Package ‘irace’

August 3, 2017

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 2.4

Date 2017-08-03

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, compiler

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 2017-08-03 17:34:51 UTC

R topics documented:

irace-package ................................................................. 2
buildCommandLine .......................................................... 5
checkIraceScenario .......................................................... 6
The irace package: 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.

Details

<table>
<thead>
<tr>
<th>Package</th>
<th>irace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Package</td>
</tr>
<tr>
<td>Version</td>
<td>2.4</td>
</tr>
<tr>
<td>Date</td>
<td>2017-08-03</td>
</tr>
<tr>
<td>License</td>
<td>GPL (&gt;= 2)</td>
</tr>
<tr>
<td>LazyLoad</td>
<td>yes</td>
</tr>
</tbody>
</table>
Author(s)

Maintainer: Manuel López-Ibáñez and Leslie Pérez Cáceres <irace@iridia.ulb.ac.be>

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

References


Examples

```r
# 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
                             ,tmmax = as.numeric(configuration>{{$"tmpmax"}})
                             ,temp = as.numeric(configuration>{{$"temp"}}))
  }

## New output interface in irace 2.0. This list may also contain:
## - 'time' if irace is called with 'maxTime'
## - 'error' is a string used to report an error
## - 'outputRaw' is a string used to report the raw output of calls to
## - an external program or function.
## - 'call' is a string used to report how target.runner called the
## - external program or function.
  return(list(cost = res$value))
}

## We define a configuration scenario by setting targetRunner to the
## function define above, instances to the first 100 random weights, and
## a maximum budget of 1000 calls to targetRunner.
scenario <- list(targetRunner = target.runner,
                  instances = weights[1:100],
                  maxExperiments = 1000,
                  logFile = "")

## We check that the scenario is valid. This will also try to execute
## target.runner.
checkIraceScenario(scenario, parameters = parameters)

## Not run:
## We are now ready to launch irace. We do it by means of the irace
## function. The function will print information about its
## progress. This may require a few minutes, so it is not run by default.
tuned.confns <- irace(scenario = scenario, parameters = parameters)

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

## 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:
test <- function(configuration)
{
  res <- lapply(weights[1:200],
    function(x) target.runner(
      experiment = list(instance = x,
        configuration = configuration),
        scenario = scenario))
  return (sapply(res, getElement, name = "cost"))
}
default <- test(data.frame(tmax=10, temp=10))

## We extract and apply the winning configuration found by irace
## to these instances:
tuned <- test(removeConfigurationsMetaData(tuned.confns[1,]))

## 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**

```r
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 parameter (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

```r
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)
```

Description

'checkIraceScenario' check the given scenario options.

Usage

```r
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.
checkScenario

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

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.
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.

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

**Usage**

```r
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")
- **repairConfiguration**: User-defined R function that takes a configuration generated by irace and repairs it. (Default: "")

**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 $10^{-\text{digits}}$. (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)
• batchmode: Specify how irace waits for jobs to finish when targetRunner submits jobs to a batch cluster: sge, pbs, torque or slurm (targetRunner must submit jobs to the cluster using, for example, 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 contains the instance and seeds used in the tuning. The order of this data frame indicates 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.
• 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.

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.

---

table

Description
'getConfigurationById' returns the configurations selected by id.

Usage
getConfigurationById(iraceResults=NULL, logFile=NULL, ids, drop.metadata=FALSE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iraceResults</td>
<td>Object created by irace and saved in scenario$logFile.</td>
</tr>
<tr>
<td>logFile</td>
<td>Log file created by irace, this file must contain the iraceResults object.</td>
</tr>
<tr>
<td>ids</td>
<td>The id or a vector of ids of the candidates configurations to obtain.</td>
</tr>
<tr>
<td>drop.metadata</td>
<td>Remove metadata, such the configuration ID and the ID of the parent, from the returned configurations. See removeConfigurationsMetaData.</td>
</tr>
</tbody>
</table>

Value
A data frame containing the elite configurations required.
getConfigurationByIteration

---

**getConfigurationByIteration**

**Description**

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

**Usage**

```r
getConfigurationByIteration(iraceResults = NULL, logfile = NULL,
                           iterations, drop.metadata = FALSE)
```

**Arguments**

- **iraceResults**: Object created by `irace` and saved in `scenario$logfile`.
- **logfile**: Log file created by `irace`; this file must contain the `iraceResults` object.
- **iterations**: The iteration number or a vector of iteration numbers from where the configurations should be obtained.
- **drop.metadata**: Remove metadata, such as the configuration ID and the ID of the parent, from the returned configurations. See `removeConfigurationsMetadata`.

