Package 'irace'

October 23, 2022

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. M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, and M. Birattari (2016) <doi:10.1016/j.orp.2016.09.002>.

Version 3.5

Depends R (>= 3.2.0)

Imports stats, utils, compiler, R6

Suggests Rmpi (>= 0.6.0), parallel, knitr, testthat, withr, mlr (>= 2.15.0), ParamHelpers, devtools, covr

VignetteBuilder knitr

License GPL (>= 2)

URL https://mlopez-ibanez.github.io/irace/, https://github.com/MLopez-Ibanez/irace

BugReports https://github.com/MLopez-Ibanez/irace/issues

ByteCompile yes

Encoding UTF-8

RoxygenNote 7.1.1

SystemRequirements GNU make

NeedsCompilation yes

Author Manuel López-Ibáñez [aut, cre] (<https://orcid.org/0000-0001-9974-1295>), Jérémie Dubois-Lacoste [aut], Leslie Pérez Cáceres [aut], Thomas Stützle [aut], Mauro Birattari [aut], Eric Yuan [ctb],

Prasanna Balaprakash [ctb],
Nguyen Dang [ctb]

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

Repository CRAN

Date/Publication 2022-10-23 14:55:06 UTC

R topics documented:

irace-package *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. M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, and M. Birattari (2016) <doi:10.1016/j.orp.2016.09.002>.

Details

License: GPL (>= 2)

Author(s)

Maintainers: Manuel López-Ibáñez and Leslie Pérez Cáceres <irace-package@googlegroups.com>

References

Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. The irace package: Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives*, 2016. doi: 10.1016/j.orp.2016.09.002

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.

See Also

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

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 20 instances (in this case, weights):
weights <- rnorm(20, 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,log (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 1000.
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 <- stats::optim(par,fn, method="SANN",
              control=list(maxit=1000
                , tmax = as.numeric(configuration[["tmax"]])
                , 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 10 random weights, and
## a maximum budget of 'maxExperiments' calls to targetRunner.
scenario <- list(targetRunner = target_runner,
                 instances = weights[1:10],
                 maxExperiments = 500,
                 # Do not create a logFile
                 logFile = "")

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


## 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_confs <- irace(scenario = scenario, parameters = parameters)

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

## We can evaluate the quality of the best configuration found by
## irace versus the default configuration of the SANN algorithm on
## the other 10 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[11:20],
                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_confs[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))
```

ablation *Performs ablation between two configurations (from source to target).*

Description

Ablation is a method for analyzing the differences between two configurations.

Usage

```
ablation(
  iraceResults,
  src = 1L,
  target = NULL,
  ab.params = NULL,
  type = c("full", "racing"),
  n_instances = 1L,
  seed = 1234567,
  ablationLogFile = "log-ablation.Rdata",
  ...
)
```

Arguments

iraceResults	(list()|character(1)) iraceResults object created by **irace** and typically saved in the log file irace.Rdata. If a character string is given, then it is interpreted as the path to the log file from which the iraceResults object will be loaded.
src, target	Source and target configuration IDs. By default, the first configuration ever evaluated (ID 1) is used as src and the best configuration found by irace is used as target.
ab.params	Specific parameter names to be used for the ablation. They must be in parameters$names. By default, use all parameters.
type	Type of ablation to perform: "full" will execute each configuration on all n_instances to determine the best-performing one; "racing" will apply racing to find the best configurations.
n_instances	(integer(1)) Number of instances used in "full" ablation will be n_instances * scenario$firstTest.
seed	(integer(1)) Integer value to use as seed for the random number generation.
ablationLogFile	(character(1)) Log file to save the ablation log. If NULL, the results are not saved to a file.
...	Further arguments to override scenario settings, e.g., debugLevel, parallel, etc.

Value

A list containing the following elements:

configurations Configurations tested in the ablation.

instances A matrix with the instances used in the experiments. First column has the instances IDs from `iraceResults$scenario$instances`, second column the seed assigned to the instance.

experiments A matrix with the results of the experiments (columns are configurations, rows are instances).

scenario Scenario object with the settings used for the experiments.

trajectory IDs of the best configurations at each step of the ablation.

best Best configuration found in the experiments.

Author(s)

Leslie Pérez Cáceres and Manuel López-Ibáñez

References

C. Fawcett and H. H. Hoos. Analysing differences between algorithm configurations through ablation. Journal of Heuristics, 22(4):431–458, 2016.

See Also

`plotAblation()`

Examples

```
logfile <- system.file(package="irace", "exdata", "sann.rda")
# Execute ablation between the first and the best configuration found by irace.
ablog <- ablation(logfile, ablationLogFile = NULL)
plotAblation(ablog)
# Execute ablation between two selected configurations, and selecting only a
# subset of parameters, directly reading the setup from the irace log file.
ablog <- ablation(logfile, src = 1, target = 10,
                  ab.params = c("temp"), ablationLogFile = NULL)
plotAblation(ablog)
```

ablation_cmdline *Launch ablation with command-line options.*

Description

Launch `ablation()` with the same command-line options as the command-line executable (`ablation.exe` in Windows).

Usage

```
ablation_cmdline(argv = commandArgs(trailingOnly = TRUE))
```

Arguments

argv	(character()) The arguments provided on the R command line as a character vector, e.g., `c("-i", "irace.Rdata", "--src", 1)`.

Details

The function reads the parameters given on the command line used to invoke R, launches `ablation()` and possibly `plotAblation()`.

List of command-line options:

