Package 'irace'

October 18, 2016

Type Package

Title Iterated Racing for Automatic Algorithm Configuration

Description Iterated race is an extension of the Iterated F-race method for the automatic configuration of optimization algorithms, that is, (offline) tuning their parameters by finding the most appropriate settings given a set of instances of an optimization problem.

Version 2.1

Date 2016-10-18

Maintainer Manuel López-Ibáñez <manuel.lopez-ibanez@manchester.ac.uk>

Author Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, Mauro Birattari, Eric Yuan and Prasanna Balaprakash

Depends R (>= 2.15.0)

Imports stats, utils

Suggests Rmpi (>= 0.6.0), parallel, knitr

VignetteBuilder knitr

License GPL (>= 2)

URL http://iridia.ulb.ac.be/irace

ByteCompile yes

LazyData yes

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2016-10-18 22:52:28

R topics documented:

`irace-package` *The irace package*

Description

Iterated racing for Automatic Algorithm Configuration

Details

Package:	irace
Type:	Package
Version:	2.1
Date:	2016-10-18
License:	GPL (>= 2)
LazyLoad:	yes

Author(s)

Maintainer: Manuel López-Ibáñez and Jérémie Dubois-Lacoste <irace@iridia.ulb.ac.be>

Author: Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle, Mauro Birattari, Eric Yuan and Prasanna Balaprakash

References

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Thomas Stützle, and Mauro Birattari. *The irace package, Iterated Race for Automatic Algorithm Configuration*. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

Manuel López-Ibáñez and Thomas Stützle. The Automatic Design of Multi-Objective Ant Colony Optimization Algorithms. IEEE Transactions on Evolutionary Computation, 2012.

Examples

```
##############################################################################
# This example illustrates how to tune the parameters of the simulated
# annealing algorithm (SANN) provided by the optim() function in the
# R base package.  The goal in this example is to optimize instances of
# the following family:
# f(x) = lambda * f_rastrigin(x) + (1 - lambda) * f_rosenbrock(x)
# where lambda follows a normal distribution whose mean is 0.9 and
# standard deviation is 0.02. f_rastrigin and f_rosenbrock are the
# well-known Rastrigin and Rosenbrock benchmark functions (taken from
# the cmaes package). In this scenario, different instances are given
# by different values of lambda.
##############################################################################
## First we provide an implementation of the functions to be optimized:
f_rosenbrock <- function (x) {
  d <- length(x)
  z <- x + 1
  hz <- z[1:(d - 1)]
  tz <- z[2:d]
  s <- sum(100 * (hz^2 - tz)^2 + (hz - 1)^2)
  return(s)
}
f_rastrigin <- function (x) {
  sum(x * x - 10 * cos(2 * pi * x) + 10)
}

## We generate 200 instances (in this case, weights):
weights <- rnorm(200, mean = 0.9, sd = 0.02)

## On this set of instances, we are interested in optimizing two
## parameters of the SANN algorithm: tmax and temp. We setup the
## parameter space as follows:
parameters.table <- '
tmax "" i (1, 5000)
temp "" r (0, 100)
'
```

```
## We use the irace function readParameters to read this table:
parameters <- readParameters(text = parameters.table)

## Next, we define the function that will evaluate each candidate
## configuration on a single instance. For simplicity, we restrict to
## three-dimensional functions and we set the maximum number of
## iterations of SANN to 5000.
target.runner <- function(experiment, scenario)
{
  instance <- experiment$instance
  configuration <- experiment$configuration

  D <- 3
  par <- runif(D, min=-1, max=1)
  fn <- function(x) {
    weight <- instance
    return(weight * f_rastrigin(x) + (1 - weight) * f_rosenbrock(x))
  }
  res <- optim(par,fn, method="SANN",
               control=list(maxit=5000
                 , tmax = as.numeric(configuration[["tmax"]])
                 , temp = as.numeric(configuration[["temp"]])
                 ))
  return(res$value)
}

## Not run:
## We are now ready to launch irace. We do it by means of the irace
## function by setting targetRunner to the function define above, instances to
## the first 100 random weights, and a maximum budget of 1000 calls to
## targetRunner. The function irace will print information about its
## progress. This may require a few minutes, so it is not run by default.
result <- irace(scenario = list(
                  targetRunner = target.runner,
                  instances = weights[1:100],
                  maxExperiments = 1000,
                  logFile = ""),
                parameters = parameters)

## We can print the best configurations found by irace as follows:
configurations.print(result)

## We can evaluate the quality of the best configuration found by
## irace versus the default configuration of the SANN algorithm on
## the other 100 instances previously generated.
## To do so, first we apply the default configuration of the SANN
## algorithm to these instances:
default <- sapply(weights[101:200], target.runner,
                  configuration=list(values=list(tmax=10,temp=10)))

## We extract and apply the winning configuration found by irace
## to these instances:
result.list <- as.list(removeConfigurationsMetaData(result[1,]))
```

```
tuned <- sapply(weights[101:200], target.runner, configuration=list(values=result.list))

## Finally, we can compare using a boxplot the quality obtained with the
## default parametrization of SANN and the quality obtained with the
## best configuration found by irace.
boxplot(list(default=default, tuned=tuned))

## End(Not run)
```

