

# Package ‘irace’

May 8, 2026

**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](https://doi.org/10.1016/j.orp.2016.09.002)>.

**Version** 4.4.3

**Depends** R (>= 4.0.0)

**Imports** R6, codetools, compiler, data.table (>= 1.15.0), fs, matrixStats (>= 1.4.1), spacefillr, stats, utils, withr

**Suggests** Rmpi (>= 0.6.0), highr, knitr, parallel, testthat (>= 3.1.8)

**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** 7.3.3

**SystemRequirements** GNU make

**Config/testthat/edition** 3

**NeedsCompilation** yes

**Author** Manuel López-Ibáñez [aut, cre] (ORCID:  
<<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** 2026-04-23 14:20:02 UTC

## Contents

irace-package . . . . .	3
ablation . . . . .	4
ablation_cmdline . . . . .	6
buildCommandLine . . . . .	8
checkIraceScenario . . . . .	9
checkParameters . . . . .	10
checkScenario . . . . .	10
check_output_target_runner . . . . .	11
configurations_print . . . . .	12
configurations_print_command . . . . .	12
defaultScenario . . . . .	13
getConfigurationById . . . . .	18
getConfigurationByIteration . . . . .	19
getFinalElites . . . . .	20
get_instanceID_seed_pairs . . . . .	21
has_testing_data . . . . .	22
irace . . . . .	22
irace_cmdline . . . . .	26
irace_license . . . . .	32
irace_main . . . . .	33
irace_summarise . . . . .	34
irace_version . . . . .	35
multi_irace . . . . .	36
parameters . . . . .	37
path_rel2abs . . . . .	39
plotAblation . . . . .	39
printParameters . . . . .	40
printScenario . . . . .	41
psRace . . . . .	42
random_seed . . . . .	43
readConfigurationsFile . . . . .	44
readParameters . . . . .	46
readScenario . . . . .	48
read_ablogfile . . . . .	49
read_logfile . . . . .	49
read_pcs_file . . . . .	50
removeConfigurationsMetaData . . . . .	51
save_irace_logfile . . . . .	52

<i>irace-package</i>	3
scenario_update_paths . . . . .	53
target_evaluator_default . . . . .	54
target_runner_default . . . . .	55
testConfigurations . . . . .	56
testing_fromfile . . . . .	57
testing_fromlog . . . . .	58
<b>Index</b>	<b>60</b>

---

<i>irace-package</i>	<i>The irace package: Iterated Racing for Automatic Algorithm Configuration</i>
----------------------	---

---

## 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](mailto: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\(\)](#) for examples and `vignette(package = "irace")` for the user-guide.

---

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"),
  nrep = 1L,
  seed = 1234567L,
  ablationLogFile = "log-ablation.Rdata",
  instancesFile = "train",
  ...
)
```

## Arguments

iraceResults	list() character(1) Object created by <b>irace</b> and typically saved in the log file <code>irace.Rdata</code> . If a character string is given, then it is interpreted as the path to the log file from which the <code>iraceResults</code> object will be loaded.
src, target	integer(1) character(1) Source and target configuration IDs. By default, the first configuration ever evaluated (ID 1) is used as <code>src</code> and the best configuration found by <code>irace</code> is used as <code>target</code> . If the argument is a string, it is interpreted as the path to a file, with the format specified by <code>readConfigurationsFile()</code> , that contains the configuration.
ab_params	character() Specific parameter names to be used for the ablation. They must be in <code>parameters\$names</code> . By default, use all parameters.
type	"full" "racing" Type of ablation to perform: "full" will execute each configuration on all <code>n_instances</code> to determine the best-performing one; "racing" will apply racing to find the best configurations.
nrep	integer(1) Number of replications per instance used in "full" ablation. When <code>nrep &gt; 1</code> , each configuration will be executed <code>nrep</code> times on each instance with different random seeds.

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.
instancesFile	character(1) Instances file used for ablation: 'train', 'test' or a filename containing the list of instances.
...	Further arguments to override scenario settings, e.g., debugLevel, parallel, etc.

**Value**

A list containing the following elements:

**allConfigurations** Configurations tested in the ablation.

**state** State of the ablation process.

**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.

**complete** TRUE if the ablation process was completed.

**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\(\)](#) [ablation\\_cmdline\(\)](#)

**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, type = "mean")
```

---

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:

<code>-l,--log-file</code>	Path to the (.Rdata) file created by irace from which the "iraceResults" object will be loaded.
<code>-S,--src</code>	Source configuration ID or the path to a file containing the configuration. Default: 1.
<code>-T,--target</code>	Target configuration ID (by default the best configuration found by irace) or the path to a file containing the configuration.
<code>-P,--params</code>	Specific parameter names to be used for the ablation (separated with commas). By default use all
<code>-t,--type</code>	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.
<code>-n,--nrep</code>	Number of replications per instance used in "full" ablation. Default: 1.
<code>--seed</code>	Integer value to use as seed for the random number generation. Default: 1234567.
<code>-o,--output-file</code>	Log file to save the ablation log. If "", the results are not saved to a file. Default: log-ablation.Rdata.
<code>--instances-file</code>	Instances file used for ablation: "train", "test" or a

	filename containing the list of instances. Default: train.
-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", "boxplot", "rank" or "rank,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:

**allConfigurations** Configurations tested in the ablation.

**state** State of the ablation process.

**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.

**complete** TRUE if the ablation process was completed.

### Author(s)

Manuel López-Ibáñez

### See Also

[plotAblation\(\)](#) [ablation\(\)](#)

### Examples

```
ablation_cmdline("--help")
# Find the ablation command-line executable:
Sys.glob(file.path(system.file(package="irace", "bin"), "ablation*"))
```