```
-l,--log-file           Path to the (.Rdata) file created by irace from which
                        the "iraceResults" object will be loaded.
-S,--src                Source configuration ID. Default: 1.
-T,--target             Target configuration ID. By default the best
                        configuration found by irace.
-P,--params             Specific parameter names to be used for the ablation
                        (separated with commas). By default use all
-t,--type               Type of ablation to perform: "full" will execute each
                        configuration on all "--n-instances" to determine the
                        best-performing one; "racing" will apply racing to
                        find the best configurations. Default: full.
-n,--n-instances        Number of instances used in "full" ablation will be
                        n_instances * scenario$firstTest. Default: 1.
   --seed               Integer value to use as seed for the random number
                        generation. Default: 1234567.
-o,--output-file        Log file to save the ablation log. If "", the results
                        are not saved to a file. Default: log-ablation.Rdata.
-p,--plot               Output filename (.pdf) for the plot. If not given, no
                        plot is created.
-O,--plot-type          Type of plot. Supported values are "mean" and
                        "boxplot". Default: mean.
   --old-path           Old path found in the log-file (.Rdata) given as input
                        to be replaced by --new-path.
```

```
   --new-path           New path to replace the path found in the log-file
                        (.Rdata) given as input.
-e,--exec-dir           Directory where the target runner will be run.
-s,--scenario           Scenario file to override the scenario given in the
                        log-file (.Rdata)
   --parallel           Number of calls to targetRunner to execute in
                        parallel. Values 0 or 1 mean no parallelization.
```

Value

A list containing the following elements:

configurations Configurations tested in the ablation.

instances A matrix with the instances used in the experiments. First column has the instances IDs from `iraceResults$scenario$instances`, second column the seed assigned to the instance.

experiments A matrix with the results of the experiments (columns are configurations, rows are instances).

scenario Scenario object with the settings used for the experiments.

trajectory IDs of the best configurations at each step of the ablation.

best Best configuration found in the experiments.

Author(s)

Manuel López-Ibáñez

Examples

```
# ablation_cmdline("--help")
```

`buildCommandLine` *Generate a command-line representation of a configuration*

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, thus generating 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).

Value

A string concatenating each element of `switches` and `values` for all parameters with a space between each pair of parameters (but none between the switches and the corresponding values).

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

Examples

```
switches <- c("--switch1 ", "--switch2 ")
values <- c("value_1", "value_2")
buildCommandLine (values, switches)
## Build a command-line from the results produced by a previous run of irace.
# First, load the data produced by irace.
irace.logfile <- file.path(system.file(package="irace"),
                           "exdata", "irace-acotsp.Rdata")
load(irace.logfile)
allConfigurations <- iraceResults$allConfigurations
parameters <- iraceResults$parameters
apply(allConfigurations[1:10, unlist(parameters$names)], 1, buildCommandLine,
      unlist(parameters$switches))
```

---

checkIraceScenario *Test that the given irace scenario can be run.*

Description

Test that the given irace scenario can be run by checking the scenario settings provided and trying to run the target-algorithm.

Usage

```
checkIraceScenario(scenario, parameters)
```

Arguments

scenario	(`list()`) Data structure containing **irace** settings. The data structure has to be the one returned by the function `defaultScenario()` or `readScenario()`.
parameters	(`list()`) Data structure containing the parameter space definition. The data structure has to similar to the one returned by the function `readParameters`.

Details

If the `parameters` argument is missing, then the parameters will be read from the file `parameterFile` given by `scenario`. If `parameters` is provided, then `parameterFile` will not be read. This function will try to execute the target-algorithm.

Value

returns `TRUE` if successful and gives an error and returns `FALSE` otherwise.

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

`checkScenario` to check that the scenario is valid.

---

checkParameters *checkParameters*

Description

FIXME: This is incomplete, for now we only repair inputs from previous irace versions.

Usage

```
checkParameters(parameters)
```

Arguments

`parameters`	(`list()`) Data structure containing the parameter space definition. The data structure has to similar to the one returned by the function `readParameters`.

checkScenario *Check and correct the given scenario*

Description

Checks for errors a (possibly incomplete) scenario setup of **irace** and transforms it into a valid scenario.

Usage

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

Arguments

scenario	(list()) Data structure containing **irace** settings. The data structure has to be the one returned by the function `defaultScenario()` or `readScenario()`.

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()` returns the default scenario settings of **irace**.

`checkScenario()` to check that the scenario is valid.

CommandArgsParser *R6 Class for parsing command-line arguments*

Description

R6 Class for parsing command-line arguments

R6 Class for parsing command-line arguments

`cmdline_usage()` prints the output of `--help`

Usage

```
cmdline_usage(cmdline_args)
```

Arguments

`cmdline_args`	Definition of the command-line arguments.

Methods

Public methods:

- `CommandArgsParser$new()`
- `CommandArgsParser$readCmdLineParameter()`
- `CommandArgsParser$readArg()`
- `CommandArgsParser$readAll()`
- `CommandArgsParser$cmdline_usage()`

Method `new()`:

Usage:

```
CommandArgsParser$new(argv, argsdef)
```

Method `readCmdLineParameter()`:

Usage:

```
CommandArgsParser$readCmdLineParameter(paramName, default = NULL)
```

Method `readArg()`:

Usage:

```
CommandArgsParser$readArg(short = "", long = "")
```

Method `readAll()`:

Usage:

```
CommandArgsParser$readAll()
```

Method `cmdline_usage()`:

Usage:

```
CommandArgsParser$cmdline_usage()
```

configurations.print *Print configurations as a data frame*

Description

Print configurations as a data frame

Usage

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

Arguments

configurations	(data.frame) Parameter configurations of the target algorithm (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-line strings.

configurations.print.command *Print configurations as command-line strings.*

Description

Prints configurations after converting them into a representation for the command-line.

Usage

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

Arguments

configurations	(data.frame) Parameter configurations of the target algorithm (one per row).
parameters	(list()) Data structure containing the parameter space definition. The data structure has to similar to the one returned by the function readParameters.

Value

None.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

configurations.print() to print the configurations as a data frame.

---

defaultScenario *Default scenario settings*

Description

Return scenario object with default values.

Usage

