Package 'irace'

February 20, 2015

Type Package

Title Iterated Racing Procedures

Version 1.06

Date 2014-11-26

Maintainer Manuel López-Ibáñez <manuel.lopez-ibanez@ulb.ac.be>

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

Description Iterated racing for automatic algorithm configuration

Depends R (>= 2.14.0)

Suggests Rmpi (>= 0.6.0), parallel

License GPL (>= 2)

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

ByteCompile yes

LazyData true

Encoding UTF-8

NeedsCompilation no

Repository CRAN

Date/Publication 2014-11-26 18:33:46

R topics documented:

`irace-package` *The irace package*

Description

Iterated racing for Automatic Algorithm Configuration

Details

Package:	irace
Type:	Package
Version:	1.06
Date:	2014-11-26
License:	GPL (>= 2)
LazyLoad:	yes

Author(s)

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

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

References

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

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

Examples

```
##########################################################################
```

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

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

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


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

## Next, we define the function that will evaluate each candidate
## configuration on a single instance. For simplicity, we restrict to
## three-dimensional functions and we set the maximum number of
## iterations of SANN to 5000.
hook.run <- function(instance, candidate, extra.params = NULL, config = list())
{
  D <- 3
  par <- runif(D, min=-1, max=1)
  fn <- function(x) {
    weight <- instance
    return(weight * f_rastrigin(x) + (1 - weight) * f_rosenbrock(x))
  }
  res <- optim(par,fn, method="SANN",
               control=list(maxit=5000
                 , tmax = as.numeric(candidate$values[["tmax"]])
```

```
                , temp = as.numeric(candidate$values[["temp"]])
                ))
  return(res$value)
}

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

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

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

## We extract and apply the winning configuration found by irace
## to these instances:
result.list <- as.list(removeCandidatesMetaData(result[1,]))
tuned <- sapply(weights[101:200], hook.run, candidate=list(values=result.list))

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

## End(Not run)
```

`buildCommandLine` *Build a Command Line*

Description

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

Usage

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

Arguments

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

Details

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

Value

A character chain containing the switches and the values.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

Examples

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

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

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

## End(Not run)
```

---

candidates.print *candidates.print*

Description

Print candidate configurations (hereafter "candidates").

Usage

```
candidates.print(cand, metadata = FALSE)
```

Arguments

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

Value

None.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`candidates.print.command` to print the candidates as command lines.

`candidates.print.command`
candidates.print.command

Description

Print candidate configurations (hereafter "candidates") as command lines.

Usage

```
candidates.print.command(cand, parameters)
```

Arguments

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

Value

None.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

candidates.print to print the candidates as a data frame.

---

checkConfiguration *checkConfiguration*

Description

checkConfiguration takes a (possibly incomplete) configuration of **irace**, checks for errors and transforms it into a valid configuration.

Usage

```
checkConfiguration(configuration = defaultConfiguration())
```

Arguments

configuration	A list where tagged elements correspond to configuration parameters of **irace**.

Details

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

Value

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

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

readConfiguration for reading the **irace** configuration from a file. printConfiguration for printing the **irace** configuration. defaultConfiguration to get the default configuration.

`defaultConfiguration` *defaultConfiguration*

Description

'defaultConfiguration' returns the default param configuration of **irace**.

Usage

```
defaultConfiguration(configuration = list())
```

Arguments

`configuration`	A list where tagged elements correspond to configuration parameters of **irace**.

Value

A list indexed by the **irace** parameter names, containing the default values for each parameter, except for those already present in the configuration passed as argument.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`readConfiguration` for reading the **irace** configuration from a file. `printConfiguration` for printing the **irace** configuration. `checkConfiguration` to check that a configuration is valid.