**Value**

A data frame containing the elite configurations required.

---

getFinalElites

---

**getFinalElites**

**Description**

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

**Usage**

```r
getFinalElites(iraceResults = NULL, logfile = NULL, n = 0, drop.metadata = FALSE)
```

**Arguments**

- **iraceResults**: Object created by `irace` and saved in `scenario$logfile`.
- **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.metadata**: Remove metadata, such as the configuration ID and the ID of the parent, from the returned configurations. See `removeConfigurationsMetadata`. 
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.

Additionally, 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 \texttt{readParameters}.

• allConfigurations: The target algorithm configurations generated by \texttt{irace}. This object is a data frame, each row is a candidate configuration, the first column (.\texttt{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 (.\texttt{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 \texttt{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 (\texttt{allConfigurations$\_$ID}.).

• experimentLog: A matrix with columns \texttt{iteration}, \texttt{instance}, \texttt{configuration}, \texttt{time}. This matrix contains the log of all the experiments that \texttt{irace} performs during its execution. The instance column refers to the index of the \texttt{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 \texttt{irace}, the recovery is done using the information contained in this object.

• testing: A list that contains the testing results. The elements of this list are: \texttt{experiments} a matrix with the testing expriments of the selected configurations in the same format as the explained above and \texttt{seeds} a vector with the seeds used to execute each experiment.

\section*{Author(s)}
Manuel Lopez-Ibañez and Jérémie Dubois-Lacoste

\section*{See Also}
\texttt{irace.main} a higher-level command-line interface to \texttt{irace}.
\texttt{readScenario} to read the scenario setup from a file.
\texttt{defaultScenario} to provide a default scenario for \texttt{irace}.
\texttt{readParameters} to read the target algorithm parameters from a file.

\section*{Examples}
\begin{verbatim}
## Not run:
parameters <- readParameters("parameters.txt")
scenario <- readScenario(filename="scenario.txt",
    scenario=defaultScenario())
irace(scenario=scenario, parameters=parameters)

## End(Not run)
\end{verbatim}
irace.cmdline

Description

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

Usage

```r
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

Description

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

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste
**irace.main**

**Description**

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

**Usage**

```r
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**

**Description**

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

**Usage**

```r
irace.usage()
```

**Author(s)**

Manuel López-Ibáñez and Jérémie Dubois-Lacoste
**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**

```r
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
**parameterFrequency**

**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**

```r
## 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)
```

**Description**

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

**Usage**

```r
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

```r
## 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)
```

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.
Description

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

Usage

\begin{verbatim}
readConfigurationsFile(filename, parameters, debugLevel = 0, text)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{filename} \hfill A filename from which the configurations should be read.
  \item \texttt{parameters} \hfill List of target algorithm parameters in the \texttt{irace} format.
  \item \texttt{debugLevel} \hfill Level of debug. Default: 0.
  \item \texttt{text} \hfill (optional) character string: if filename is not supplied and this is, then configurations are read from the value of text via a text connection.
\end{itemize}

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 \texttt{parameters}.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

\begin{verbatim}
readParameters
\end{verbatim} to obtain a valid parameter structure from a parameters list.

Description

'readParameters' reads the parameters to be tuned by \texttt{irace} from a file or directly from a character string.