```
defaultScenario(scenario = list(), params_def = .irace.params.def)
```

Arguments

scenario	(list()) Data structure containing **irace** settings. The data structure has to be the one returned by the function defaultScenario() or readScenario().
params_def	(data.frame()) Definition of the options accepted by the scenario. This should only be modified by packages that wish to extend **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 options:

`scenarioFile` Path of the file that describes the configuration scenario setup and other irace settings. (Default: `"./scenario.txt"`)

`execDir` Directory where the programs will be run. (Default: `"./"`)

`logFile` File to save tuning results as an R dataset, either absolute path or relative to execDir. (Default: `"./irace.Rdata"`)

`quiet` Reduce the output generated by irace to a minimum. (Default: `0`)

`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 (by default, generate a random seed). (Default: `NA`)

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

`postselection` Percentage of the configuration budget used to perform a postselection race of the best configurations of each iteration after the execution of irace. (Default: `0`)

`aclib` Enable/disable AClib mode. This option enables compatibility with GenericWrapper4AC as targetRunner script. (Default: `0`)

- 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` In elitist irace, maximum number per race of elimination tests that do not eliminate a configuration. Use 0 for no limit. (Default: `2`)

- Internal `irace` options:

  `sampleInstances` Randomly sample the training instances or use them in the order given. (Default: `1`)

  `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`, `NULL` or `""`, it is computed as `10^-digits`. (Default: `""`)

  `nbIterations` Maximum number of iterations. (Default: `0`)

  `nbExperimentsPerIteration` Number of runs of the target algorithm per iteration. (Default: `0`)

  `minNbSurvival` Minimum number of configurations needed to continue the execution of each race (iteration). (Default: `0`)

  `nbConfigurations` Number of configurations to be sampled and evaluated at each iteration. (Default: `0`)

  `mu` Parameter used to define the number of configurations sampled and evaluated at each iteration. (Default: `5`)

- Target algorithm parameters:

  `parameterFile` File that contains the description of the parameters of the target algorithm. (Default: `"./parameters.txt"`)

  `forbiddenExps` Vector of R logical expressions that cannot evaluate to `TRUE` for any evaluated configuration. (Default: `""`)

forbiddenFile 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: "")

digits Maximum number of decimal places that are significant for numerical (real) parameters. (Default: 4)

- Target algorithm execution:

targetRunner Executable called for each configuration that executes the target algorithm to be tuned. See the templates and examples provided. (Default: "./target-runner")

targetRunnerLauncher Executable that will be used to launch the target runner, when targetRunner cannot be executed directly (.e.g, a Python script in Windows). (Default: "")

targetRunnerLauncherArgs Command-line arguments provided to targetRunnerLauncher. The substrings {targetRunner} and {targetRunnerArgs} will be replaced by the value of the option targetRunner and by the arguments usually passed when calling targetRunner, respectively. Example: "-m {targetRunner} --args {targetRunnerArgs}". (Default: "{targetRunner} {targetRunnerArgs}")

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. (Default: "")

targetRunnerParallel Optional R function to provide custom parallelization of targetRunner. (Default: "")

targetEvaluator Optional script or R function that provides a numeric value for each configuration. See templates/target-evaluator.tmpl (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. Values 0 or 1 mean no parallelization. (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, slurm or htcondor. targetRunner must submit jobs to the cluster using, for example, qsub. (Default: 0)

- Initial configurations:

initConfigurations Data frame describing initial configurations (usually read from a file using readConfigurations). (Default: "")

configurationsFile File that contains a table of initial configurations. If empty or NULL, all initial configurations are randomly generated. (Default: "")

- Training instances:

instances Character vector of the instances to be used in the targetRunner. (Default: "")

trainInstancesDir Directory where training 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` File that contains a list of training 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 in seconds for the executions of `targetRunner`. `targetRunner` must return two values: cost and time. (Default: `0`)

  `budgetEstimation` Fraction (smaller than 1) of the budget used to estimate the mean computation time of a configuration. Only used when `maxTime` > 0 (Default: `0.02`)

  `minMeasurableTime` Minimum time unit that is still (significantly) measureable. (Default: `0.01`)

- Statistical test:

  `testType` Statistical test used for elimination. The default value selects `t-test` if `capping` is enabled or `F-test`, otherwise. Valid values are: F-test (Friedman test), t-test (pairwise t-tests with no correction), t-test-bonferroni (t-test with Bonferroni's correction for multiple comparisons), t-test-holm (t-test with Holm's correction for multiple comparisons). (Default: `""`)

  `firstTest` Number of instances evaluated before the first elimination test. It must be a multiple of `eachTest`. (Default: `5`)

  `eachTest` Number of instances evaluated between elimination tests. (Default: `1`)

  `confidence` Confidence level for the elimination test. (Default: `0.95`)

- Adaptive capping:

  `capping` Enable the use of adaptive capping, a technique designed for minimizing the computation time of configurations. This is only available when `elitist` is active. (Default: `0`)

  `cappingType` Measure used to obtain the execution bound from the performance of the elite configurations.

  - median: Median performance of the elite configurations.
  - mean: Mean performance of the elite configurations.
  - best: Best performance of the elite configurations.
  - worst: Worst performance of the elite configurations.

  (Default: `"median"`)

  `boundType` Method to calculate the mean performance of elite configurations.

  - candidate: Mean execution times across the executed instances and the current one.
  - instance: Execution time of the current instance.

  (Default: `"candidate"`)

  `boundMax` Maximum execution bound for `targetRunner`. It must be specified when capping is enabled. (Default: `0`)

  `boundDigits` Precision used for calculating the execution time. It must be specified when capping is enabled. (Default: `0`)

  `boundPar` Penalization constant for timed out executions (executions that reach `boundMax` execution time). (Default: `1`)

`boundAsTimeout` Replace the configuration cost of bounded executions with `boundMax`. (Default: `1`)

- Recovery:

`recoveryFile` Previously saved log file to recover the execution of `irace`, either absolute path or relative to the current directory. If empty or `NULL`, recovery is not performed. (Default: `""`)

- Testing:

`testInstancesDir` Directory where testing instances are located, either absolute or relative to current directory. (Default: `""`)

`testInstancesFile` File containing a list of test instances and optionally additional parameters for them. (Default: `""`)

`testInstances` Character vector of the instances to be used in the `targetRunner` when executing the testing. (Default: `""`)

`testNbElites` Number of elite configurations returned by irace that will be tested if test instances are provided. (Default: `1`)

`testIterationElites` Enable/disable testing the elite configurations found at each iteration. (Default: `0`)

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()` returns the default scenario settings of **irace**.

`checkScenario()` to check that the scenario is valid.

---

getConfigurationById *Returns the configurations selected by ID.*

Description

Returns the configurations selected by ID.

Usage