`buildCommandLine` *Build a Command Line*

Description

'buildCommandLine' receives two vectors, one containing the values of the parameters, the other containing the switches of the parameters. It builds a string with the switches and the values that can be used as a command line to call the program to be tuned, instanciating one candidate configuration.

Usage

```
buildCommandLine(values, switches)
```

Arguments

`values`	A vector containing the value of each parameter for the candidate configuration.
`switches`	A vector containing the switches of each paramter (in an order that corresponds to the values vector).

Details

The chain concatenates <switch> <value> for all parameters with a space between each parameter (but none between the switches and the corresponding values).

Value

A character chain containing the switches and the values.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

Examples

```
switches <- c("--switch1 ", "--switch2 ")
values <- c("value_1", "value_2")
buildCommandLine (values, switches)

## Not run:
## Build commandlines from the results produced by a previous run of
## irace.

# First, load the data produced by irace.
load("irace.Rdata")
attach(iraceResults)
apply(allConfigurations[1:10, unlist(parameters$names)], 1, buildCommandLine,
      unlist(parameters$switches))

## End(Not run)
```

`checkIraceScenario` *checkIraceScenario*

Description

'checkIraceScenario' check the given scenario options.

Usage

```
checkIraceScenario(scenario, parameters=NULL)
```

Arguments

`scenario`	Data structure containing **irace** settings.The data structure has to be the one returned by the function `defaultScenario()` and `readScenario()`. See documentation of this function for details.
`parameters`	Data structure containing the parameter definition. The data structure has to be the one returned by the function `readParameters()`. See documentation of this function for details.

Details

Provide the `parameters` argument only if the parameter list should not be obtained from a parameter file. If the parameter list is provided it will be not checked.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

readScenario for reading a configuration scenario from a file. defaultScenario returns the default scenario settings of **irace**. checkScenario to check that the scenario is valid.

---

checkScenario *checkScenario*

Description

checkScenario takes a (possibly incomplete) scenario setup of **irace**, checks for errors and transforms it into a valid scenario.

Usage

```
checkScenario(scenario = defaultScenario())
```

Arguments

scenario	A list where tagged elements correspond to scenario settings of **irace**.

Details

This function checks that the directories and the file names provided and required by the **irace** exist. It also checks that the settings are of the proper type, e.g. that settings expected to be integers are really integers. Finally, it also checks that there is no inconsistency between settings. If an error is found that prevents **irace** from running properly, it will stop with an error.

Value

The scenario received as a parameter, possibly corrected. Unset scenario settings are set to their default values.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

readScenario for reading a configuration scenario from a file. printScenario prints the given scenario. defaultScenario to get the default scenario settings.

configurations.print *configurations.print*

Description

Print configuration configurations (hereafter "configurations").

Usage

```
configurations.print(configuration, metadata = FALSE)
```

Arguments

configuration	A matrix containing the configurations (one per row).
metadata	A Boolean specifying whether to print the metadata or not. The metadata are data for the configurations (additionally to the value of each parameter) used by **irace**.