`hook.evaluate.default` *hook.evaluate.default*

Description

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

Usage

```
hook.evaluate.default(instance, candidate, num.candidates, extra.params,
config, hook.run.call)
```

Arguments

`instance`	A string containing the name of the instance (or filename and full path in case the instance is a file).
`candidate`	The candidate configuration that must be run.
`num.candidates`	The total number of candidates evaluated in this iteration.
`extra.params`	Extra parameters (like instance-specific ones) to be passed when evaluating this candidate.
`config`	options passed when invoking **irace**.
`hook.run.call`	a string describing the call to hookRun that corresponds to this call to hookEvaluate. This is used only for providing extra information to the user, for example, in case hookEvaluate fails.

Value

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

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

`hook.run.default` *hook.run.default*

Description

`hook.run.default` is the default hookRun function. You can use it as an advanced example of how to create your own hooRun function.

Usage

```
hook.run.default(instance, candidate, extra.params, config)
```

Arguments

`instance`	A string containing the name of the instance (or filename and full path in case the instance is a file).
`candidate`	The candidate configuration that must be run.
`extra.params`	Extra parameters (like instance-specific ones) to be passed when evaluating this candidate.
`config`	options passed when invoking **irace**.

Value

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

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

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

`irace.cmdline` *irace.cmdline*

Description

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

Usage

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

Arguments

`args`	The arguments provided on the R command line as a character vector, e.g., `c("--config-file", "tune-conf", "-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 configuration file, initializes the configuration from the file (with the function 'readConfiguration') and possibly from parameters passed on the command line. It finally starts **irace** by calling 'irace.main'.

Value

None.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

`irace.main` to start **irace** from a configuration.

`irace.license` *irace.license*

Description

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

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

`irace.main` *irace.main*

Description

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

Usage

```
irace.main(tunerConfig = defaultConfiguration(), output.width = 9999)
```

Arguments

`tunerConfig`	The configuration of **irace**.
`output.width`	The width that must be used for the screen output.

Details

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

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

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

`readConfiguration` to read the configuration of **irace** from a file.

`defaultConfiguration` to provide a default configuration for **irace**.

`irace.usage` *irace.usage*

Description

This function prints all possible arguments for the configuration to be used by **irace**, with the corresponding switches and a short description.

Usage

```
irace.usage()
```

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

`irace.version` *irace.version*

Description

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

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

`printConfiguration` *printConfiguration*

Description

'printConfiguration' prints the internal configuration of **irace**.

Usage

```
printConfiguration(configuration)
```

Arguments

`configuration`	A list where tagged elements correspond to configuration parameters of **irace**.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

readConfiguration for reading the **irace** configuration from a file. defaultConfiguration to get the default configuration. checkConfiguration to check that the configuration is valid.

---

readConfiguration *readConfiguration*

Description

'readConfiguration' reads the configuration to be used by **irace** from a file.

Usage

```
readConfiguration(filename = "", configuration = list())
```

Arguments

filename	A filename from which the configuration will be read. If empty, the default configurationFile is used. An example configuration file is provided in system.file(package="irace", "templates/tune-conf.tmpl").
configuration	A list where tagged elements correspond to configuration parameters of **irace**. This is an initial configuration that is overwritten for every parameter specified in the file to be read.

Value

The configuration read from the file. Anything not mentioned in the file is not present in the list, that is, it is NULL.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

checkConfiguration to check that the configuration is valid. defaultConfiguration to set the configuration to the default. printConfiguration to print the configuration.

readParameters *readParameters*

Description

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

Usage

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

Arguments

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

Details

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

Value

A list containing the definitions of the parameters read.

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

Examples

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

removeCandidatesMetaData *removeCandidatesMetaData*

Description

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

Usage

```
removeCandidatesMetaData(candidates)
```

Arguments

candidates	A matrix containg the candidates, one per row.

Value

The same matrix without the "metadata".

Author(s)

Manuel López-Ibáñez and Jérémie Dubois-Lacoste

See Also

candidates.print.command to print the candidates as command lines. candidates.print to print the candidates as a data frame.

Index