---

buildCommandLine	<i>Generate a command-line representation of a configuration</i>
------------------	--

---

### 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 parameter (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-", "--switch3=")
values <- list("value_1", 1L, sqrt(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.
logfile <- file.path(system.file(package="irace"), "exdata", "irace-acotsp.Rdata")
iraceResults <- read_logfile(logfile)
allConfigurations <- iraceResults$allConfigurations
parameters <- iraceResults$scenario$parameters
apply(allConfigurations[1:10, unlist(parameters$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)
```

### Arguments

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

### 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	<i>checkParameters</i>
-----------------	------------------------

---

**Description**

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

**Usage**

```
checkParameters(parameters)
```

**Arguments**

parameters	ParameterSpace Data structure containing the parameter space definition. The data structure has to similar to the one returned by the function <a href="#">readParameters</a> .
------------	--

---

checkScenario	<i>Check and correct the given scenario</i>
---------------	---

---

**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 <b>irace</b> settings. The data structure has to be the one returned by the function <a href="#">defaultScenario()</a> or <a href="#">readScenario()</a> .
----------	--

**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.

---

check\_output\_target\_runner

*Check the output of the target runner and repair it if possible. If the output is incorrect, this function will throw an error.*

---

**Description**

Check the output of the target runner and repair it if possible. If the output is incorrect, this function will throw an error.

**Usage**

```
check_output_target_runner(output, scenario, bound = NULL)
```

**Arguments**

output	The output from target runner.
scenario	<code>list()</code> Data structure containing <b>irace</b> settings. The data structure has to be the one returned by the function <a href="#">defaultScenario()</a> or <a href="#">readScenario()</a> .
bound	Optional time bound that the target runner should have respected.

**Value**

The output with its contents repaired.

---

`configurations_print` *Print configurations as a data frame*

---

**Description**

Print configurations as a data frame

**Usage**

```
configurations_print(configurations, metadata = FALSE)
```

**Arguments**

<code>configurations</code>	<code>data.frame</code> Parameter configurations of the target algorithm (one per row).
<code>metadata</code>	<code>logical(1)</code> whether to print the metadata or not. The metadata are data for the configurations (additionally to the value of each parameter) used by <b>irace</b> .

**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	ParameterSpace Data structure containing the parameter space definition. The data structure has to similar to the one returned by the function <a href="#">readParameters</a> .

**Value**

None.

**Author(s)**

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

**See Also**

[configurations\\_print\(\)](#) to print the configurations as a data frame.

---

defaultScenario	<i>Default scenario settings</i>
-----------------	----------------------------------

---

**Description**

Return scenario object with default values.

**Usage**

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

**Arguments**

scenario	list() Data structure containing <b>irace</b> settings. The data structure has to be the one returned by the function <a href="#">defaultScenario()</a> or <a href="#">readScenario()</a> .
params_def	data.frame() Definition of the options accepted by the scenario. This should only be modified by packages that wish to extend <b>irace</b> .

**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 (1, 2 or 3) 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 Perform a postselection race after the execution of irace to consume all remaining budget. Value 0 disables the postselection race. (Default: `1`)

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. (Default: `1e-04`)

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"`)

parameters Parameters space object (usually read from a file using readParameters). (Default: `""`)

- 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: `""`)
- `targetCmdline` Command-line arguments provided to `targetRunner` (or `targetRunnerLauncher` if defined). The substrings `\{configurationID\}`, `\{instanceID\}`, `\{seed\}`, `\{instance\}`, and `\{bound\}` will be replaced by their corresponding values. The substring `\{targetRunnerArgs\}` will be replaced by the concatenation of the switch and value of all active parameters of the particular configuration being evaluated. The substring `\{targetRunner\}`, if present, will be replaced by the value of `targetRunner` (useful when using `targetRunnerLauncher`). (Default: `"{configurationID} {instanceID} {seed} {instance} {bound} {targetRunnerArgs}"`)
- `targetRunnerRetries` Number of times to retry a call to `targetRunner` if the call failed. (Default: `0`)
- `targetRunnerTimeout` Timeout in seconds of any `targetRunner` call (only applies to `target-runner` executables not to R functions), ignored if 0. (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: `""`)

`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: `""`)

`blockSize` Number of training instances, that make up a 'block' in `trainInstancesFile`. Elimination of configurations will only be performed after evaluating a complete block and never in the middle of a block. Each block typically contains one instance from each instance class (type or family) and the block size is the number of classes. The value of `blockSize` will multiply `firstTest`, `eachTest` and `elitistNewInstances`. (Default: 1)

- 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)

`minExperiments` Minimum number of runs (invocations of `targetRunner`) that will be performed. It determines the minimum budget of experiments for the tuning. The actual budget depends on the number of parameters and `minNbSurvival`. (Default: NA)

`maxTime` Maximum total execution time for the executions of `targetRunner`. `targetRunner` must return two values: cost and time. This value and the one returned by `targetRunner` must use the same units (seconds, minutes, iterations, evaluations, ...). (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: 0.05)

`minMeasurableTime` Minimum time unit that is still (significantly) measurable. (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. Capping is enabled by default if `elitist` is active, `maxTime > 0` and `boundMax > 0`. (Default: NA)

`cappingAfterFirstTest` If set to 1, elimination due to capping only happens after `firstTest` instances are seen. (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 multiplication constant (PARX) 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) Object created by <b>irace</b> and typically saved in the log file <code>irace.Rdata</code> . If a character string is given, then it is interpreted as the path to the log file from which the <code>iraceResults</code> object will be loaded.
ids	integer() The id or a vector of ids of the candidates configurations to obtain.
drop.metadata	logical(1) Remove metadata, such as the configuration ID and the ID of the parent, from the returned configurations. See <a href="#">removeConfigurationsMetaData()</a> .

### Value

A data frame containing the elite configurations required, in the order and with the repetitions given by `ids`.

### Author(s)

Manuel López-Ibáñez and Leslie Pérez Cáceres

### Examples