Value

None.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

configurations.print.command to print the configurations as command lines.

configurations.print.command *configurations.print.command*

Description

Print configuration configurations (hereafter "configurations") as command lines.

Usage

```
configurations.print.command(configuration, parameters)
```

Arguments

configuration	A matrix containing the configurations (one per row).
parameters	A data structure similar to that provided by the 'readParameters' function.

Value

None.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`configurations.print` to print the configurations as a data frame.

`defaultScenario` *defaultScenario*

Description

'defaultScenario' returns the default scenario settings of **irace**.

Usage

```
defaultScenario(scenario = list())
```

Arguments

`scenario`	A list where tagged elements correspond to scenario settings of **irace**.

Value

A list indexed by the **irace** parameter names, containing the default values for each parameter, except for those already present in the scenario passed as argument. The scenario list contains the following elements:

General:

- scenarioFile: Path of the file that describes the configuration scenario setup and other irace settings. (Default: "./scenario.txt")
- debugLevel: Debug level of the output of `irace`. Set this to 0 to silence all debug messages. Higher values provide more verbose debug messages. (Default: 0)
- seed: Seed of the random number generator (must be a positive integer, NA means use a random seed). (Default: `NA`)
- execDir: Directory where the experiments will be run. (Default: "./")
- logFile: Path of the file to save tuning results as an R dataset, either absolute path or relative to execDir. (Default: "./irace.Rdata")

Elitist irace:

- elitist: Enable/disable elitist irace. (Default: 1)

- elitistNewInstances: Number of instances added to the execution list before previous instances in elitist irace. (Default: 1)
- elitistLimit: Limit for the elitist race, number statistical test without elimination peformed. Use 0 for no limit. (Default: 2)

`irace` internal: For most of these parameters is adviced to use the default settings.

- sampleInstances: Enable/disable the sampling of instances. (Default: 1)
- nbIterations: Number of iterations to be performed by `irace`.
- nbExperimentsPerIteration: Number of experiments (runs) per iteration.
- nbConfigurations: Number of configurations to be sampled and evaluated at each iteration.
- mu: Parameter used to define the number of configurations sampled and evaluated at each iteration. (Default: 5)
- minNbSurvival: Minimum number of configurations needed to continue executing a race.
- softRestart: Enable/disable the soft restart strategy that avoids premature convergence of the probabilistic model. (Default: 1)
- softRestartThreshold: Soft restart threshold value for numerical parameters. If `NA`, it computed as `10^-digits`. (Default: `NA`)

Target algorithm parameters:

- parameterFile: Path to the file that contains the description of the parameters to be tuned. See the template. (Default: "./parameters.txt")
- digits: Indicates the number of decimal places to be considered for the real parameters. (Default: 4)
- forbiddenExps: MANUEL
- forbiddenFile: Path to a file that contains a list of logical expressions that cannot be `TRUE` for any evaluated configuration. If empty or `NULL`, do not use forbidden expressions. (Default: "")

Target algorithm execution:

- targetRunner: Path to the script called for each configuration that launches the algorithm to be tuned. See templates. (Default: "./target-runner")
- targetRunnerRetries: Number of times to retry a call to `targetRunner` if the call failed. (Default: 0)
- targetRunnerData: Optional data passed to `targetRunner`. This is ignored by the default `targetRunner` function, but it may be used by custom `targetRunner` functions to pass persistent data around.
- targetRunnerParallel: Optional R function to provide custom parallelization of `targetRunner`.
- targetEvaluator: Path to the script that provides a numeric value for each configuration. See templates. (Default: "")
- deterministic: If the target algorithm is deterministic, configurations will be evaluated only once per instance. (Default: 0)
- parallel: Number of calls to `targetRunner` to execute in parallel. 0 or 1 mean disabled. (Default: 0)

