The **irace** Package: User Guide

Manuel López-Ibáñez, Leslie Pérez Cáceres, Jérémie Dubois-Lacoste,
Thomas Stützle and Mauro Birattari
IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium

Version 4.0, November 26, 2024

Contents

1 General information

1.1 Background

The **irace** package implements an *iterated racing* procedure, which is an extension of Iterated F-race (I/F-Race) [3]. The main use of **irace** is the automatic configuration of optimization and decision algorithms, that is, finding the most appropriate settings of an algorithm given a set of instances of a problem. However, it may also be useful for configuring other types of algorithms when performance depends on the used parameter settings. It builds upon the **race** package by Birattari and it is implemented in R. The **irace** package is available from CRAN:

`https://cran.r-project.org/package=irace`

More information about **irace** is available at `https://mlopez-ibanez.github.io/irace`.

1.2 Version

The current version of the **irace** package is 4.0. Previous versions of the package can also be found in the CRAN website.

The algorithm underlying the current version of **irace** and its motivation are described by López-Ibáñez et al. [12]. The **adaptive capping mechanism** available from version 3.0 is described by Pérez Cáceres et al. [15]. Details of the implementation before version 2.0 can be found in a previous technical report [11].

Versions of **irace** before 2.0 are not compatible with the file formats detailed in this document.

1.3 License

The **irace** package is Copyright © 2024 and distributed under the GNU General Public License version 3.0 (`http://www.gnu.org/licenses/gpl-3.0.en.html`). The **irace** package is free software (software libre): You can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

The **irace** package is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Please be aware that the fact that this program is released as Free Software does not excuse you from scientific propriety, which obligates you to give appropriate credit! If you write a scientific paper describing research that made substantive use of this program, it is your obligation as a scientist to (a) mention the fashion in which this software was used in the Methods section; (b) mention the algorithm in the References section. The appropriate citation is:

> 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*, 3:43–58, 2016. doi: 10.1016/j.orp.2016.09.002

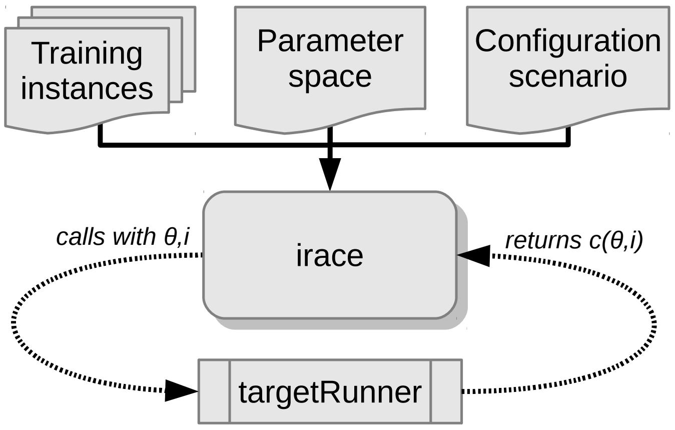

Figure 1: Scheme of **irace** flow of information.

2 Before starting

The **irace** package provides an automatic configuration tool for tuning optimization algorithms, that is, automatically finding good configurations for the parameters values of a (target) algorithm saving the effort that normally requires manual tuning.

Figure 1 gives a general scheme of how **irace** works. **Irace** receives as input a *parameter space definition* corresponding to the parameters of the target algorithm that will be tuned, a set of *instances* for which the parameters must be tuned for and a set of options for **irace** that define the *configuration scenario*. Then, **irace** searches in the parameter search space for good performing algorithm configurations by executing the target algorithm on different instances and with different parameter configurations. A `targetRunner` must be provided to execute the target algorithm with a specific parameter configuration (θ) and instance (i). The `targetRunner` function (or program) acts as an interface between the execution of the target algorithm and **irace**: It receives the instance and configuration as arguments and must return the evaluation of the execution of the target algorithm.

The following user guide contains guidelines for installing **irace**, defining configuration scenarios, and using **irace** to automatically configure your algorithms.

3 Installation

3.1 System requirements

- R (version $\geq$ 3.2.0) is required for running irace, but you don't need to know the R language to use it. R is freely available and you can download it from the R project website (`https://www.r-project.org`). See Appendix A for a quick installation guide of R.

- For GNU/Linux and OS X, the command-line executable `parallel-irace` requires GNU Bash. Individual examples may require additional software.