```
log_file <- system.file("exdata/irace-acotsp.Rdata", package="irace", mustWork=TRUE)
getConfigurationById(log_file, ids = c(2,1), drop.metadata = TRUE)
```

---

`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

<code>iraceResults</code>	<code>list()</code>   <code>character(1)</code> Object created by <b>irace</b> and typically saved in the log file <code>irace.Rdata</code> . If a character string is given, then it is interpreted as the path to the log file from which the <code>iraceResults</code> object will be loaded.
<code>iterations</code>	<code>integer()</code> The iteration number or a vector of iteration numbers from where the configurations should be obtained. Negative values start counting from the last iteration.
<code>drop.metadata</code>	<code>logical(1)</code> Remove metadata, such as the configuration ID and the ID of the parent, from the returned configurations. See <a href="#">removeConfigurationsMetaData()</a> .

## 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)
getConfigurationByIteration(log_file, iterations = c(-2, -1), drop.metadata = TRUE)
```

---

getFinalElites	<i>Return the elite configurations of the final iteration.</i>
----------------	--

---

### Description

Return the elite configurations of the final iteration.

### Usage

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

### Arguments

iraceResults	list() character(1) Object created by <b>irace</b> and typically saved in the log file <code>irace.Rdata</code> . If a character string is given, then it is interpreted as the path to the log file from which the <code>iraceResults</code> object will be loaded.
n	integer(1) 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	logical(1) Remove metadata, such as the configuration ID and the ID of the parent, from the returned configurations. See <a href="#">removeConfigurationsMetaData()</a> .

### 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)))
```

---

`get_instanceID_seed_pairs`

*Returns the pairs of instance IDs and seeds used as instances in the race (and optionally the actual instances).*

---

### Description

Returns the pairs of instance IDs and seeds used as instances in the race (and optionally the actual instances).

### Usage

```
get_instanceID_seed_pairs(iraceResults, index, instances = FALSE)
```

### Arguments

<code>iraceResults</code>	<code>list()</code>   <code>character(1)</code> Object created by <b>irace</b> and typically saved in the log file <code>irace.Rdata</code> . If a character string is given, then it is interpreted as the path to the log file from which the <code>iraceResults</code> object will be loaded.
<code>index</code>	<code>integer()</code> Indexes of the (instanceID,seed) pairs to be returned. The default returns everything.
<code>instances</code>	<code>logical(1)</code> Whether to add the actual instances as an additional column (only if the instances are of atomic type).

### Value

`data.table()`  
With default arguments, a `data.table` containing two columns "instanceID" and "seed". With `instances=TRUE` and if the instances are of atomic type (see `is.atomic()`) type, another column `instance` is added that contains the actual instance.

### Author(s)

Manuel López-Ibáñez

### Examples

```
log_file <- system.file("exdata/irace-acotsp.Rdata", package="irace", mustWork=TRUE)
head(get_instanceID_seed_pairs(log_file))
# Add the instance names
get_instanceID_seed_pairs(log_file, index=1:10, instances=TRUE)
```

---

has_testing_data	<i>Check if the results object generated by irace has data about the testing phase.</i>
------------------	---

---

### Description

Check if the results object generated by irace has data about the testing phase.

### Usage

```
has_testing_data(iraceResults)
```

### Arguments

iraceResults	list() character(1) Object created by <b>irace</b> 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.
--------------	--

### Value

logical(1)

### Examples

```
irace_results <- read_logfile(system.file("exdata/irace-acotsp.Rdata", package="irace",
                                         mustWork=TRUE))
print(has_testing_data(irace_results))
```

---

irace	<i>Execute one run of the Iterated Racing algorithm.</i>
-------	--

---

### Description

The function `irace` implements the Iterated Racing procedure for parameter tuning. It receives a configuration scenario and a parameter space to be tuned, and returns the best configurations found, namely, the elite configurations obtained from the last iterations. As a first step, it checks the correctness of `scenario` using `checkScenario()` and recovers a previous execution if `scenario$recoveryFile` is set. A R data file log of the execution is created in `scenario$logFile`.

### Usage

```
irace(scenario)
```

## Arguments

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

## Details

The execution of this function is reproducible under some conditions. See the FAQ section in the [User Guide](#).

## 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. The element `scenario$parameters` contains the parameters R object that describes 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.`).

`experimen_log` A `data.table` 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 `race_state$instances_log` 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` An environment 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:
# In general, there are three steps:
scenario <- readScenario(filename = "scenario.txt")
irace(scenario = scenario)

## End(Not run)
#####
# 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:
#  $f(x) = \lambda * f_{\text{rastrigin}}(x) + (1 - \lambda) * f_{\text{rosenbrock}}(x)$ 
# 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[1L:(d - 1L)]
  tz <- z[2L:d]
  sum(100 * (hz^2 - tz)^2 + (hz - 1)^2)
}
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))
  }
  # For reproducible results, we should use the random seed given by
  # experiment$seed to set the random seed of the target algorithm.
  res <- withr::with_seed(experiment$seed,
    stats::optim(par,fn, method="SANN",
      control=list(maxit=1000
        , tmax = as.numeric(configuration[["tmax"]])
        , temp = as.numeric(configuration[["temp"]])
      )))

  ## 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 = "",
parameters = parameters)

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

## 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)

## 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.
test_index <- 11:20
test_seeds <- sample.int(2147483647L, size = length(test_index), replace = TRUE)
test <- function(configuration)
{
  res <- lapply(seq_along(test_index),
               function(x) target_runner(
                 experiment = list(instance = weights[test_index[x]],
                                   seed = test_seeds[x],
                                   configuration = configuration),
                 scenario = scenario))
  return (sapply(res, getElement, name = "cost"))
}
## To do so, first we apply the default configuration of the SANN
## algorithm to these instances:
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))