- loadBalancing: Enable/disable load-balancing when executing experiments in parallel. Load-balancing makes better use of computing resources, but increases communication overhead. If this overhead is large, disabling load-balancing may be faster. (Default: 1)
- mpi: Enable/disable MPI. Use Rmpi to execute targetRunner in parallel (parameter parallel is the number of slaves). (Default 0)
- sgeCluster: Enable/disable SGE cluster mode. Use qstat to wait for cluster jobs to finish (targetRunner must invoke qsub). (Default: 0)

Initial configurations:

- configurationsFile: Path to a file that contains a set of initial configurations. If empty or `NULL` do not use a initial configurations. (Default: "")

Training instances:

- instances: Array of the instances to be used in the `targetRunner`.
- instancesList: Data frame that constains the instance and seeds used in the tuning. The order of this data frame indicate the order in which the instances have been used.
  - instance: The instance index (corresponding to the `instance` array).
  - seed: Seed to be used with the corresponding instance.
- instances.extra.params: Array that contains the extra parameters defined per instance, the name of each element corresponds to the instance name in the `instances` array. When no extra parameters are provided this element is `NULL`.
- trainInstancesDir: Directory where tuning instances are located; either absolute path or relative to current directory.. If no `trainInstancesFiles` is provided all the files in `trainInstancesDir` will be listed as instances. (Default: "./Instances")
- trainInstancesFile: Path to a file that contains a list of instances and optionally additional parameters for them. If `trainInstancesDir` is provided `irace` will search for the files in this folder. (Default: "")

Tuning budget:

- maxExperiments: Maximum number of runs (invocations of `targetRunner`) that will be performed. It determines the maximum budget of experiments for the tuning. (Default: 0)
- maxTime: Maximum total execution time for the executions of `targetRunner`. targetRunner must return two values: quality time. (Default: 0)
- budgetEstimation: Percentage of the time budget used to estimate the mean computation time of a configuration. Only used when maxTime is provided. (Default: 0.02)

Statistical test:

- testType: Statistical test used for elimination. (Default: "F-test")
- confidence: Confidence level of the statistical test. (Default: 0.95)
- firstTest: Number of instances evaluated before the first elimination test. (Default: 5)
- eachTest: Number of instances evaluated between elimination tests. (Default: 1)

Recovery:

- recoveryFile: Path to an irace log file used to recover the `irace` execution. (Default: "")

Testing:

- testNbElites: Number of elite configurations returned by irace to be tested if test instances are provided. (Default: 1)
- testIterationElites: Enable/disable testing the elite configurations found at each iteration. (Default: 0)
- testInstancesDir: Directory where testing instances are located, either absolute or relative to current directory. (Default: "")
- testInstancesFile: Path to a file containing a list of test instances and optionally additional parameters for them. (Default: "")
- testInstances: Array of the instances to be used in the `targetRunner` when executing the testing.
- testInstances.extra.params: Array that contains the extra parameters defined per instance, the name of each element corresponds to the instance name in the `testInstances` array. When no extra parameters are provided this element is `NULL`.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`readScenario` for reading a configuration scenario from a file. `checkScenario` to check that the scenario is valid. `printScenario` prints the given scenario.

`getConfigurationById` *getConfigurationById*

Description

'getConfigurationById' returns the configurations selected by id.

Usage

```
getConfigurationById(iraceResults=NULL, irace.logFile=NULL, ids, drop.internals=FALSE)
```

Arguments

`iraceResults`	Object created by **irace** and saved in `scenario$logFile`.
`irace.logFile`	Log file created by **irace**, this file must contain the `iraceResults` object.
`ids`	The id or a vector of ids of the candidates configurations to obtain.
`drop.internals`	Remove the internal identifier and parent identifier from the returned configurations data frame.

Value

A data frame containing the elite configurations required.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`defaultScenario` returns the default scenario settings of

`getConfigurationByIteration`
getConfigurationByIteration

Description

'getConfigurationByIteration' returns the configurations by the iteration in which they were executed.