```
getConfigurationById(iraceResults, ids, drop.metadata = FALSE)
```

Arguments

iraceResults	(list()|character(1)) iraceResults object created by **irace** and typically saved in the log file irace.Rdata. If a character string is given, then it is interpreted as the path to the log file from which the iraceResults object will be loaded.
ids	The id or a vector of ids of the candidates configurations to obtain.
drop.metadata	Remove metadata, such the configuration ID and the ID of the parent, from the returned configurations. See removeConfigurationsMetaData().

Value

A data frame containing the elite configurations required.

Author(s)

Manuel López-Ibáñez and Leslie Pérez Cáceres

---

getConfigurationByIteration

Returns the configurations by the iteration in which they were executed.

Description

Returns the configurations by the iteration in which they were executed.

Usage

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

Arguments

iraceResults	(list()|character(1)) iraceResults object created by **irace** and typically saved in the log file irace.Rdata. If a character string is given, then it is interpreted as the path to the log file from which the iraceResults object will be loaded.
iterations	The iteration number or a vector of iteration numbers from where the configurations should be obtained.
drop.metadata	(FALSE) Remove metadata, such the configuration ID and the ID of the parent, from the returned configurations. See removeConfigurationsMetaData.

Value

A data frame containing the elite configurations required.

Author(s)

Manuel López-Ibáñez and Leslie Pérez Cáceres

`getFinalElites` *Return the elite configurations of the final iteration.*

Description

Return the elite configurations of the final iteration.

Usage

```
getFinalElites(iraceResults, n = 0L, drop.metadata = FALSE)
```

Arguments

`iraceResults`	(`list()`|`character(1)`) `iraceResults` object created by **irace** and typically saved in the log file `irace.Rdata`. If a character string is given, then it is interpreted as the path to the log file from which the `iraceResults` object will be loaded.
`n`	Number of elite configurations to return, if `n` is larger than the number of configurations, then only the existing ones are returned. The default (`n=0`) returns all of them.
`drop.metadata`	Remove metadata, such the configuration ID and the ID of the parent, from the returned configurations. See `removeConfigurationsMetaData`.

Value

A data frame containing the elite configurations required.

Author(s)

Manuel López-Ibáñez and Leslie Pérez Cáceres

Examples

```
log_file <- system.file("exdata/irace-acotsp.Rdata", package="irace", mustWork=TRUE)
print(removeConfigurationsMetaData(getFinalElites(log_file, n=1)))
```

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	(list()) Data structure containing **irace** settings. The data structure has to be the one returned by the function defaultScenario() or readScenario().
parameters	(list()) Data structure containing the parameter space definition. The data structure has to similar to the one returned by the function readParameters.

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

(data.frame)

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 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 contains the testing results. The elements of this list are: `experiments` a matrix with the testing experiments 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 López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`irace.main()` a higher-level interface to `irace`.

`irace.cmdline()` a command-line interface to `irace`.

`readScenario()` for reading a configuration scenario from a file.

`readParameters()` read the target algorithm parameters from a file.

`defaultScenario()` returns the default scenario settings of **irace**.

`checkScenario()` to check that the scenario is valid.

Examples

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

## End(Not run)
```

`irace.cmdline` *Launch* `irace` *with command-line options.*

Description

Calls `irace.main()` using command-line options, maybe parsed from the command line used to invoke R.

Usage

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

Arguments

`argv`	(`character()`) 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 in the command line. It finally starts **irace** by calling `irace.main`.