```

## 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))
```

```
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:

<code>-h,--help</code>	Show this help.
<code>-v,--version</code>	Show irace package version.
<code>-c,--check</code>	Check scenario.
<code>-i,--init</code>	Initialize the working directory with template config files.
<code>--only-test</code>	Only test the configurations given in the file passed as argument.
<code>-s,--scenario</code>	File that describes the configuration scenario setup and other irace settings. Default: <code>./scenario.txt</code> .
<code>--exec-dir</code>	Directory where the programs will be run. Default: <code>./</code> .
<code>-p,--parameter-file</code>	File that contains the description of the parameters of the target algorithm. Default: <code>./parameters.txt</code> .
<code>--configurations-file</code>	File that contains a table of initial configurations. If empty or <code>`NULL`</code> , all initial configurations are randomly generated.
<code>-l,--log-file</code>	File to save tuning results as an R dataset, either absolute path or relative to <code>execDir</code> . Default: <code>./irace.Rdata</code> .
<code>--recovery-file</code>	Previously saved log file to recover the execution of <code>`irace`</code> , either absolute path or relative to the current directory. If empty or <code>`NULL`</code> , recovery is not performed.
<code>--train-instances-dir</code>	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.

`--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.

`--block-size` Number of training instances, that make up a 'block' in ``trainInstancesFile``. Elimination of configurations will only be performed after evaluating a complete block and never in the middle of a block. Each block typically contains one instance from each instance class (type or family) and the block size is the number of classes. The value of ``blockSize`` will multiply ``firstTest``, ``eachTest`` and ``elitistNewInstances``. Default: 1.

`--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-cmdline` Command-line arguments provided to ``targetRunner`` (or ``targetRunnerLauncher`` if defined). The substrings ``\{configurationID\}``, ``\{instanceID\}``, ``\{seed\}``,

``\{instance\}``, and ``\{bound\}`` will be replaced by their corresponding values. The substring ``\{targetRunnerArgs\}`` will be replaced by the concatenation of the switch and value of all active parameters of the particular configuration being evaluated. The substring ``\{targetRunner\}``, if present, will be replaced by the value of ``targetRunner`` (useful when using ``targetRunnerLauncher``). Default: `{configurationID}{instanceID} {seed} {instance} {bound} {targetRunnerArgs}`.

`--target-runner-retries` Number of times to retry a call to ``targetRunner`` if the call failed. Default: 0.

`--target-runner-timeout` Timeout in seconds of any ``targetRunner`` call (only applies to ``target-runner`` executables not to R functions), ignored if 0. 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.

`--min-experiments` Minimum number of runs (invocations of ``targetRunner``) that will be performed. It determines the minimum budget of experiments for the tuning. The actual budget depends on the number of parameters and ``minNbSurvival``.

`--max-time` Maximum total execution time for the executions of ``targetRunner``. ``targetRunner`` must return two values: cost and time. This value and the one returned by ``targetRunner`` must use the same units (seconds, minutes, iterations, evaluations, ...). 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: 0.05.

`--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.

<code>--mpi</code>	Enable/disable MPI. Use <code>`Rmpi`</code> to execute <code>`targetRunner`</code> in parallel (parameter <code>`parallel`</code> is the number of slaves). Default: 0.
<code>--batchmode</code>	Specify how irace waits for jobs to finish when <code>`targetRunner`</code> submits jobs to a batch cluster: sge, pbs, torque, slurm or htcondor. <code>`targetRunner`</code> must submit jobs to the cluster using, for example, <code>`qsub`</code> . Default: 0.
<code>-q,--quiet</code>	Reduce the output generated by irace to a minimum. Default: 0.
<code>--debug-level</code>	Debug level of the output of <code>`irace`</code> . Set this to 0 to silence all debug messages. Higher values (1, 2 or 3) provide more verbose debug messages. Default: 0.
<code>--seed</code>	Seed of the random number generator (by default, generate a random seed).
<code>--soft-restart</code>	Enable/disable the soft restart strategy that avoids premature convergence of the probabilistic model. Default: 1.
<code>--soft-restart-threshold</code>	Soft restart threshold value for numerical parameters. Default: 1e-04.
<code>-e,--elitist</code>	Enable/disable elitist irace. Default: 1.
<code>--elitist-new-instances</code>	Number of instances added to the execution list before previous instances in elitist irace. Default: 1.
<code>--elitist-limit</code>	In elitist irace, maximum number per race of elimination tests that do not eliminate a configuration. Use 0 for no limit. Default: 2.
<code>--capping</code>	Enable the use of adaptive capping, a technique designed for minimizing the computation time of configurations. Capping is enabled by default if <code>`elitist`</code> is active, <code>`maxTime &gt; 0`</code> and <code>`boundMax &gt; 0`</code> .
<code>--capping-after-first-test</code>	If set to 1, elimination due to capping only happens after <code>`firstTest`</code> instances are seen. Default: 0.
<code>--capping-type</code>	Measure used to obtain the execution bound from the performance of the elite configurations: median, mean, worst, best. Default: median.
<code>--bound-type</code>	Method to calculate the mean performance of elite configurations: candidate or instance. Default: candidate.
<code>--bound-max</code>	Maximum execution bound for <code>`targetRunner`</code> . It must be specified when capping is enabled. Default: 0.
<code>--bound-digits</code>	Precision used for calculating the execution time. It must be specified when capping is enabled. Default: 0.
<code>--bound-par</code>	Penalization multiplication constant (PARX) for timed out executions (executions that reach <code>`boundMax`</code> execution time). Default: 1.
<code>--bound-as-timeout</code>	Replace the configuration cost of bounded executions