Usage

```
getConfigurationByIteration(iraceResults = NULL, irace.logFile = NULL,
                            iterations, drop.internals = FALSE)
```

Arguments

`iraceResults`	Object created by **irace** and saved in `scenario$logFile`.
`irace.logFile`	Log file created by **irace**, this file must contain the `iraceResults` object.
`iterations`	The iteration number or a vector of iterations numbers from where the configurations should be obtained.
`drop.internals`	Remove the internal identifier and parent identifier from the returned configurations data frame.

Value

A data frame containing the elite configurations required.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`defaultScenario` returns the default scenario settings of

getFinalElites *getFinalElites*

Description

'getFinalElites' returns the elite configurations of the final iteration.

Usage

```
getFinalElites(iraceResults=NULL, irace.logFile=NULL, n=0, drop.internals=FALSE)
```

Arguments

`iraceResults`	Object created by **irace** and saved in `scenario$logFile`.
`irace.logFile`	Log file created by **irace**, this file must contain the `iraceResults` object.
`n`	Number of elite configurations to return, if `n` is larger than the number of configurations, then only the existing ones are returned.
`drop.internals`	Remove the internal identifier and parent identifier from the returned configurations data frame.

Value

A data frame containing the elite configurations required.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`defaultScenario` returns the default scenario settings of

irace *irace*

Description

`irace` implements iterated Race. It receives some parameters to be tuned and returns the best configurations found, namely, the elite configurations obtained from the last iterations (and sorted by rank).

Usage

```
irace(scenario, parameters)
```

Arguments

`scenario`	Data structure containing **irace** settings.The data structure has to be the one returned by the function `defaultScenario()` and `readScenario()`. See documentation of this function for details.
`parameters`	Data structure containing the parameter definition. The data structure has to be the one returned by the function `readParameters()`. See documentation of this function for details.

Details

The function `irace` executes the tuning procedure using the information provided in `scenario` and `parameters`. Initially it checks the correctness of `scenario` and recovers a previous execution if `scenario$recoveryFile` is set. A R data file log of the execution is created in `scenario$logFile`.

Value

A data frame with the set of best algorithm configurations found by **irace**. The data frame has the following columns:

- .ID.:Internal id of the candidate configuration.
- Parameter names: One column per parameter name in `parameters`.
- .PARENT.: Internal id of the parent candidate configuration.

Additinally, this function saves an R data file containing an object called `iraceResults`. The path of the file is indicated in `scenario$logFile`. The `iraceResults` object is a list with the following structure:

- scenario: The scenario R object containing the **irace** options used for the execution. See defaultScenario help for more information.
- parameters: The parameters R object containing the description of the target algorithm parameters. See readParameters.
- allConfigurations: The target algorithm configurations generated by **irace**. This object is a data frame, each row is a candidate configuration, the first column (`.ID.`) indicates the internal identifier of the configuration, the following columns correspond to the parameter values, each column named as the parameter name specified in the parameter object. The final column (`.PARENT.`) is the identifier of the configuration from which model the actual configuration was sampled.
- allElites: A list that contains one element per iteration, each element contains the internal identifier of the elite candidate configurations of the corresponding iteration (identifiers correspond to `allConfigurations$.ID.`).
- iterationElites: A vector containing the best candidate configuration internal identifier of each iteration. The best configuration found corresponds to the last one of this vector.
- experiments: A matrix with configurations as columns and instances as rows. Column names correspond to the internal identifier of the configuration (`allConfigurations$.ID.`).
- experimentLog: A matrix with columns `iteration, instance, configuration, time`. This matrix contains the log of all the experiments that **irace** performs during its execution. The instance column refers to the index of the `scenario$instancesList` data frame. Time is saved ONLY when reported by the targetRunner.

- softRestart: A logical vector that indicates if a soft restart was performed on each iteration. If FALSE, then no soft restart was performed.
- state: A list that contains the state of **irace**, the recovery is done using the information contained in this object.
- testing: A list that constains the testing results. The elements of this list are: experiments a matrix with the testing expriments of the selected configurations in the same format as the explained above and seeds a vector with the seeds used to execute each experiment.

Author(s)

Manuel Lopez-Ibañez and Jérémie Dubois-Lacoste

See Also

irace.main a higher-level command-line interface to irace.

readScenario to read the scenario setup from a file.

defaultScenario to provide a default scenario for **irace**.

readParameters to read the target algorithm parameters from a file.