3.2 irace installation

The **irace** package can be installed automatically within R or by manual download and installation. We advise to use the automatic installation unless particular circumstances do not allow it. The instructions to install **irace** with the two mentioned methods are the following:

3.2.1 Install automatically within R

Execute the following line in the R console to install the package:

```r
install.packages("irace")
```

Select a mirror close to your location, and test the installation in the R console with:

```r
library("irace")
q() # To exit R
```

Alternatively, within the R graphical interface, you may use the `Packages and data->Package installer` menu on OS X or the `Packages` menu on Windows.

3.2.2 Manual download and installation

From the **irace** package CRAN website (https://cran.r-project.org/package=irace), download one of the three versions available depending on your operating system:

- `irace_4.0.tar.gz` (Unix/BSD/GNU/Linux)
- `irace_4.0.tgz` (OS X)
- `irace_4.0.zip` (Windows)

To install the package on GNU/Linux and OS X, you must execute the following command at the shell (replace `<package>` with the path to the downloaded file, either `irace_4.0.tar.gz` or `irace_4.0.zip`):

```
R CMD INSTALL <package>
```

To install the package on Windows, open R and execute the following line on the R console (replace `<package>` with the path to the downloaded file `irace_4.0.zip`):

```r
install.packages("<package>", repos = NULL)
```

If the previous installation instructions fail because of insufficient permissions and you do not have sufficient admin rights to install **irace** system-wide, then you need to force a local installation.

3.2.3 Local installation

Let's assume you wish to install **irace** on a path denoted by `<R_LIBS_USER>`, which is a filesystem path for which you have sufficient rights. This directory **must** exist before attempting the installation. Moreover, you must provide to R the path to this library when loading the package. However, the latter can be avoided by adding the path to the system variable `R_LIBS` or to the R internal variable `.libPaths`, as we will see below.[1]

On GNU/Linux or OS X, execute the following commands to install the package on a local directory:

[1] On Windows, see also https://cran.r-project.org/bin/windows/base/rw-FAQ.html#I-don_0027t-have-permission-to-write-to-the-R_002d3_002e3_002e1_005clibrary-directory.

```
export R_LIBS_USER="<R_LIBS_USER>"
# Create R_LIBS_USER if it doesn't exist
mkdir $R_LIBS_USER
# Replace <package> with the path to the downloaded file.
R CMD INSTALL --library=$R_LIBS_USER <package>
# Tell R where to find R_LIBS_USER
export R_LIBS=${R_LIBS_USER}:${R_LIBS}
```

On Windows, you can install the package on a local directory by executing the following lines in the R console:

```
# Replace <package> with the path to the downloaded file.
# Replace <R_LIBS_USER> with the path used for installation.
install.packages("<package>", repos = NULL, lib = "<R_LIBS_USER>")
# Tell R where to find R_LIBS_USER.
# This must be executed for every new session.
.libPaths(c("<R_LIBS_USER>", .libPaths()))
```

3.2.4 Testing the installation and invoking irace

Once **irace** has been installed, load the package and test that the installation was successful by opening an R console and executing:

```
# Load the package
library("irace")
# Obtain the installation path
system.file(package = "irace")
```

The last command must print out the filesystem path where **irace** is installed. In the remainder of this guide, the variable `$IRACE_HOME` is used to denote this path. When executing any provided command that includes the `$IRACE_HOME` variable do not forget to replace this variable with the installation path of **irace**.

On GNU/Linux or OS X, you can let the operating system know where to find **irace** by defining the `$IRACE_HOME` variable and adding it to the system `PATH`. Append the following commands to `~/.bash_profile`, `~/.bashrc` or `~/.profile`:

```
# Replace <IRACE_HOME> with the irace installation path
export IRACE_HOME=<IRACE_HOME>
export PATH=${IRACE_HOME}/bin/:$PATH
# Tell R where to find R_LIBS_USER
# Use the following line only if local installation was forced
export R_LIBS=${R_LIBS_USER}:${R_LIBS}
```