	with <code>`boundMax`</code> . Default: 1.
<code>--postselection</code>	Perform a postselection race after the execution of irace to consume all remaining budget. Value 0 disables the postselection race. Default: 1.
<code>--aclib</code>	Enable/disable AClib mode. This option enables compatibility with GenericWrapper4AC as targetRunner script. Default: 0.
<code>--iterations</code>	Maximum number of iterations. Default: 0.
<code>--experiments-per-iteration</code>	Number of runs of the target algorithm per iteration. Default: 0.
<code>--min-survival</code>	Minimum number of configurations needed to continue the execution of each race (iteration). Default: 0.
<code>--num-configurations</code>	Number of configurations to be sampled and evaluated at each iteration. Default: 0.
<code>--mu</code>	Parameter used to define the number of configurations sampled and evaluated at each iteration. Default: 5.
<code>--confidence</code>	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. The element `scenario$parameters` contains the parameters R object that describes 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.`).

**experimen\_log** A `data.table` 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 `race_state$instances_log` 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** An environment 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	<i>Higher-level interface to launch irace.</i>
------------	--

---

### Description

Higher-level interface to launch irace.

### Usage

```
irace_main(scenario, output.width = 9999L)
```

### Arguments

scenario	list() Data structure containing <b>irace</b> settings. The data structure has to be the one returned by the function <code>defaultScenario()</code> or <code>readScenario()</code> .
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. The element `scenario$parameters` contains the parameters R object that describes 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.`).
- `experimen_log` A `data.table` 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 `race_state$instances_log` 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` An environment 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 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**.

---

irace\_summarise

*Summarise the results of a run of irace*

---

**Description**

Summarise the results of a run of irace

**Usage**

```
irace_summarise(iraceResults)
```

**Arguments**

`iraceResults` `list()`|`character(1)`  
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.

**Value**

`list()`

**Author(s)**

Manuel López-Ibáñez

**Examples**

```
irace_results <- read_logfile(system.file("exdata/irace-acotsp.Rdata",  
                                         package="irace", mustWork=TRUE))  
irace_summarise(irace_results)
```

---

<code>irace_version</code>	<i>A character string containing the version of irace including git SHA.</i>
----------------------------	--

---

**Description**

A character string containing the version of `irace` including git SHA.

**Usage**

```
irace_version
```

**Format**

An object of class `character` of length 1.

---

multi_irace	<i>Execute <code>irace()</code> multiple times with the same or different scenarios and parameter space definitions.</i>
-------------	--

---

### Description

There are three modes of operation:

- One scenarios and k parameters: k runs with the same scenario and each parameter space definition.
- One parameters and k scenarios: k runs with the same parameter space definition and each scenario.
- k parameters and k scenarios: k runs with each scenario and parameter space definition.

Each of the k runs can be repeated n times by supplying a value for n.

### Usage

```
multi_irace(
  scenarios,
  parameters,
  n = 1L,
  parallel = 1L,
  split_output = parallel > 1L,
  global_seed = NULL
)
```

### Arguments

scenarios	list() A list of scenarios. If only a single scenario is supplied, it is used for all parameters.
parameters	list() A list of parameter space definitions. If only a single definition is supplied, it is used for all scenarios.
n	integer(1) The number of repetitions.
parallel	integer(1) The number of workers to use. A value of 1 means sequential execution. Note that parallel > 1 is not supported on Windows.
split_output	logical(1) If TRUE, the output of <code>irace()</code> is written to <code>{execDir}/run_{i}/irace.out</code> instead of the standard output.
global_seed	integer(1) The global seed used to seed the individual runs.

**Value**

A list of the outputs of `irace()`.

**See Also**

`irace()` the main interface for single irace runs.

---

parameters

*Create a parameter space to be tuned.*

---

**Description**

- `param_cat()` creates a categorical parameter.
- `param_ord()` creates an ordinal parameter.
- `param_real()` creates a real-valued parameter.
- `param_int()` creates an integer parameter.

**Usage**

```
parametersNew(..., forbidden = NULL, debugLevel = 0L)
```

```
param_cat(name = name, values, label = "", condition = TRUE)
```

```
param_ord(name, values, label = "", condition = TRUE)
```

```
param_real(
  name,
  lower,
  upper,
  label = "",
  condition = TRUE,
  transf = "",
  digits = 15L
)
```

```
param_int(name, lower, upper, label = "", condition = TRUE, transf = "")
```

**Arguments**

- |                         |   |
|-------------------------|---|
| <code>...</code>        | one or more parameters created by <code>param_int()</code> , <code>param_real()</code> , <code>param_ord()</code> , or <code>param_cat()</code> . |
| <code>forbidden</code>  | <code>expression()</code>   <code>character(1)</code><br>String or list of expressions that define forbidden parameter configurations.            |
| <code>debugLevel</code> | <code>integer(1)</code><br>Larger values produce more verbose output.   |

name	character(1) Parameter name (must be alphanumeric).
values	character() Domain as a vector of strings.
label	character(1) Label associated to the parameter. Often used to encode a command-line switch that activates the parameter.
condition	expression(1) character(1) Expression that defines when the parameter is active according to the value of other parameters.
lower, upper	Lower and upper limits of the valid domain.
transf	character(1) If "log", then the parameter is sampled in a logarithmic scale.
digits	integer(1) The number of decimal places to be considered for real-valued parameters.