Examples

```
## Not run:
parameters <- readParameters("parameters.txt")
scenario <- readScenario(filename="scenario.txt",
                         scenario=defaultScenario())
irace(scenario=scenario, parameters=parameters)

## End(Not run)
```

---

irace.cmdline *irace.cmdline*

Description

'irace.cmdline' starts **irace** using the parameters of the command line used to invoke R.

Usage

```
irace.cmdline(args = commandArgs(trailingOnly = TRUE))
```

Arguments

args	The arguments provided on the R command line as a character vector, e.g., c("--scenario", "scenario.txt", "-p", "parameters.txt"). Using the default value (not providing the parameter) is the easiest way to call irace.cmdline.

Details

The function reads the parameters given on the command line used to invoke R, finds the name of the scenario file, initializes the scenario from the file (with the function 'readScenario) and possibly from parameters passed on the command line. It finally starts **irace** by calling 'irace.main'.

Value

None.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`irace.main` to start **irace** with a given scenario.

`irace.license` *irace.license*

Description

A character string containing the license information of **irace**.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

`irace.main` *irace.main*

Description

`irace.main` is a higher-level interface to invoke `irace`.

Usage

```
irace.main(scenario = defaultScenario(), output.width = 9999)
```

Arguments

`scenario`	The scenario setup of **irace**.
`output.width`	The width that must be used for the screen output.

Details

The function `irace.main` checks the correctness of the scenario, prints it, reads the parameter space from `scenario$parameterFile`, invokes `irace` and prints its results in various formatted ways. If you want a lower-level interface, please see function `irace`.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`irace.cmdline` a higher-level command-line interface to `irace.main`.

`readScenario` to read the scenario setup from a file.

`defaultScenario` to provide a default scenario for **irace**.

`irace.usage` *irace.usage*

Description

This function prints all command-line options of **irace**, with the corresponding switches and a short description.

Usage

```
irace.usage()
```

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

`irace.version` *irace.version*

Description

A character string containing the version of **irace**.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

parallelCoordinatesPlot *parallelCoordinatesPlot*

Description

'parallelCoordinatesPlot' plots a set of parameter configurations in parallel coordinates.

Usage

```
parallelCoordinatesPlot(configurations, parameters,
                        param_names = parameters$names, hierarchy = TRUE,
                        filename = NULL, pdf.width = 14, mar = c(8,1,4,1))
```

Arguments

configurations	Data frame containing target algorithms configurations in the format used by **irace**.
parameters	List of target algorithm parameters in the **irace** format.
param_names	Parameters names that should be included. Default: parameters$names.
hierarchy	If TRUE conditional parameters will be displayed in a different plot. Default TRUE.
filename	Filename prefix to generate the plots. If NULL the plot displayed but not saved.
pdf.width	Width for the pdf file generated.
mar	Margin to use for the plot. See par.

Value

A set of parallel coordinates plots showing the parameters values. If a filename is provided this plots are saved in one or more files.

Author(s)

Manuel López-Ibáñez and Leslie Pérez Cáceres

See Also

readParameters to obtain a valid parameter structure from a parameters file. readConfigurationsFile to obtain a set of target algorithm configurations from a configurations file.

Examples

```
## Not run:
## To use data obtained by irace

# First, load the data produced by irace.
load("irace.Rdata")
attach(iraceResults)
parallelCoordinatesPlot(allConfigurations, parameters, hierarchy = FALSE)

## End(Not run)
```

`parameterFrequency` *parameterFrequency*

Description

'parameterFrequency' plots the frequency of the parameters values of a set of target algorithm configurations.

Usage

```
parameterFrequency(configurations, parameters, rows = 4, cols = 3,
                   filename = NULL, pdf.width = 12, col = "gray")
```

Arguments

`configurations`	Data frame containing target algorithms configurations in the format used by **irace**.
`parameters`	List of target algorithm parameters in the **irace** format.
`rows`	Number of plots per column.
`cols`	Number of plots per row.
`filename`	Filename prefix to generate the plots. If `NULL` the plot displayed but not saved.
`pdf.width`	Width for the pdf file generated.
`col`	Color of the bar plot.