List of command-line options:

```
-h,--help                Show this help.
-v,--version             Show irace package version.
-c,--check               Check scenario.
-i,--init                Initialize the working directory with template config
                         files.
   --only-test           Only test the configurations given in the file passed
                         as argument.
-s,--scenario            File that describes the configuration scenario setup
                         and other irace settings. Default: ./scenario.txt.
   --exec-dir            Directory where the programs will be run. Default: ./.
-p,--parameter-file      File that contains the description of the parameters
                         of the target algorithm. Default: ./parameters.txt.
   --forbidden-file      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.
   --configurations-file  File that contains a table of initial configurations.
                         If empty or NULL, all initial configurations are
                         randomly generated.
-l,--log-file            File to save tuning results as an R dataset, either
                         absolute path or relative to execDir. Default:
                         ./irace.Rdata.
```

```
--recovery-file       Previously saved log file to recover the execution of
                      irace, either absolute path or relative to the current
                      directory.  If empty or NULL, recovery is not
                      performed.
--train-instances-dir  Directory where training 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.
--train-instances-file  File that contains a list of training instances and
                      optionally additional parameters for them. If
                      trainInstancesDir is provided, irace will search for
                      the files in this folder.
--sample-instances    Randomly sample the training instances or use them in
                      the order given. Default: 1.
--test-instances-dir  Directory where testing instances are located, either
                      absolute or relative to current directory.
--test-instances-file  File containing a list of test instances and
                      optionally additional parameters for them.
--test-num-elites     Number of elite configurations returned by irace that
                      will be tested if test instances are provided.
                      Default: 1.
--test-iteration-elites  Enable/disable testing the elite configurations
                      found at each iteration. Default: 0.
--test-type           Statistical test used for elimination. The default
                      value selects t-test if capping is enabled or F-test,
                      otherwise. Valid values are: F-test (Friedman test),
                      t-test (pairwise t-tests with no correction),
                      t-test-bonferroni (t-test with Bonferroni's correction
                      for multiple comparisons), t-test-holm (t-test with
                      Holm's correction for multiple comparisons).
--first-test          Number of instances evaluated before the first
                      elimination test. It must be a multiple of eachTest.
                      Default: 5.
--each-test           Number of instances evaluated between elimination
                      tests. Default: 1.
--target-runner       Executable called for each configuration that executes
                      the target algorithm to be tuned. See the templates
                      and examples provided. Default: ./target-runner.
--target-runner-launcher  Executable that will be used to launch the target
                      runner, when targetRunner cannot be executed directly
                      (.e.g, a Python script in Windows).
--target-runner-args  Command-line arguments provided to
                      targetRunnerLauncher. The substrings \{targetRunner\}
                      and \{targetRunnerArgs\} will be replaced by the value
                      of the option targetRunner and by the arguments
                      usually passed when calling targetRunner,
                      respectively. Example: "-m {targetRunner --args
```

```
                        {targetRunnerArgs}"}. Default: {targetRunner}
                        {targetRunnerArgs}.
    --target-runner-retries  Number of times to retry a call to targetRunner if
                        the call failed. Default: 0.
    --target-evaluator  Optional script or R function that provides a numeric
                        value for each configuration. See
                        templates/target-evaluator.tmpl
    --deterministic     If the target algorithm is deterministic,
                        configurations will be evaluated only once per
                        instance. Default: 0.
    --max-experiments   Maximum number of runs (invocations of targetRunner)
                        that will be performed. It determines the maximum
                        budget of experiments for the tuning. Default: 0.
    --max-time          Maximum total execution time in seconds for the
                        executions of targetRunner. targetRunner must return
                        two values: cost and time. Default: 0.
    --budget-estimation Fraction (smaller than 1) of the budget used to
                        estimate the mean computation time of a configuration.
                        Only used when maxTime > 0 Default: 0.02.
    --min-measurable-time  Minimum time unit that is still (significantly)
                        measureable. Default: 0.01.
    --parallel          Number of calls to targetRunner to execute in
                        parallel. Values 0 or 1 mean no parallelization.
                        Default: 0.
    --load-balancing    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, slurm or htcondor. targetRunner must
                        submit jobs to the cluster using, for example, qsub.
                        Default: 0.
    --digits            Maximum number of decimal places that are significant
                        for numerical (real) parameters. Default: 4.
-q,--quiet              Reduce the output generated by irace to a minimum.
                        Default: 0.
    --debug-level       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 (by default,
                        generate a random seed).
    --soft-restart      Enable/disable the soft restart strategy that avoids
                        premature convergence of the probabilistic model.
```

```
                          Default: 1.
      --soft-restart-threshold  Soft restart threshold value for numerical
                          parameters. If NA, NULL or "", it is computed as
                          10^-digits.
   -e,--elitist           Enable/disable elitist irace. Default: 1.
      --elitist-new-instances  Number of instances added to the execution list
                          before previous instances in elitist irace. Default:
                          1.
      --elitist-limit     In elitist irace, maximum number per race of
                          elimination tests that do not eliminate a
                          configuration. Use 0 for no limit. Default: 2.
      --capping           Enable the use of adaptive capping, a technique
                          designed for minimizing the computation time of
                          configurations. This is only available when elitist is
                          active. Default: 0.
      --capping-type      Measure used to obtain the execution bound from the
                          performance of the elite configurations: median, mean,
                          worst, best. Default: median.
      --bound-type        Method to calculate the mean performance of elite
                          configurations: candidate or instance. Default:
                          candidate.
      --bound-max         Maximum execution bound for targetRunner. It must be
                          specified when capping is enabled. Default: 0.
      --bound-digits      Precision used for calculating the execution time. It
                          must be specified when capping is enabled. Default: 0.
      --bound-par         Penalization constant for timed out executions
                          (executions that reach boundMax execution time).
                          Default: 1.
      --bound-as-timeout  Replace the configuration cost of bounded executions
                          with boundMax. Default: 1.
      --postselection     Percentage of the configuration budget used to perform
                          a postselection race of the best configurations of
                          each iteration after the execution of irace. Default:
                          0.
      --aclib             Enable/disable AClib mode. This option enables
                          compatibility with GenericWrapper4AC as targetRunner
                          script. Default: 0.
      --iterations        Maximum number of iterations. Default: 0.
      --experiments-per-iteration  Number of runs of the target algorithm per
                          iteration. Default: 0.
      --min-survival      Minimum number of configurations needed to continue
                          the execution of each race (iteration). Default: 0.
      --num-configurations  Number of configurations to be sampled and evaluated
                          at each iteration. Default: 0.
      --mu                Parameter used to define the number of configurations
                          sampled and evaluated at each iteration. Default: 5.
      --confidence        Confidence level for the elimination test. Default:
                          0.95.
```

Value

`(invisible(data.frame))`

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` 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 contains the testing results. The elements of this list are: `experiments` a matrix with the testing experiments 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 López-Ibáñez and Jérémie Dubois-Lacoste

See Also

irace.main() to start **irace** with a given scenario.

Examples

```
irace.cmdline("--version")
```

`irace.license` *irace.license*

Description

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

Usage

```
irace.license
```

Format

An object of class `character` of length 1.