## Value

ParameterSpace

## Examples

```
digits <- 4L
parametersNew(
  param_cat(name = "algorithm", values = c("as", "mmas", "eas", "ras", "acs"), label = "--"),
  param_ord(name = "localsearch", values = c("0", "1", "2", "3"), label = "--localsearch "),
  param_real(name = "alpha", lower = 0.0, upper=5.0, label = "--alpha ", digits = digits),
  param_real(name = "beta", lower = 0.0, upper = 10.0, label = "--beta ", digits = digits),
  param_real(name = "rho", lower = 0.01, upper = 1.00, label = "--rho ", digits = digits),
  param_int(name = "ants", lower = 5, upper = 100, transf = "log", label = "--ants "),
  param_real(name = "q0", label = "--q0 ", lower=0.0, upper=1.0,
    condition = expression(algorithm == "acs")),
  param_int(name = "rasrank", label = "--rasranks ", lower=1, upper=quote(min(ants, 10)),
    condition = 'algorithm == "ras"'),
  param_int(name = "elitistants", label = "--elitistants ", lower=1, upper=expression(ants),
    condition = 'algorithm == "eas"'),
  param_int(name = "nnls", label = "--nnls ", lower = 5, upper = 50,
    condition = expression(localsearch %in% c(1,2,3))),
  param_cat(name = "dlb", label = "--dlb ", values = c(0,1),
    condition = "localsearch %in% c(1,2,3)"),
  forbidden = "(alpha == 0) & (beta == 0)")
```

---

path_rel2abs	<i>Converts a relative path to an absolute path.</i>
--------------	--

---

**Description**

If the path passed corresponds to an executable, it tries to find its path using `Sys.which()`. Expansion of '~' in Windows follows the definition of `fs::path_expand()` rather than `base::path.expand()`. This function 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	<i>Create plot from an ablation log</i>
--------------	---

---

**Description**

Create plot from an ablation log

**Usage**

```
plotAblation(
  ablog,
  pdf_file = NULL,
  width = 20,
  height = 7,
  type = c("mean", "boxplot", "rank"),
  n = 0L,
  mar = NULL,
  ylab = "Mean configuration cost",
  ylim = NULL,
  rename_labels = NULL,
  ...
)
```

**Arguments**

ablog	(list() character(1)) Ablation log object returned by <code>ablation()</code> . Alternatively, the path to an .Rdata file, e.g., "log-ablation.Rdata", from which the object will be loaded.
pdf_file	Output filename.
width	Width provided to create the PDF file.
height	Height provided to create the PDF file.
type	Type of plot. Supported values are "mean" and "boxplot". Adding "rank" will plot rank per instance instead of raw cost value.
n	integer(1) Number of parameters included in the plot. By default all parameters are included.
mar	Vector with the margins for the ablation plot.
ylab	Label of y-axis.
ylim	Numeric vector of length 2 giving the y-axis range.
rename_labels	character() Renaming table for nicer labels. For example, c("No value"="NA", "LongParameterName"="LPN").
...	Further graphical parameters may also be supplied as arguments. See <code>graphics::plot.default()</code> .

**Author(s)**

Leslie Pérez Cáceres and Manuel López-Ibáñez

**See Also**

`ablation()` `ablation_cmdline()`

**Examples**

```
logfile <- file.path(system.file(package="irace"), "exdata", "log-ablation.Rdata")
plotAblation(ablog = logfile)
plotAblation(ablog = logfile, type = "mean")
plotAblation(ablog = logfile, type = c("rank", "boxplot"), rename_labels = c(
  "localsearch"="ls", algorithm="algo", source="default"))
```

---

printParameters

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

---

**Description**

Print parameter space in the textual format accepted by irace.

**Usage**

```
printParameters(parameters)
```

**Arguments**

parameters ParameterSpace  
 Data structure containing the parameter space definition. The data structure has to similar to the one returned by the function [readParameters](#).

**Value**

character()

**See Also**

[readParameters\(\)](#)

**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)
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"
q0dep       "--q0 "          r   (0.0, q0)       | 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)

[forbidden]
(alpha == 0.0) & (beta == 0.0)
'

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

*Post-selection race*


---

**Description**

psRace performs a post-selection race of a set of configurations.

**Usage**

```
psRace(
  iraceResults,
  max_experiments,
  conf_ids = NULL,
  iteration_elites = FALSE,
  psrace_logFile = NULL
)
```

**Arguments**

iraceResults    list()|character(1)  
 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.

max\_experiments  
 numeric(1)  
 Number of experiments for the post-selection race. If it is equal to or smaller than 1, then it is a fraction of the total budget given by `iraceResults$scenario$maxExperiments` or `iraceResults$scenario$maxTime / iraceResults$state$boundEstimate`.

conf\_ids        IDs of the configurations in `iraceResults$allConfigurations` to be used for the post-selection. If NULL, then the configurations are automatically selected.

`iteration_elites` If FALSE, give priority to selecting the configurations that were elite in the last iteration. If TRUE, select from all elite configurations of all iterations. This parameter only has an effect when `conf_ids` is not NULL.

`psrace_logFile` character(1)  
Log file to save the post-selection race log. If NULL, the log is saved in `iraceResults$scenario$logFile`

### Value

The elite configurations after the post-selection. In addition, if `iraceResults$scenario$logFile` is defined, it saves an updated copy of `iraceResults` in that file, where `iraceResults$psrace_log` is a list with the following elements:

**configurations** Configurations used in the post-selection race.

**instances** Data frame with the instances used in the post-selection race. First column has the instances IDs from `iraceResults$scenario$instances`, second column the seed assigned to the instance.

**max\_experiments** Configuration budget assigned to the post-selection race.

**experiments** Matrix of results generated by the post-selection race, in the same format as the matrix `iraceResults$experiments`. Column names are the configuration IDs and row names are the instance IDs.