Value

A set of plots showing the Frequency of parameters values. If a filename is provided this plots are saved in one or more files.

Author(s)

Manuel López-Ibáñez and Leslie Pérez Cáceres

See Also

readParameters to obtain a valid parameter structure from a parameters file. readConfigurationsFile to obtain a set of target algorithm configurations from a configurations file.

Examples

```
## Not run:
## To use data obtained by irace

# First, load the data produced by irace.
load("irace.Rdata")
attach(iraceResults)
parameterFrequency(allConfigurations, parameters)

## End(Not run)
```

---

printScenario *printScenario*

Description

'printScenario' prints the given scenario.

Usage

```
printScenario(scenario)
```

Arguments

scenario	A list where tagged elements correspond to scenario settings of **irace**.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

readScenario for reading a configuration scenario from a file. defaultScenario returns the default scenario settings of **irace**. checkScenario to check that the scenario is valid.

`readConfigurationsFile` *readConfigurationsFile*

Description

'readConfigurationsFile' reads a set of target algorithms configurations from a file and puts them in **irace** format. The configurations are checked to match the parameters description provided.

Usage

```
readConfigurationsFile(filename, parameters, debugLevel = 0)
```

Arguments

`filename`	A filename from which the configurations should be read.
`parameters`	List of target algorithm parameters in the **irace** format.
`debugLevel`	Level of debug. Default: 0.

Value

A data frame containing the obtained configurations. Each row of the data frame is a candidate configuration, the columns correspond to the parameter names in `parameters`.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`readParameters` to obtain a valid parameter structure from a parameters list.

`readParameters` *readParameters*

Description

'readParameters' reads the parameters to be tuned by **irace** from a file or directly from a character string.

Usage

```
readParameters(file, digits = 4, debugLevel = 0, text)
```

Arguments

file	(optional) character string: the name of the file containing the definitions of the parameters to be tuned.
digits	The number of decimal places to be considered for the real parameters.
debugLevel	Integer: the debug level to increase the amount of output.
text	(optional) character string: if file is not supplied and this is, then parameters are read from the value of text via a text connection.

Details

Either 'file' or 'text' must be given. If 'file' is given, the parameters are read from the file 'file'. If 'text' is given instead, the parameters are read directly from the 'text' character string. In both cases, the parameters must be given (in 'text' or in the file whose name is 'file') in the expected form. See the documentation for details. If none of these parameters is given, **irace** will stop with an error.

A fixed parameter is a parameter that should not be sampled but instead should be always set to the only value of its domain. In this function we set isFixed to TRUE only if the parameter is a categorical and has only one possible value. If it is an integer and the minimum and maximum are equal, or it is a real and the minimum and maximum values satisfy 'round(minimum, digits) == round(maximum, digits)', then the parameter description is rejected as invalid to identify potential user errors.

Value

A list containing the definitions of the parameters read. The list is structured as follows:

names	Vector that contains the names of the parameters.
types	Vector that contains the type of each parameter 'i', 'c', 'r', 'o'.
switches	Vector that contains the switches to be used for the parameters on the command line.
domain	List of vectors, where each vector may contain two values (minimum, maximum) for real and integer parameters, or possibly more for categorical parameters.
conditions	List of R logical expressions, with variables corresponding to parameter names.
isFixed	Logical vectors that specifies which parameter is fixed and, thus, it does not need to be tuned.
nbParameters	An integer, the total number of parameters.
nbFixed	An integer, the number of parameters with a fixed value.
nbVariable	Number of variable (to be tuned) parameters.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

Examples

```
## Read the parameters directly from text
parameters.table <- 'tmax "" i (2, 10)
temp "" r (10, 50)
'
parameters <- readParameters(text=parameters.table)
parameters
```

---

readScenario *readScenario*

Description

'readScenario' reads the scenario to be used by **irace** from a file.