`irace.main` *Higher-level interface to launch irace.*

Description

Higher-level interface to launch irace.

Usage

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

Arguments

`scenario`	(`list()`) Data structure containing **irace** settings. The data structure has to be the one returned by the function `defaultScenario()` or `readScenario()`.
`output.width`	(`integer(1)`) The width used for the screen output.

Details

This function checks the correctness of the scenario, reads the parameter space from `scenario$parameterFile`, invokes `irace()`, prints its results in various formatted ways, (optionally) calls `psRace()` and, finally, evaluates the best configurations on the test instances (if provided). If you want a lower-level interface that just runs irace, please see function `irace()`.

Value

`(invisible(data.frame))`

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` 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 contains the testing results. The elements of this list are: `experiments` a matrix with the testing experiments 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 López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`irace.cmdline()` 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**.

`irace.version` *irace.version*

Description

A character string containing the version of `irace`.

Usage

```
irace.version
```

Format

An object of class `character` of length 1.

`path_rel2abs` *Converts a relative path to an absolute path. It tries really hard to create canonical paths.*

Description

Converts a relative path to an absolute path. It tries really hard to create canonical paths.

Usage

```
path_rel2abs(path, cwd = getwd())
```

Arguments

`path`	(`character(1)`) Character string representing a relative path.
`cwd`	(`character(1)`) Current working directory.

Value

(`character(1)`) Character string representing the absolute path

Examples

```
path_rel2abs("..")
```

plotAblation *Create plot from an ablation log*

Description

Create plot from an ablation log

Usage

```
plotAblation(
  ablog,
  pdf.file = NULL,
  pdf.width = 20,
  type = c("mean", "boxplot"),
  mar = par("mar"),
  ylab = "Mean configuration cost",
  ylim = NULL,
  ...
)
```

Arguments

ablog	(list()|character(1)) Ablation log object returned by ablation(). Alternatively, the path to an .Rdata file, e.g., "log-ablation.Rdata", from which the object will be loaded.
pdf.file	Output filename.
pdf.width	Width provided to create the pdf file.
type	Type of plot. Supported values are "mean" and "boxplot".
mar	Vector with the margins for the ablation plot.
ylab	Label of y-axis.
ylim	Numeric vector of length 2, giving the y coordinates ranges.
...	Further graphical parameters may also be supplied as arguments. See graphics::plot.default().

Author(s)

Leslie Pérez Cáceres and Manuel López-Ibáñez

See Also

ablation()

Examples

```
logfile <- file.path(system.file(package="irace"), "exdata", "log-ablation.Rdata")
plotAblation(ablog = logfile)
```

printParameters *Print parameter space in the textual format accepted by irace.*

Description

FIXME: Dependent parameter bounds are not supported yet.

Usage

```
printParameters(params, digits = 15L)
```

Arguments

params	(list()) Parameter object stored in irace.Rdata or read with irace::readParameters().
digits	(integer()) The desired number of digits after the decimal point for real-valued parameters. Default is 15, but it should be the value in scenario$digits.

Examples

```
 parameters.table <- '
 # name       switch           type  values               [conditions (using R syntax)]
 algorithm    "--"             c     (as,mmas,eas,ras,acs)
 localsearch  "--localsearch " c     (0, 1, 2, 3)
 ants         "--ants "        i,log (5, 100)
 q0           "--q0 "          r     (0.0, 1.0)           | algorithm == "acs"
 nnls         "--nnls "        i     (5, 50)              | localsearch %in% c(1,2,3)
 '
parameters <- readParameters(text=parameters.table)
printParameters(parameters)
```

printScenario *Prints the given scenario*

Description

Prints the given scenario

Usage

```
printScenario(scenario)
```

Arguments

scenario	(list()) Data structure containing **irace** settings. The data structure has to be the one returned by the function defaultScenario() or readScenario().

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()` returns the default scenario settings of **irace**.
`checkScenario()` to check that the scenario is valid.

---

psRace *psRace*

Description

psRace performs a postselection race a set of configurations.

Usage

```
psRace(
  iraceLogFile = NULL,
  iraceResults = NULL,
  conf.ids = NULL,
  postselection = NULL,
  max.experiments = NULL,
  elites = FALSE,
  seed = 1234567
)
```

Arguments

iraceLogFile	NULL Log file created by **irace**, this file must contain the iraceResults object.
iraceResults	NULL Object created by **irace** and saved in scenario$logFile.
conf.ids	NULL IDs of the configurations in iraceResults$allConfigurations to be used for ablation. If NULL, the elites argument will be used.
postselection	NULL Percentage of the maxExperiments provided in the scenario to be used in the race.
max.experiments	NULL Number of experiments available for the race. If NULL budget for the race is set by the parameter scenario$postselection, which defines the percentage of the total budget of **irace** (iraceResults$scenario$maxExperiments or iraceResults$scenario$maxTime/iraceResults$state$timeEstimate) to use for the postselection.
elites	FALSE Flag for selecting configurations. If FALSE, the best configurations of each iteration are used for the race. If TRUE, the elite configurtions of each iteration are used for the race.
seed	1234567 Numerical value to use as seed for the random number generation.

Value

If iraceLogFile is NULL, it returns a list with the following elements:

configurations Configurations used in the race.

instances A matrix with the instances used in the experiments. First column has the instances ids from iraceResults$scenario$instances, second column the seed assigned to the instance.

maxExperiments Maximum number of experiments set for the race.

experiments A matrix with the results of the experiments (columns are configurations, rows are instances).

elites Best configurations found in the experiments.

If `iraceLogFile` is provided this list object will be saved in `iraceResults$psrace.log`.