**elites** Best configurations found in the experiments.

### Author(s)

Leslie Pérez Cáceres and Manuel López-Ibáñez

### Examples

```
irace_log <- read_logfile(system.file(package="irace", "exdata", "sann.rda"))
# Use a temporary file to not change the original "sann.rda".
psrace_logFile <- withr::local_tempfile(fileext = ".Rdata")
# Execute the post-selection after the execution of irace. Use 10% of the total budget.
psRace(irace_log, max_experiments=0.1, psrace_logFile = psrace_logFile)
# Print psrace_log
irace_log <- read_logfile(psrace_logFile)
str(irace_log$psrace_log)
```

---

random\_seed

*Get, set and restore the state of the random number generator state.*

---

### Description

Get, set and restore the state of the random number generator state.

**Usage**

```

get_random_seed()

set_random_seed(seed)

restore_random_seed(seed)

```

**Arguments**

```

seed          (list())integer(1)
              Either an integer or the list returned by get_random_seed().

```

**Details**

These functions originate from the `withr` package.

**Value**

`get_random_seed()` returns a list with two components `random_seed` and `rng_kind` or `NULL` if no seed was set; `set_random_seed()` and `restore_random_seed()` do not return anything.

**Examples**

```

old_seed <- get_random_seed()
on.exit(restore_random_seed(old_seed))
set_random_seed(42)
value1 <- runif(1)
set_random_seed(42)
value2 <- runif(1)
stopifnot(all.equal(value1, value2))

```

---

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 = 0L, text)

```

**Arguments**

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

**Details**

Example of an input file:

```
# This is a comment line
param_1  param_2
  0.5    "value_1"
  1.0           NA
  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	<i>Reads the parameters to be tuned by <b>irace</b> from a file or from a character string.</i>
----------------	---

---

### Description

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

### Usage

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

### Arguments

file	character(1) Filename containing the definitions of the parameters to be tuned.
digits	integer(1) The number of decimal places to be considered for real-valued 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.

The order of the parameters determines the order in which parameters are given to `targetRunner`. Changing the order may also change the results produced by **irace**, even with the same random seed.

### 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.

is\_dependent Logical vector that specifies which parameter has a dependent domain.

digits Integer vector that specifies the number of digits per parameter.

forbidden List of expressions that define which parameter configurations are forbidden.

### 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 " o   (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)

[forbidden]
(alpha == 0.0) & (beta == 0.0)
[global]
digits = 4
'
parameters <- readParameters(text=parameters_table)
str(parameters)
```

---

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

---

### Description

The scenario argument is an initial scenario that is overwritten for every setting specified in the file to be read.

### 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 <code>system.file(package="irace", "templates/scenario.txt.tpl")</code> .
scenario	list() Data structure containing <b>irace</b> settings. The data structure has to be the one returned by the function <code>defaultScenario()</code> or <code>readScenario()</code> .
params_def	data.frame() Definition of the options accepted by the scenario. This should only be modified by packages that wish to extend <b>irace</b> .

### 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_pcs_file	<i>Read parameters in PCS (AClib) format and write them in irace format.</i>
---------------	--

---

### Description

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

### Usage

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

### Arguments

file	character(1) Filename containing the definitions of the parameters to be tuned.
digits	integer(1) The number of decimal places to be considered for real-valued 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:** Multiple conditions and default configuration 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.

**See Also**[readParameters\(\)](#)**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        [1, 100][10]il
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}
{alpha=0, beta=0}'
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 data frame containing configurations. Currently, metadata corresponds to column names starting with a period. This function should be used before printing the configurations to output only the values for the parameters of the configuration without metadata possibly useless to the user.

**Usage**

```
removeConfigurationsMetaData(configurations)
```

**Arguments**

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

**Value**

The same data frame without "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.

---

save\_irace\_logfile     *Save the log generated by **irace** to a file (by default irace.Rdata).*

---

**Description**

This function may be useful if you are manually editing the log data generated by a run of **irace**.

**Usage**

```
save_irace_logfile(iraceResults, logfile)
```

**Arguments**

iraceResults     `list()`  
Object created by **irace** and typically saved in the log file `irace.Rdata`.

logfile     `character(1)`  
Filename to save `iraceResults`. Usually, this is given by `scenario$logFile`.  
If `NULL` or `""`, no data is saved.

**See Also**

[read\\_logfile\(\)](#)

---

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)
```

### Arguments

scenario	list() Data structure containing <b>irace</b> settings. The data structure has to be the one returned by the function <code>defaultScenario()</code> or <code>readScenario()</code> .
from	character(1) Character string containing a regular expression (or character string for <code>fixed = TRUE</code> ) to be matched.
to	character(1) The replacement string.character string. For <code>fixed = FALSE</code> this can include backreferences "\1" to "\9" to parenthesized subexpressions of <code>from</code> .
fixed	logical(1) If <code>TRUE</code> , <code>from</code> 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: <ul style="list-style-type: none"> <li>id_configuration An alphanumeric string that uniquely identifies a configuration;</li> <li>id_instance An alphanumeric string that uniquely identifies an instance;</li> <li>seed Seed for the random number generator to be used for this evaluation, ignore the seed for deterministic algorithms;</li> <li>instance String giving the instance to be used for this evaluation;</li> <li>bound (only when capping is enabled) Time bound for the execution;</li> <li>configuration 1-row data frame with a column per parameter name;</li> </ul>
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 <b>irace</b> settings. The data structure has to be the one returned by the function <code>defaultScenario()</code> or <code>readScenario()</code> .
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**

<code>experiment</code>	A list describing the experiment. It contains at least: <ul style="list-style-type: none"> <li><code>id_configuration</code> An alphanumeric string that uniquely identifies a configuration;</li> <li><code>id_instance</code> An alphanumeric string that uniquely identifies an instance;</li> <li><code>seed</code> Seed for the random number generator to be used for this evaluation, ignore the seed for deterministic algorithms;</li> <li><code>instance</code> String giving the instance to be used for this evaluation;</li> <li><code>bound</code> (only when capping is enabled) Time bound for the execution;</li> <li><code>configuration</code> 1-row data frame with a column per parameter name;</li> </ul>
<code>scenario</code>	<code>list()</code> Data structure containing <b>irace</b> settings. The data structure has to be the one returned by the function <code>defaultScenario()</code> or <code>readScenario()</code> .

