical, default TRUE) Do we compute and return reference values to compare to your footprint ?
<code>add_storage_estimation</code>	(logical, default FALSE) Do we compute the footprint of mass storage ? By default FALSE because it is far less important than cpu and memory usage. Note that green-algorithms original tool do not compute mass storage usage.
<code>mass_storage</code>	(int. in GB, default NULL) The size of the mass_storage. Only used if <code>add_storage_estimation</code> is set to TRUE. If set to NULL, use the <code>base::gc()</code> function to estimate storage used.
<code>carbon_intensity</code>	(default NULL). Custom carbon intensity values. Note that only the value for the specified <code>location_code</code> will be used. Accepts three formats: • NULL (default): use the bundled Green Algorithms database (<code>carbon_intensity_internal</code>).

	• A single numeric value: carbon intensity in gCO₂e/kWh, used directly regardless of location_code. Useful when you know your local CI from sources like the Electricity Maps API. • A named numeric vector: names are location codes, values are carbon intensities in gCO₂e/kWh. These override matching entries in the bundled database and add new ones. For example, <code>c("FR" = 56, "CUSTOM_DC" = 450)</code>. • A data.frame with location and carbonIntensity columns: used as-is (original advanced-user behaviour).
TDP_cpu	(default NULL). Advanced users only. A dataframe with model, n_cores and TDP_per_core columns. Set to TDP_cpu_internal if NULL. TDP_cpu_internal is set using command line <code>csv_from_url_ga("https://raw.githubusercontent.com/GreenAlgorithms/GA-data/5266caba6601dae0ffc93af8971e758f55292e08/v3.0/CPU.csv")</code>
ref_value	(default NULL). Advanced users only. A dataframe with variable and value columns. Set to ref_value_internal if NULL. ref_value_internal is set using command line <code>csv_from_url_ga("https://raw.githubusercontent.com/GreenAlgorithms/GA-data/5266caba6601dae0ffc93af8971e758f55292e08/v3.0/referenceValues.csv")</code>

Value

A list of values

- runtime_h: the input run time in hours
- location_code: the input location code
- TDP_per_core: the input TDP_per_core (if cpu_model is set, correspond to the TDP_per_core for this cpu)
- n_cores: the input n_cores (if cpu_model is set, correspond to the n_cores for this cpu)
- cpu_model: the input cpu model. If set to "Any", TDP_per_core and ncore are used
- memory_ram: the input memory ram in GB
- power_draw_per_gb: the input power draw per GB
- usage_core: the input usage core
- carbon_intensity: the input carbon intensity (depend on location code)
- PUE: the input PUE
- PSF: the input PUE
- power_draw_for_cores_kWh: the output power draw for cores in kWh
- power_draw_for_memory_kWh: the output power draw for RAM memory in kWh
- energy_needed_kWh: the output energy needed in kWh
- carbon_footprint_cores: the output carbon footprint in grams of CO₂ for cores usage
- carbon_footprint_memory: the output carbon footprint in grams of CO₂ for memory usage
- carbon_footprint_total_gCO₂: the total output carbon footprint in grams of CO₂
- ref_value: (optionnal, return if add_ref_values is TRUE) : a dataframe
- power_draw_storage_kWh: (optionnal, return if add_storage_estimation is TRUE) the output power draw for mass storage in kWh

Author(s)