Author(s)

Leslie Pérez Cáceres

Examples

```
## Not run:
  # Execute the postselection automatically after irace
  scenario <- readScenario(filename="scenario.txt")
  parameters <- readParameters("parameters.txt")
  # Use 10% of the total budget
  scenario$postselection <- 0.1
  irace(scenario=scenario, parameters=parameters)
  # Execute the postselection after the execution of \pkg{irace}.
  psRace(iraceLogFile="irace.Rdata", max.experiments=120)

## End(Not run)
```

`readConfigurationsFile` *Read parameter configurations from a file*

Description

Reads a set of target-algorithm 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, text)
```

Arguments

filename	(character(1)) Filename from which the configurations should be read. The contents should be readable by read.table(, header=TRUE).
parameters	(list()) Data structure containing the parameter space definition. The data structure has to similar to the one returned by the function readParameters.
debugLevel	(integer(1)) Larger values produce more verbose output.
text	(character(1)) If file is not supplied and this is, then parameters are read from the value of text via a text connection.

Details

Example of an input file:

```
# This is a comment line
<param_name_1>  <param_name_2> ...
      0.5           "value_1"    ...
      1.0           "value_2"    ...
      1.2           "value_3"    ...
      ...           ...
```

The order of the columns does not necessarily have to be the same as in the file containing the definition of the parameters.

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 file.

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

Description

Reads the parameters to be tuned by **irace** from a file or from a character string.

Usage

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

Arguments

file	(character(1)) Filename containing the definitions of the parameters to be tuned.
digits	The number of decimal places to be considered for the real parameters.
debugLevel	(integer(1)) Larger values produce more verbose output.
text	(character(1)) 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'. Numerical parameters can be sampled in a log-scale with 'i,log' and 'r,log' (no spaces).

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 vector 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.

`depends` List of character vectors, each vector specifies which parameters depend on this one.

`isDependent` Logical vector that specifies which parameter has a dependent domain.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

Examples

```
## Read the parameters directly from text
parameters.table <- '
# name       switch           type  values               [conditions (using R syntax)]
algorithm    "--"             c     (as,mmas,eas,ras,acs)
localsearch  "--localsearch " c     (0, 1, 2, 3)
alpha        "--alpha "       r     (0.00, 5.00)
beta         "--beta "        r     (0.00, 10.00)
rho          "--rho  "        r     (0.01, 1.00)
ants         "--ants "        i,log (5, 100)
q0           "--q0 "          r     (0.0, 1.0)           | algorithm == "acs"
rasrank      "--rasranks "    i     (1, "min(ants, 10)") | algorithm == "ras"
elitistants  "--elitistants " i     (1, ants)            | algorithm == "eas"
nnls         "--nnls "        i     (5, 50)              | localsearch %in% c(1,2,3)
dlb          "--dlb "         c     (0, 1)               | localsearch %in% c(1,2,3)
'
parameters <- readParameters(text=parameters.table)
str(parameters)
```

`readScenario` *Reads from a file the scenario settings to be used by* **irace**.

Description

Reads from a file the scenario settings to be used by **irace**.

Usage

```
readScenario(filename = "", scenario = list(), params_def = .irace.params.def)
```

Arguments

filename	(character(1)) 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.tmpl").
scenario	(list()) Data structure containing **irace** settings. The data structure has to be the one returned by the function defaultScenario() or readScenario(). This is an initial scenario that is overwritten
params_def	(data.frame()) Definition of the options accepted by the scenario. This should only be modified by packages that wish to extend **irace**.

Value

The scenario list read from the file. The scenario settings not present in the file are not present in the list, i.e., they are NULL.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

printScenario() prints the given scenario.

defaultScenario() returns the default scenario settings of **irace**.

checkScenario() to check that the scenario is valid.

---

read_logfile *Read the log file produced by irace (*irace.Rdata*).*

Description

Read the log file produced by irace (irace.Rdata).

Usage

```
read_logfile(filename, name = "iraceResults")
```

Arguments

filename	Filename that contains the log file saved by irace. Example: irace.Rdata.
name	Optional argument that allows overriding the default name of the object in the file.

Value

(list())

`read_pcs_file` *Read parameters in PCS (AClib) format and write them in irace format.*

Description

Read parameters in PCS (AClib) format and write them in irace format.

Usage

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

Arguments

`file`	(`character(1)`) Filename containing the definitions of the parameters to be tuned.
`digits`	The number of decimal places to be considered for the real parameters.
`debugLevel`	(`integer(1)`) Larger values produce more verbose output.
`text`	(`character(1)`) 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.

FIXME: Forbidden configurations, default configuration and transformations ("log") are currently ignored. See https://github.com/MLopez-Ibanez/irace/issues/31

Value

A string representing the parameters in irace format.

Author(s)

Manuel López-Ibáñez

References

Frank Hutter, Manuel López-Ibáñez, Chris Fawcett, Marius Thomas Lindauer, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. **AClib: A Benchmark Library for Algorithm Configuration**. In P. M. Pardalos, M. G. C. Resende, C. Vogiatzis, and J. L. Walteros, editors, *Learning and Intelligent Optimization, 8th International Conference, LION 8*, volume 8426 of Lecture Notes in Computer Science, pages 36–40. Springer, Heidelberg, 2014.

Examples

```
## Read the parameters directly from text
pcs_table <- '
# name       domain
algorithm    {as,mmas,eas,ras,acs}[as]
localsearch  {0, 1, 2, 3}[0]
alpha        [0.00, 5.00][1]
beta         [0.00, 10.00][1]
rho          [0.01, 1.00][0.95]
ants         [5, 100][10]i
q0           [0.0, 1.0][0]
rasrank      [1, 100][1]i
elitistants  [1, 750][1]i
nnls         [5, 50][5]i
dlb          {0, 1}[1]
Conditionals:
q0 | algorithm in {acs}
rasrank | algorithm in {ras}
elitistants | algorithm in {eas}
nnls | localsearch in {1,2,3}
dlb | localsearch in {1,2,3}
'
parameters_table <- read_pcs_file(text=pcs_table)
cat(parameters_table)
parameters <- readParameters(text=parameters_table)
str(parameters)
```

`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`	(`data.frame`) Parameter configurations of the target algorithm (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.

---

scenario_update_paths *Update filesystem paths of a scenario consistently.*

Description

This function should be used to change the filesystem paths stored in a scenario object. Useful when moving a scenario from one computer to another.

Usage