Usage

```
readScenario(filename = "", scenario = list())
```

Arguments

filename	A filename from which the scenario will be read. If empty, the default scenarioFile is used. An example scenario file is provided in system.file(package="irace", "templates/scenar
scenario	A list where tagged elements correspond to scenario settings for **irace**. This is an initial scenario that is overwritten for every parameter specified in the file to be read.

Value

The scenario list read from the file. The scenario parameter not present in the file are not present in the list, that is, they are NULL.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

checkScenario to check that the scenario is valid. defaultScenario to set the scenario to the default. printScenario to print the scenario.

`removeConfigurationsMetaData` *removeConfigurationsMetaData*

Description

Remove the columns with "metadata" of a matrix containing some configuration configurations. These "metadata" are used internaly by **irace**. This function can be used e.g. before printing the configurations, to output only the values for the parameters of the configuration without data possibly useless to the user.

Usage

```
removeConfigurationsMetaData(configurations)
```

Arguments

`configurations`	A matrix containg the configurations, one per row.

Value

The same matrix without the "metadata".

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`configurations.print.command` to print the configurations as command lines. `configurations.print` to print the configurations as a data frame.

`target.evaluator.default` *target.evaluator.default*

Description

`target.evaluator.default` is the default `targetEvaluator` function that is invoked if `targetEvaluator` is a string (by default `targetEvaluator` is `NULL` and this function is not invoked). You can use it as an advanced example of how to create your own `targetEvaluator` function.

Usage

```
target.evaluator.default(experiment, num.configurations, all.conf.id,
                         scenario, target.runner.call)
```

Arguments

`experiment` A list describing the experiment. It should contain at least:

- instanceA string containing the name of the instance (or filename and full path in case the instance is a file).
- idA numeric identifier for the candidate configuration that is evaluated. This must match the one passed earlier to `target.runner`.

`num.configurations`
The total number of candidate configurations evaluated in this iteration.

`all.conf.id` List of configuration ids that have been evaluated in the instance This is used, for example, when calculating the bounds for the hyper volume evaluation.

`scenario` options passed when invoking **irace**.

`target.runner.call`
a string describing the call to targetRunner that corresponds to this call to targetEvaluator. This is used only for providing extra information to the user, for example, in case targetEvaluator fails.

Value

This function returns the output of evaluating the candidate configuration, which must be a numerical value.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

`target.runner.default` *target.runner.default*

Description

`target.runner.default` is the default targetRunner function. You can use it as an advanced example of how to create your own targetRunner function.

Usage

```
target.runner.default(experiment, scenario)
```

Arguments

`experiment` A list describing the experiment. It should contain at least:

- instanceA string containing the name of the instance (or filename and full path in case the instance is a file).
- idA numeric identifier for the candidate configuration that is evaluated.
- configurationThe candidate configuration that must be run.
- seedA random seed

- extra.paramsExtra parameters (like instance-specific ones) to be passed when evaluating this candidate configuration.
- switchesCommand-line switches that correspond to each parameter

`scenario` options passed when invoking **irace**.

Value

If `targetEvaluator` is `NULL`, then this function returns the output of evaluating the candidate configuration, which must be a numerical value.

If `maxTime`, then this function returns the output of evaluating the candidate configuration and the execution time, if `targetEvaluator` is not `NULL`, this function must return only the execution time.

Otherwise, it returns a string. By the default, this string is the actual command-line call to the target-runner program. This information is only used for debugging purposes if `targetEvaluator` fails later.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

`testing.main` *testing.main*

Description

`testing.main` executes the testing of the target algorithm configurations found on an **irace** execution.

Usage

```
testing.main(logFile)
```

Arguments

`logFile` Path to the .Rdata file produced by **irace**.

Details

The function `testing.main` load the `logFile` and obtains the needed configurations according to the specified test. Use the `scenario$testNbElites` to test N final elite configurations or use `scenario$testIterationElites` to test the best configuration of each iteration. A test instance set must be provided through `scenario$testInstancesDir` and `testInstancesFile`.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`defaultScenario` to provide a default scenario for **irace**.

Index