Adrien Taudière

Examples

```
# Basic usage with explicit parameters
result <- ga_footprint(
  runtime_h = 2,
  n_cores = 4,
  TDP_per_core = 15,
  memory_ram = 16,
  location_code = "WORLD"
)
result$carbon_footprint_total_gCO2

# Using specific CPU model (automatically sets cores and TDP)
ga_footprint(
  runtime_h = 1,
  cpu_model = "Core i5-9600KF",
  location_code = "FR"
)

# Calculate footprint for current R session
ga_footprint(runtime_h = "session")

# Compare different locations
locations <- c("WORLD", "FR", "US", "NO")
sapply(locations, function(loc) {
  ga_footprint(runtime_h = 1, location_code = loc)$carbon_footprint_total_gCO2
})

# Use a custom carbon intensity value (e.g. from Electricity Maps API)
ga_footprint(runtime_h = 2, carbon_intensity = 42)$carbon_footprint_total_gCO2

# Override specific locations with custom CI values
ga_footprint(
  runtime_h = 2,
  location_code = "FR",
  carbon_intensity = c("FR" = 56, "CUSTOM_DC" = 450)
)$carbon_footprint_total_gCO2

# Advanced usage with storage estimation and reference values
res_ga <- ga_footprint(
  runtime_h = 4,
  n_cores = 8,
  memory_ram = 32,
  add_storage_estimation = TRUE,
  add_ref_values = TRUE
)

library(ggplot2)
ggplot(res_ga$ref_value, aes(y = variable, x = as.numeric(value), fill = log10(prop_footprint))) +
```

```

geom_col() +
geom_col(data = data.frame(
  variable = "Total",
  value = res_ga$carbon_footprint_total_gC02
), fill = "grey30") +
geom_col(data = data.frame(
  variable = "Cores",
  value = res_ga$carbon_footprint_cores
), fill = "darkred") +
geom_col(data = data.frame(
  variable = "Memory",
  value = res_ga$carbon_footprint_memory
), fill = "orange") +
geom_col(data = data.frame(
  variable = "Mass storage",
  value = res_ga$carbon_footprint_storage
), fill = "violet") +
scale_x_continuous(
  trans = "log1p",
  breaks = c(0, 10^c(1:max(log1p(as.numeric(res_ga$ref_value$value))))))
) +
geom_vline(
  xintercept = res_ga$carbon_footprint_total_gC02,
  col = "grey30", lwd = 1.2
) +
geom_label(aes(label = round_conditionally(prop_footprint)),
  fill = "grey90", position = position_stack(vjust = 1.1)
) +
labs(
  title = "Carbon footprint of the analysis",
  subtitle = paste0(
    "(", res_ga$carbon_footprint_total_gC02,
    " g C02", ")"
  ),
  caption = "Please cite Lannelongue et al. 2021 (10.1002/advs.202100707)"
) +
xlab("Carbon footprint (g C02) in log10") +
ylab("Modality") +
theme(legend.position = "none")

```

ga_targets

Calculate carbon footprint for targets pipelines

Description

Calculates the total carbon footprint of a targets pipeline by analyzing the metadata from completed targets. This function is a wrapper around `ga_footprint()` that automatically extracts runtime and storage information from the targets metadata and computes the cumulative environmental impact.

The function aggregates:

- Total runtime across all targets
- Memory usage patterns (when storage estimation is enabled)
- Hardware specifications you provide

Usage

```
ga_targets(
  names_targets = NULL,
  targets_only = TRUE,
  complete_only = FALSE,
  store = targets::tar_config_get("store"),
  tar_meta_raw = NULL,
  ...
)
```

Arguments

names_targets	Character vector of target names to include in analysis. If NULL (default), analyzes all available targets. See <code>?targets::tar_meta()</code>
targets_only	Logical (default TRUE). Whether to analyze only actual targets or also include metadata on functions and other global objects.
complete_only	Logical (default FALSE). Whether to return only targets with complete metadata (no NA values in critical fields).
store	Character string, path to the targets data store. See <code>?targets::tar_meta()</code> for details.
tar_meta_raw	Optional data.frame. If provided, uses this metadata directly instead of calling <code>targets::tar_meta()</code> . Useful for custom analyses or when working with pre-loaded metadata.
...	Additional arguments passed to <code>ga_footprint()</code> , such as: • location_code: geographical location for carbon intensity • n_cores: number of CPU cores used • TDP_per_core: thermal design power per core • memory_ram: RAM memory in GB • PUE: power usage effectiveness

Value

A list with the same structure as `ga_footprint()`. See `?ga_footprint` for complete details on return values.

