Package: greenAlgoR (via r-universe)

November 21, 2024

Title Compute ecological footprint in R

Version 0.1.1

Description This package computes ecological footprint in R (based on [green-algorithms](https://calculator.green-algorithms.org/). greenAlgoR also made it simple to compute ecological footprint of \{[targets](https://github.com/ropensci/targets)\} pipelines.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

URL `https://github.com/adrientaudiere/greenAlgoR`, `https://adrientaudiere.github.io/greenAlgoR/`

BugReports `https://github.com/adrientaudiere/greenAlgoR/issues`

Depends R (>= 3.5.0), benchmarkme, targets, ggplot2

Suggests here, RCurl, testthat (>= 3.0.0)

Config/testthat/edition 3

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

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

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

RemoteRef HEAD

RemoteSha f9345c2eb2d85fe2cc1b5e136da5a4c4bf346238

Contents

greenAlgoR-package 2
csv_from_url_ga 2
ga_footprint 3
ga_targets 7
round_conditionaly 9
session_runtime 10

Index 12

greenAlgoR-package greenAlgoR *package*

Description

This package computes ecological footprint in R (based on green-algorithms. greenAlgoR also made it simple to compute ecological footprint of targets pipelines..

csv_from_url_ga *Load csv files from green algo github repositories*

Description

Mainly for internal use

Usage

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

Arguments

`url`	url to a raw csv file
`remove_first_line`	(logical, default TRUE): Do we remove the first line from the csv file.

Value

a data.frame

Author(s)

Adrien Taudière

Examples

```
carbon_intensity_internal <-
  csv_from_url_ga("https://raw.githubusercontent.com/GreenAlgorithms/green-algorithms-tool/refs/heads/master/da
```

`ga_footprint` *Compute footprint in grams of CO2 using Rhrefhttps://doi.org/10.1002/advs.202100707Lannelongue et al. 2021 algorithm*

Description

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

Default value are from https://github.com/GreenAlgorithms/green-algorithms-tool:

- PUE: https://github.com/GreenAlgorithms/green-algorithms-tool/blob/master/data/v2.2/defaults_PUE.csv
- TDP_per_core: https://raw.githubusercontent.com/GreenAlgorithms/green-algorithms-tool/refs/heads/master/data/v2.2
- 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:

$$carbonfootprint = energyneeded * carbonintensity$$

Where the energy needed is:

$$runtime * (powerdrawforcores * usage + powerdrawformemory) * PUE * PSF$$

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

`runtime_h` Run time in hours (int). If runtime_h == "session", the runtime is compute using the actual R session

`location_code` (character list of country or region available in)

`PUE` (int) Power usage effectiveness of the server. See https://github.com/GreenAlgorithms/green-algorithms-tool/blob/master/data/v2.2/defaults_PUE.csv for example of values. If you are using your personal computer, set PUE to 1.

`TDP_per_core` (int. in Watt, default 12). Find your cpu TDP and your nb of cpu on https://www.techpowerup.com/cpu-specs/ or in http://calculator.green-algorithms.org/ if available. Owerwrite by cpu_model param.

`n_cores` (int, default 1) Number of cores. Owerwrite by cpu_model param.

`cpu_model` Must be present in the list of http://calculator.green-algorithms.org/. If CPU is set, the parameter TPD_per_core and n_cores are overwriting by info from the cpu_model. "auto" modality (find the cpu using `benchmarkme::get_cpu()$model_name`) is not running for the moment.

`memory_ram` (int. in GB) The memory RAM. If memory_ram is NULL, use benchmarkme::get_ram() to get the RAM.

`power_draw_per_gb` (int. in Watt, default 0.3725) The power draw for each GB of RAM

`PSF` (int, default 1) Pragmatic Scaling Factor. Citation from Lannelongue et al. 2021: "Many analyses are presented as a single run of a particular algorithm or software tool; however, computations are rarely performed only once. Algorithms are run multiple times, sometimes hundreds, systematically or manually, with different parameterizations. Statistical models may include any number of combinations of covariates, fitting procedures, etc. It is important to include these repeats in the carbon footprint. To take into account the number of times a computation is performed in practice, the PSF was defined, a scaling factor by which the estimated GHG emissions are multiplied."

`usage_core` (int, default 1). The usage factor corrects for the real core usage (default is 1, i.e. full usage).

`add_ref_values` (logical, default TRUE) Do we compute and return reference values to compare to your footprint ?