**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	<i>Execute the given configurations on the testing instances specified in the scenario</i>
--------------------	--

---

**Description**

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

**Usage**

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

**Arguments**

<code>configurations</code>	<code>data.frame</code> Parameter configurations of the target algorithm (one per row).
<code>scenario</code>	<code>list()</code> Data structure containing <b>irace</b> settings. The data structure has to be the one returned by the function <code>defaultScenario()</code> or <code>readScenario()</code> .

**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	<i>Test configurations given an explicit table of configurations and a scenario file</i>
------------------	--

---

**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	character(1) Path to a file containing configurations: one configuration per line, one parameter per column, parameter names in header.
scenario	list() Data structure containing <b>irace</b> settings. The data structure has to be the one returned by the function <a href="#">defaultScenario()</a> or <a href="#">readScenario()</a> .

**Value**

iraceResults

**Author(s)**

Manuel López-Ibáñez

**See Also**

[testing\\_fromlog\(\)](#) provides a different interface for testing.

---

testing_fromlog	<i>Test configurations given in the logfile (typically irace.Rdata) produced by <b>irace</b>.</i>
-----------------	---

---

### 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	character(1) Path to the logfile (typically irace.Rdata) produced by <b>irace</b> .
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**

logical(1)  
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

- \* **ablation**
  - ablation, 4
  - ablation\_cmdline, 6
  - plotAblation, 39
  - read\_ablogfile, 49
- \* **analysis**
  - get\_instanceID\_seed\_pairs, 21
  - getConfigurationById, 18
  - getConfigurationByIteration, 19
  - getFinalElites, 20
  - irace\_summarise, 34
  - read\_logfile, 49
  - save\_irace\_logfile, 52
- \* **automatic**
  - irace-package, 3
- \* **configuration**
  - irace-package, 3
- \* **datasets**
  - irace\_license, 32
  - irace\_version, 35
- \* **optimize**
  - irace-package, 3
- \* **package**
  - irace-package, 3
- \* **running**
  - irace, 22
  - irace\_cmdline, 26
  - irace\_main, 33
  - multi\_irace, 36
  - testing\_fromfile, 57
  - testing\_fromlog, 58
- \* **tuning**
  - irace-package, 3
- ablation, 4
- ablation(), 6, 7, 40, 49
- ablation\_cmdline, 6
- ablation\_cmdline(), 5, 40
- base::grep(), 53
- base::path.expand(), 39
- buildCommandLine, 8
- check\_output\_target\_runner, 11
- checkIraceScenario, 9
- checkParameters, 10
- checkScenario, 9, 10
- checkScenario(), 11, 17, 22, 24, 42, 48
- configurations\_print, 12
- configurations\_print(), 13, 52
- configurations\_print\_command, 12
- configurations\_print\_command(), 12, 52
- defaultScenario, 9, 13, 23, 31, 33
- defaultScenario(), 9–11, 13, 17, 23, 24, 33, 34, 42, 48, 53–57, 59
- fs::path\_expand(), 39
- get\_instanceID\_seed\_pairs, 21
- get\_random\_seed(random\_seed), 43
- get\_random\_seed(), 44
- getConfigurationById, 18
- getConfigurationByIteration, 19
- getFinalElites, 20
- graphics::plot.default(), 40
- has\_testing\_data, 22
- irace, 22
- irace(), 3, 24, 33, 34, 36, 37
- irace-package, 3
- irace.cmdline(irace\_cmdline), 26
- irace\_cmdline, 26
- irace\_cmdline(), 24, 27, 34
- irace\_license, 32
- irace\_main, 33
- irace\_main(), 24, 27, 32
- irace\_summarise, 34
- irace\_version, 35
- is.atomic(), 21

multi\_irace, 36

param\_cat (parameters), 37  
param\_int (parameters), 37  
param\_ord (parameters), 37  
param\_real (parameters), 37  
parameters, 37  
parametersNew (parameters), 37  
path\_rel2abs, 39  
plotAblation, 39  
plotAblation(), 5–7  
printParameters, 40  
printScenario, 9, 41  
printScenario(), 11, 17, 42, 48  
psRace, 42  
psRace(), 33

random\_seed, 43  
read\_ablogfile, 49  
read\_logfile, 49  
read\_logfile(), 52  
read\_pcs\_file, 50  
readConfigurationsFile, 44  
readConfigurationsFile(), 4, 57  
readParameters, 10, 13, 23, 31, 33, 41, 45, 46  
readParameters(), 24, 34, 41, 45, 51  
readScenario, 9, 48  
readScenario(), 9–11, 13, 17, 23, 24, 27, 33,  
34, 42, 48, 53–57  
removeConfigurationsMetaData, 51  
removeConfigurationsMetaData(), 18–20  
restore\_random\_seed (random\_seed), 43  
restore\_random\_seed(), 44

save\_irace\_logfile, 52  
scenario\_update\_paths, 53  
set\_random\_seed (random\_seed), 43  
set\_random\_seed(), 44  
Sys.which(), 39

target\_evaluator\_default, 54  
target\_runner\_default, 55  
testConfigurations, 56  
testing\_fromfile, 57  
testing\_fromfile(), 59  
testing\_fromlog, 58  
testing\_fromlog(), 57

withr, 44