Author(s)

Adrien Taudière

Examples

```

## Not run:
# Basic usage in a targets project directory
pipeline_footprint <- ga_targets()

# With specific hardware configuration
pipeline_footprint <- ga_targets(
  location_code = "FR",
  n_cores = 4,
  memory_ram = 16,
  PUE = 1.2
)

# Analyze specific targets only
pipeline_footprint <- ga_targets(
  names_targets = c("data_prep", "model_fit", "results"),
  add_storage_estimation = TRUE
)

## End(Not run)

# The next exemple emulate a mini-targets before to ask for tar_meta
targets::tar_dir({ # tar_dir() runs code from a temp dir for CRAN.
  targets::tar_script(
    {
      list(
        targets::tar_target(
          name = waiting,
          command = Sys.sleep(2),
          description = "Sleep 2 seconds"
        ),
        targets::tar_target(x, writelines(
          targets::tar_option_get("error"),
          "error.txt"
        ))
      )
    },
    ask = FALSE
  )

  targets::tar_make()
  tm <- targets::tar_meta()

  res_gat <-
    ga_targets(
      tar_meta_raw = tm,
      n_cores = 6,
      TDP_per_core = 15.8,
      location_code = "FR",
      PUE = 2,
      add_storage_estimation = TRUE
    )

```

```

library(ggplot2)
ggplot(res_gat$ref_value, aes(
  y = reorder(variable, as.numeric(value)),
  x = as.numeric(value), fill = log10(prop_footprint)
)) +
  geom_col() +
  geom_col(data = data.frame(
    variable = "Total ",
    value = res_gat$carbon_footprint_total_gCO2
  ), fill = "grey30") +
  geom_col(
    data = data.frame(
      variable = "Cores",
      value = res_gat$carbon_intensity * res_gat$power_draw_for_cores_kWh
    ),
    fill = "darkred"
  ) +
  geom_col(
    data = data.frame(
      variable = "Memory",
      value = res_gat$carbon_intensity * res_gat$power_draw_for_memory_kWh
    ),
    fill = "orange"
  ) +
  geom_col(
    data = data.frame(
      variable = "Storage",
      value = res_gat$carbon_intensity * res_gat$power_draw_per_gb
    ),
    fill = "violet"
  ) +
  scale_x_continuous(trans = "log1p") +
  geom_vline(
    xintercept = res_gat$carbon_footprint_total_gCO2,
    col = "grey30", lwd = 1.2
  ) +
  geom_label(aes(label = round(prop_footprint, 1)), fill = "grey90") +
  xlab("g CO^2") +
  ylab("Modality")
})

```

ga_targets_meta_plot *Plot per-target runtime and output size from targets metadata*

Description

Draw one horizontal bar per target from `targets::tar_meta()`, with bar length mapped to the target runtime (seconds) and fill mapped to the output size (Gb). Because `tar_meta()` only stores

the metadata of the **latest** build of each target, this figure summarizes the resource impact of the *last* pipeline run. Targets that emitted a warning during that run are flagged with a red star.

This is the metadata-based companion of [ga_autometric_lastrun\(\)](#) (which reconstructs the same "last run" picture from an autometric log) and of [ga_targets_network\(\)](#) (which shows the dependency structure).

Usage

```
ga_targets_meta_plot(
  names_targets = NULL,
  targets_only = TRUE,
  store = targets::tar_config_get("store"),
  tar_meta_raw = NULL,
  transform = "pseudo_log",
  mark_warnings = TRUE
)
```