`add_storage_estimation`
(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 website do not compute mass storage usage.

`mass_storage` (int. in GB, default NULL) The size of the mass_storage. Only used if add_storage_estimation is set to TRUE. If set to NULL, use the `base::gc()` function to estimate storage used.

`carbon_intensity`
(default NULL). Advanced users only. A dataframe with `location` and `carbonIntensity` columns. Set to carbon_intensity_internal if NULL. carbon_intensity_internal is set using command line csv_from_url_ga("https://raw.githubusercontent.com/GreenAlgorithms/green-algorithms-tool/refs/heads/master/data/v2.2/CI_aggregated.csv")

`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 csv_from_url_ga("https://raw.githubusercontent.com/GreenAlgorithms/green-algorithms-tool/refs/heads/master/data/v2.2/TDP_cpu.csv")

`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 csv_from_url_ga("https://raw.githubusercontent.com/GreenAlgorithms/green-algorithms-tool/refs/heads/master/data/v2.2/referenceValues.csv")

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

```
ga_footprint(
  runtime_h = 12,
  n_cores = 6,
  TDP_per_core = 15.8,
  location_code = "FR",
  PUE = 1,
  cpu_model = "Core i5-9600KF"
)

ga_footprint(
  runtime_h = "session",
  PUE = 1,
)

res_ga <- ga_footprint(
  runtime_h = 12,
  n_cores = 6,
  memory_ram = 64,
  PUE = 1,
  add_storage_estimation = TRUE,
  mass_storage = 1
)

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_gCO2
  ), 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_gCO2,
    col = "grey30", lwd = 1.2
  ) +
  geom_label(aes(label = round_conditionaly(prop_footprint)),
    fill = "grey90", position = position_stack(vjust = 1.1)
  ) +
  labs(
    title = "Carbon footprint of the analysis",
    subtitle = paste0(
      "(", res_ga$carbon_footprint_total_gCO2,
      " g CO2", ")"
    ),
    caption = "Please cite Lannelongue et al. 2021 (10.1002/advs.202100707)"
  ) +
  xlab("Carbon footprint (g CO2) in log10") +
  ylab("Modality") +
  theme(legend.position = "none")
```

ga_targets   *Compute footprint in grams of CO2 for {targets} pipelines*

Description

It is mainly a wrapper of function `ga_footprint()` that compute run time and mass_storage (only used if add_storage_estimation = TRUE) using `targets::tar_meta()`.

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` Optional, names of the targets. See ?targets::tar_meta()

`targets_only` Logical, whether to just show information about targets or also return metadata on functions and other global objects.

`complete_only` Logical, whether to return only complete rows (no NA values).

`store` Character of length 1, path to the targets data store. See ?targets::tar_meta()

`tar_meta_raw` Optional, if not NULL, other listed options above (params for `targets::tar_meta()` are not used.

`...` Other args to be passed on `ga_footprint()`

Value

A list of value. See ?ga_footprint for the details.

Author(s)

Adrien Taudière

Examples

```
# In a targets folder, just run function ga_targets()
# with the options you want

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

  tar_make()
  tm <- 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
    )
  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")
})
```

round_conditionaly *Round numeric vector conditionaly*

Description

Round numeric vector conditionaly

Usage

```
round_conditionaly(
  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
cond	: a matrix of 2 row an n column with the first row defining the condition and the second row defining the number to round. cond is order in decreasing order of the 1 row internally. Thus the order in cond rows is not important

Value

a numeric vector of the same length as vec

Author(s)

Adrien Taudière

Examples

```
round_conditionaly(vec = c(1000.27890, 10.87988, 1.769869, 0.99796, 0.000179))
round_conditionaly(
  vec = c(1000.27890, 0.000179, 10e-11),
  cond = cbind(c(10e-5, 5), c(10, 2))
)
```

session_runtime *Compute session run time and mass storage use (based on `gc()`)*

Description

Compute cpu times using `base::proc.time()` and mass storage using `base::gc()`

Usage

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

Arguments

compute_mass_storage	(logical, default TRUE) Do the mass storage is computed from `base::gc()` function

Value

A list of values

Author(s)

Adrien Taudière

Examples

```
session_runtime()
session_runtime(compute_mass_storage = FALSE)
```

Index

csv_from_url_ga, 2

ga_footprint, 3
ga_targets, 7
greenAlgoR-package, 2

round_conditionaly, 9

session_runtime, 10