Usage

\begin{verbatim}
readParameters(file, digits = 4, debugLevel = 0, text)
\end{verbatim}
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.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste
Examples

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

Description

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

Usage

```r
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/scenario.txt")`.
- `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.
Description

Remove the columns with "metadata" of a matrix containing some configuration configurations. These "metadata" are used internally 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

```r
removeConfigurationsMetaData(configurations)
```

Arguments

- `configurations`: A matrix containing 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.

```r
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

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

experiment A list describing the experiment. It contains at least:
• id.configurationAn alphanumeric string that uniquely identifies a configuration;
• id.instanceAn alphanumeric string that uniquely identifies a pair (instance, seed);
• seedSeed for the random number generator to be used for this evaluation, ignore the seed for deterministic algorithms;
• instanceString giving the instance to be used for this evaluation;
• configuration1-row data frame with a column per parameter name;
• switchesVector of parameter switches (labels) in the order of parameters used in configuration.

num.configurations is the number of configurations alive in the race;
all.conf.id is the vector of configuration IDs of the alive configurations;
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 for providing extra information to the user, for example, in case targetEvaluator fails.

Value

The function targetEvaluator must return a list with one element "cost", the numerical value corresponding to the cost measure of the given configuration on the given instance.

The return list may also contain the following optional elements that are used by irace for reporting errors in targetEvaluator:
• erroris a string used to report an error;
• outputRawis a string used to report the raw output of calls to an external program or function;
• callis a string used to report how targetEvaluator called an external program or function.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

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

eperiment A list describing the experiment. It contains at least:

• id.configuration An alphanumeric string that uniquely identifies a configuration;
• id.instance An alphanumeric string that uniquely identifies a pair (instance, seed);
• seed Seed for the random number generator to be used for this evaluation, ignore the seed for deterministic algorithms;
• instance String giving the instance to be used for this evaluation;
• configuration 1-row data frame with a column per parameter name;
• switches Vector of parameter switches (labels) in the order of parameters used in configuration.

scenario Options passed when invoking \texttt{irace}.

Value

If \texttt{targetEvaluator} is \texttt{NULL}, then the \texttt{targetRunner} function must return a list with at least one element "cost", the numerical value corresponding to the evaluation of the given configuration on the given instance.

If the scenario option \texttt{maxTime} is non-zero, then the list must contain at least another element "time" that reports the execution time for this call to \texttt{targetRunner}.

The return list may also contain the following optional elements that are used by \texttt{irace} for reporting errors in \texttt{targetRunner}:

• error is a string used to report an error;
• outputRaw is a string used to report the raw output of calls to an external program or function;
• call is a string used to report how \texttt{targetRunner} called an external program or function.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

\begin{center}
| \texttt{testing.main} | \texttt{testing.main} |
\end{center}

Description

\texttt{testing.main} executes the testing of the target algorithm configurations found on an \texttt{irace} execution.
Usage

testing.main(logFile)

Arguments

logFile Path to the .Rdata file produced by \texttt{irace}.

Details

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

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

\texttt{defaultScenario} to provide a default scenario for \texttt{irace}. 
Index

*Topic **automatic configuration**
  irace-package, 2
*Topic **optimize**
  irace-package, 2
*Topic **package**
  irace-package, 2
*Topic **tuning**
  irace-package, 2

buildCommandLine, 5
checkIraceScenario, 6
checkScenario, 7, 7, 12, 20, 23
configurations.print, 8, 9, 24
configurations.print.command, 8, 8, 24
defaultScenario, 7, 9, 14, 15, 17, 20, 23, 27
getConfigurationById, 12
getConfigurationByIteration, 13
getFinalElites, 13
irace, 14, 17
irace-package, 2
irace.cmdline, 16, 17
irace.license, 16
irace.main, 15, 16, 17
irace.usage, 17
irace.version, 18

par, 18
parallelCoordinatesPlot, 18
parameterFrequency, 19
printScenario, 7, 12, 20, 23
readConfigurationsFile, 19, 20, 21
readParameters, 15, 19–21, 21
readScenario, 7, 12, 15, 17, 20, 23
removeConfigurationsMetaData, 24
target.evaluator.default, 24
target.runner.default, 25
testing.main, 26