Then, open a new terminal and launch **irace** as follows:

```
irace --help
```

On Windows, you need to add both R and the installation path of **irace** to the environment variable `PATH`. To edit the `PATH`, search for "Environment variables" in the control panel, edit `PATH` and add a string similar to `C:\R_PATH\bin;C:\IRACE_HOME\bin\x64\` where `R_PATH` is the installation path of R and IRACE_HOME is the installation path of **irace**. If **irace** was installed locally, you also need to edit the environment variable R_LIBS to add R_LIBS_USER. Then, open a new terminal (run program cmd.exe) and launch **irace** as:

```
irace.exe --help
```

Alternatively, you may directly invoke **irace** from within the R console by executing:

```
library("irace")
irace_cmdline("--help")
```

4 Running irace

Before performing the tuning of your algorithm, it is necessary to define a tuning scenario that will give **irace** all the necessary information to optimize the parameters of the algorithm. The tuning scenario is composed of the following elements:

1. Target algorithm parameter description (see Section 5.1).
2. Target algorithm runner (see Section 5.2).
3. Training instances list (see Section 5.4)
4. **irace** options (see Section 11).
5. *Optional:* Initial configurations (see Section 5.5).
6. *Optional:* Target algorithm evaluator (see Section 5.3).

These scenario elements can be provided as plain text files or as R objects. This user guide provides examples of both types, but we advise the use of plain text files, which we consider the simpler option.

For a step-by-step guide to create the scenario elements for your target algorithm continue to Section 4.1. For an example execution of **irace** using the **ACOTSP** scenario go to Section 4.2.

4.1 Step-by-step setup guide

This section provides a guide to setup a basic execution of **irace**. The template files provided in the package ($IRACE_HOME/templates) will be used as basis for creating your new scenario. Please follow carefully the indications provided in each step and in the template files used; if you have doubts check the the sections that describe each option in detail.

1. Create a directory (e.g., ./tuning/) for the scenario setup. This directory will contain all the files that describe the scenario. On GNU/Linux or OS X, you can do this as follows:

   ```
   mkdir ./tuning
   cd ./tuning
   ```

2. Initialize the tuning directory with template config files. On GNU/Linux or OS X, you can do this as follows:

```
irace --init
```

3. Define the target algorithm parameters to be tuned by following the instructions in `parameters.txt`. Available parameter types and other guidelines can be found in Section 5.1.

4. *Optional*: Define the initial parameter configuration(s) of your algorithm, which allows you to provide good starting configurations (if you know some) for the tuning. Follow the instructions in `configurations.txt` and set `configurationsFile="configurations.txt"` in `scenario.txt`. More information in Section 5.5. If you do not need to define initial configurations remove this file from the directory.

5. Place the instances you would like to use for the tuning of your algorithm in the folder `./tuning/Instances/`. In addition, you can create a file (e.g., `instances-list.txt`) that specifies which instances from that directory should be run and which instance-specific parameters to use. To use such an instance file, set the appropriate option in `scenario.txt`, e.g., `trainInstancesFile = "instances-list.txt"`. See Section 5.4 for guidelines.

6. Uncomment and assign in `scenario.txt` only the options for which you need a value different from the default. Some common options that you might want to adjust are:

   `execDir` (`--exec-dir`): the directory in which **irace** will execute the target algorithm; the default value is the current directory.

   `maxExperiments` (`--max-experiments`): the maximum number of executions of the target algorithm that **irace** will perform.

   `maxTime` (`--max-time`): maximum total execution time in seconds for the executions of `targetRunner`. In this case, `targetRunner` must return two values: cost and time. Note that you must provide either `maxTime` or `maxExperiments`.

   `trainInstancesDir` (`--train-instances-dir`): set if to `./Instances` if you put the training instances in that folder as instructed above.

   For setting the tuning budget see Section 10.1. For more information on **irace** options and their default values, see Section 11.

7. Modify the `target-runner` script to run your algorithm. This script must execute your algorithm with the parameters and instance specified by **irace** and return the evaluation of the execution and *optionally* the execution time (`cost [time]`). When the `maxTime` option is used, returning `time` is mandatory. The `target-runner` template is written in GNU Bash scripting language, which can be executed easily in GNU/Linux and OS X systems. However, you may use any other programming language. We provide examples written in Python, MATLAB and other languages in `$IRACE_HOME/examples/`. Follow these instructions to adjust the given `target-runner` template to your algorithm:

   (a) Set the `EXE` variable with the path to the executable of the target algorithm.