Arguments

names_targets	(character, default NULL) Optional vector of target names to restrict the plot to. When NULL, every target is shown. Passed to <code>targets::tar_meta()</code> .
targets_only	(logical, default TRUE) Whether to show only actual targets (not functions or other global objects). Passed to <code>targets::tar_meta()</code> .
store	(character, default <code>targets::tar_config_get("store")</code>) Path to the targets data store. See <code>?targets::tar_meta()</code> .
tar_meta_raw	(optional data.frame) A precomputed <code>targets::tar_meta()</code> result. When supplied, <code>store</code> , <code>names_targets</code> and <code>targets_only</code> are ignored. Useful for tests or pre-loaded metadata.
transform	(character, default "pseudo_log") Transformation applied to the x (runtime) axis via <code>ggplot2::scale_x_continuous()</code> . Runtimes often span several orders of magnitude, so a compressing transform such as "pseudo_log" or "log1p" keeps short targets visible. Use "identity" for a linear axis.
mark_warnings	(logical, default TRUE) Whether to overlay a red star on targets that recorded a warning in their metadata.

Value

A [ggplot](#) object with one bar per target.

Author(s)

Adrien Taudière

See Also

[ga_autometric_lastrun\(\)](#), [ga_autometric_history\(\)](#), [ga_targets_network\(\)](#)

Examples

```
targets::tar_dir({ # tar_dir() runs code from a temp dir for CRAN.
  targets::tar_script(
    {
      list(
        targets::tar_target(raw, rnorm(1e5)),
        targets::tar_target(scaled, raw * 2),
        targets::tar_target(summ, mean(scaled))
      )
    },
    ask = FALSE
  )
  targets::tar_make(reporter = "silent")
  ga_targets_meta_plot(tar_meta_raw = targets::tar_meta(targets_only = TRUE))
})
```

ga_targets_network	<i>Visualize a targets pipeline as a resource-annotated network</i>
--------------------	---

Description

Draw the dependency graph of a targets pipeline with ggnetwork, encoding the per-target computational cost as visual aesthetics: node **size** and **color** map to a resource metric (runtime in seconds or output size in bytes) and each edge **width** maps to the runtime of its downstream target. This gives a quick, at-a-glance picture of where a pipeline spends its time and storage, complementing the aggregated figures of [ga_targets\(\)](#).

The per-target metrics (seconds, bytes) are taken directly from `targets::tar_network()`, so no separate `tar_meta()` call is needed.

Usage

```
ga_targets_network(
  store = targets::tar_config_get("store"),
  script = targets::tar_config_get("script"),
  tar_network_raw = NULL,
  size_by = "seconds",
  color_by = "bytes",
  layout = "fruchtermanreingold",
  label = TRUE,
  arrow_gap = 0.02,
  ...
)
```

Arguments

store	(character, default <code>targets::tar_config_get("store")</code>) Path to the targets data store. See <code>?targets::tar_network()</code> .
-------	--

script	(character, default <code>targets::tar_config_get("script")</code>) Path to the target script file (e.g. <code>_targets.R</code>). Required by <code>targets::tar_network()</code> to build the pipeline graph. See <code>?targets::tar_network()</code> .
tar_network_raw	(optional list) A precomputed <code>targets::tar_network()</code> result (a list with <code>\$vertices</code> and <code>\$edges</code>). When supplied, <code>store</code> is ignored. Useful for tests, custom analyses, or when the pipeline metadata is already loaded.
size_by	(character, default "seconds") Vertex attribute mapped to node size. One of "seconds" (runtime) or "bytes" (output size).
color_by	(character, default "bytes") Vertex attribute mapped to node color. One of "seconds", "bytes" or "status" (the target build status).
layout	(character, default "fruchtermanreingold") A network layout name passed to <code>ggnetwork::ggnetwork()</code> .
label	(logical, default TRUE) Draw the target names next to the nodes.
arrow_gap	(numeric, default 0.02) Gap left between the arrow head and the target node, passed to <code>ggnetwork::ggnetwork()</code> .
...	Additional arguments passed on to <code>ggnetwork::ggnetwork()</code> .

Value

A [ggplot](#) object showing the pipeline network.