```
scenario_update_paths(scenario, from, to, fixed = TRUE)

scenario.update.paths(scenario, from, to, fixed = TRUE)
```

Arguments

`scenario`	(`list()`) Data structure containing **irace** settings. The data structure has to be the one returned by the function `defaultScenario()` or `readScenario()`.
`from`	character string containing a regular expression (or character string for `fixed = TRUE`) to be matched.
`to`	the replacement string.character string. For `fixed = FALSE` this can include backreferences `"\1"` to `"\9"` to parenthesized subexpressions of `from`.
`fixed`	logical. If `TRUE`, `from` is a string to be matched as is.

Value

The updated scenario

See Also

`base::grep()`

Examples

```
## Not run:
scenario <- readScenario(filename = "scenario.txt")
scenario <- scenario_update_paths(scenario, from = "/home/manuel/", to = "/home/leslie")

## End(Not run)
```

`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 contains at least: `id.configuration` An alphanumeric string that uniquely identifies a configuration; `id.instance` An alphanumeric string that uniquely identifies an instance; `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; `bound` (only when `capping` is enabled) Time bound for the execution; `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`.
`num.configurations`	Number of configurations alive in the race.
`all.conf.id`	Vector of configuration IDs of the alive configurations.
`scenario`	(`list()`) Data structure containing **irace** settings. The data structure has to be the one returned by the function `defaultScenario()` or `readScenario()`.

`target.runner.call`
: 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`:

`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 `targetRunner` called an external program or function.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

`target.runner.default` *Default* `targetRunner` *function.*

Description

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 contains at least:

`id.configuration` An alphanumeric string that uniquely identifies a configuration;

`id.instance` An alphanumeric string that uniquely identifies an instance;

`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;

`bound` (only when `capping` is enabled) Time bound for the execution;

`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`
: (`list()`)
Data structure containing **irace** settings. The data structure has to be the one returned by the function `defaultScenario()` or `readScenario()`.

Value

If `targetEvaluator` is `NULL`, then the `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 `maxTime` is non-zero or if `capping` is enabled then the list must contain at least another element `"time"` that reports the execution time for this call to `targetRunner`. The return list may also contain the following optional elements that are used by **irace** for reporting errors in `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 `targetRunner` called an external program or function.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

`testConfigurations` *Execute the given configurations on the testing instances specified in the scenario*

Description

Execute the given configurations on the testing instances specified in the scenario

Usage

```
testConfigurations(configurations, scenario, parameters)
```

Arguments

`configurations`	(`data.frame`) Parameter configurations of the target algorithm (one per row).
`scenario`	(`list()`) Data structure containing **irace** settings. The data structure has to be the one returned by the function `defaultScenario()` or `readScenario()`.
`parameters`	(`list()`) Data structure containing the parameter space definition. The data structure has to similar to the one returned by the function `readParameters`.

Details

A test instance set must be provided through `scenario[["testInstances"]]`.

Value

A list with the following elements:

`experiments` Experiments results.

`seeds` Array of the instance seeds used in the experiments.

Author(s)

Manuel López-Ibáñez

See Also

`testing_fromlog()`

`testing_fromfile` *Test configurations given an explicit table of configurations and a scenario file*

Description

Executes the testing of an explicit list of configurations given in `filename` (same format as in `readConfigurationsFile()`). A `logFile` is created unless disabled in `scenario`. This may overwrite an existing one!

Usage

```
testing_fromfile(filename, scenario)
```

Arguments

`filename`	Path to a file containing configurations: one configuration per line, one parameter per column, parameter names in header.
`scenario`	(`list()`) Data structure containing **irace** settings. The data structure has to be the one returned by the function `defaultScenario()` or `readScenario()`.

Value

iraceResults

Author(s)

Manuel López-Ibáñez

See Also

`testing_fromlog()` provides a different interface for testing.

testing_fromlog *Test configurations given in* .Rdata *file*

Description

testing_fromlog executes the testing of the target algorithm configurations found by an **irace** execution.

Usage

```
testing_fromlog(
  logFile,
  testNbElites,
  testIterationElites,
  testInstancesDir,
  testInstancesFile,
  testInstances
)
```

Arguments

logFile	Path to the .Rdata file produced by **irace**.
testNbElites	Number of (final) elite configurations to test. Overrides the value found in logFile.
testIterationElites	(logical(1)) If FALSE, only the final testNbElites configurations are tested; otherwise, also test the best configurations of each iteration. Overrides the value found in logFile.
testInstancesDir	Directory where testing instances are located, either absolute or relative to current directory.
testInstancesFile	File containing a list of test instances and optionally additional parameters for them.
testInstances	Character vector of the instances to be used in the targetRunner when executing the testing.

Details

The function testing_fromlog loads the logFile and obtains the testing setup and configurations to be tested. Within the logFile, the variable scenario$testNbElites specifies how many final elite configurations to test and scenario$testIterationElites indicates whether test the best configuration of each iteration. The values may be overridden by setting the corresponding arguments in this function. The set of testing instances must appear in scenario[["testInstances"]].

Value

Boolean. TRUE if the testing ended successfully otherwise, FALSE.

Author(s)

Manuel López-Ibáñez and Leslie Pérez Cáceres

See Also

defaultScenario() to provide a default scenario for **irace**. testing_fromfile() provides a different interface for testing.

Index