Author(s)

Adrien Taudière

See Also

[ga_targets\(\)](#), [ga_autometric_plot\(\)](#)

Examples

```
targets::tar_dir({ # tar_dir() runs code from a temp dir for CRAN.
  targets::tar_script(
    {
      list(
        targets::tar_target(raw, rnorm(1e5)),
        targets::tar_target(scaled, raw * 2),
        targets::tar_target(summ, mean(scaled))
      )
    },
    ask = FALSE
  )
  targets::tar_make(reporter = "silent")
  net <- targets::tar_network(targets_only = TRUE)
  ga_targets_network(tar_network_raw = net)
})
```

round_conditionally	<i>Conditionally round numeric values based on magnitude</i>
---------------------	--

Description

Applies different rounding rules based on the magnitude of values. Larger values are rounded to fewer decimal places, while smaller values retain more precision. This is useful for presenting results with appropriate precision across different scales.

Usage

```
round_conditionally(
  vec,
  cond = cbind(c(1e-05, 5), c(0.001, 3), c(0.01, 3), c(1, 2), c(10, 1), c(100, 0))
)
```

Arguments

vec	A numeric vector to be rounded
cond	A matrix with 2 rows and n columns where: • First row: threshold values for applying rounding rules • Second row: number of decimal places to round to The function automatically sorts conditions in decreasing order of thresholds. Default provides reasonable rounding for most carbon footprint values.

Value

A numeric vector of the same length as vec with values rounded according to the conditional rules

Author(s)

Adrien Taudière

Examples

```
# Default rounding behavior
values <- c(1000.27890, 10.87988, 1.769869, 0.99796, 0.000179)
round_conditionally(values)

# Custom rounding rules
custom_rules <- cbind(c(10e-5, 5), c(10, 2)) # 5 decimals for tiny values, 2 for others
round_conditionally(c(1000.27890, 0.000179, 10e-11), cond = custom_rules)

# Useful for carbon footprint reporting
footprint_values <- c(0.001234, 1.23456, 123.456, 12345.6)
round_conditionally(footprint_values)
```

session_runtime	<i>Compute session runtime and memory usage statistics</i>
-----------------	--

Description

Analyzes the current R session to extract timing and memory usage information. This function is particularly useful for understanding resource consumption patterns and can be used with `ga_footprint(runtime_h = "session")`.

The function uses `base::proc.time()` to get CPU timing information and `base::gc()` to estimate memory usage when requested.

Usage

```
session_runtime(compute_mass_storage = TRUE)
```

Arguments

`compute_mass_storage`

Logical (default TRUE). Whether to compute memory usage statistics using the `base::gc()` function. Set to FALSE if you only need timing information.

Value

A list containing:

- `cpu_times_users`: User CPU time in seconds
- `cpu_times_system`: System CPU time in seconds
- `time_elapsed`: Total elapsed time in seconds
- `cpu_times`: Combined user and system CPU time
- `mass_storage_used`: Memory currently used (if requested)
- `mass_storage_max`: Maximum memory used (if requested)

Author(s)

Adrien Taudière

Examples

```
# Get complete session information
session_info <- session_runtime()
print(session_info)

# Get only timing information (faster)
timing_only <- session_runtime(compute_mass_storage = FALSE)
cat("Session has been running for", timing_only$time_elapsed, "seconds\n")
```

Index

`csv_from_url_ga`, [3](#)

`ga_autometric_history`, [4](#), [6](#), [7](#), [17](#)

`ga_autometric_lastrun`, [4](#), [5](#), [6](#), [17](#)

`ga_autometric_plot`, [5](#), [7](#), [7](#), [19](#)

`ga_footprint`, [2](#), [7](#), [8](#), [8](#)

`ga_targets`, [2](#), [13](#), [18](#), [19](#)

`ga_targets_meta_plot`, [4–7](#), [16](#)

`ga_targets_network`, [8](#), [17](#), [18](#)

`ggplot`, [5–8](#), [17](#), [19](#)

`greenAlgoR-package`, [2](#)

`round_conditionally`, [20](#)

`session_runtime`, [2](#), [21](#)