   (b) Set the `FIXED_PARAMS` if you need extra arguments in the execution line of your algorithm. An example could be the time that your algorithm is required to run (`FIXED_PARAMS="--time 60"`) or the number of evaluations required (`FIXED_PARAMS= "--evaluations 10000"`).

   (c) The line provided in the template executes the executable described in the `EXE` variable.

   ```
   $EXE ${FIXED_PARAMS} -i ${INSTANCE} --seed ${SEED} ${CONFIG_PARAMS}
   ```

You must change this line according to the way your algorithm is executed. In this example, the algorithm receives the instance to solve with the flag `-i` and the seed of the random number generator with the flag `--seed`. The variable `CONFIG_PARAMS` adds to the command line the parameters that **irace** has given for the execution. You must set the command line execution as needed. For example, the instance might not need a flag and might need to be the first argument:

```
$EXE ${INSTANCE} ${FIXED_PARAMS} --seed ${SEED} ${CONFIG_PARAMS}
```

The output of your algorithm is saved to the file defined in the `$STDOUT` variable, and error output is saved in the file given by `$STDERR`. The line:

```
if [ -s "$STDOUT" ]; then
```

checks if the file containing the output of your algorithm is not empty. The example provided in the template assumes that your algorithm prints in the last output line the best result found (only a number). The line:

```
COST=$(cat ${STDOUT} | grep -e '^[[:space:]]*[+-]\?[0-9]' | cut -f1)
```

parses the output of your algorithm to obtain the result from the last line. The `target-runner` script must print **only** one number. In the template example, the result is printed with `echo "$COST"` (assuming `maxExperiments` is used) and the generated files are deleted (you may remove that line if you wish to keep them).

The `target-runner` script must be an executable file, unless you specify `targetRunnerLauncher`.

You can test the target runner from the R console by checking the scenario as explained earlier in Section 4.

If you have problems related to the `target-runner` script when executing **irace**, see Appendix B for a check list to help diagnose common problems. For more information about the `targetRunner`, please see Section 5.2,

8. *Optional*: Modify the `target-evaluator` file. This is rarely needed and the `target-runner` template does not use it. Section 5.3 explains when a `targetEvaluator` is needed and how to define it.

9. The **irace** executable provides an option (`--check`) to check that the scenario is correctly defined. We recommend to perform a check every time you create a new scenario. When performing the check, **irace** will verify that the scenario and parameter definitions are correct and will test the execution of the target algorithm. To check your scenario execute the following commands:

   - From the command-line (on Windows, execute `irace.bat`):

   ```
   # $IRACE_HOME is the installation directory of irace.
   $IRACE_HOME/bin/irace --scenario scenario.txt --check
   ```

   - Or from the R console:

   ```
   library("irace")
   scenario <- readScenario(filename = "scenario.txt",
                            scenario = defaultScenario())
   checkIraceScenario(scenario = scenario)
   ```

10. Once all the scenario elements are prepared you can execute **irace**, either using the command-line wrappers provided by the package or directly from the R console:

   - **From the command-line console**, call the command (on Windows, you should execute `irace.exe`):

```
cd ./tuning/
# $IRACE_HOME is the installation directory of irace
# By default, irace reads scenario.txt, you can specify a different file
# with --scenario.
$IRACE_HOME/bin/irace
```

   For this example we assume that the needed scenario files have been set properly in the `scenario.txt` file using the options described in Section 11. Most **irace** options can be specified in the command line or directly in the `scenario.txt` file.

   - **From the R console**, evaluate:

```
library("irace")
# Go to the directory containing the scenario files
setwd("./tuning")
scenario <- readScenario(filename = "scenario.txt",
                         scenario = defaultScenario())
irace_main(scenario = scenario)
```

   This will perform one run of **irace**. See the output of `irace --help` in the command-line or `irace_cmdline("--help")` in R for quick information on additional **irace** options. For more information about **irace** options, see Section 11.

Command-line options override the same options specified in the `scenario.txt` file.

4.2 Setup example for ACOTSP

The **ACOTSP** tuning example can be found in the package installation in the folder `$IRACE_HOME/examples/acotsp`. Other example scenarios can be found in the same folder. More examples of tuning scenarios can be found in the Algorithm Configuration Library (AClib, http://www.aclib.net/).

In this section, we describe how to execute the **ACOTSP** scenario. If you wish to start setting up your own scenario, continue to the next section. For this example, we assume a GNU/Linux system such as Ubuntu with a working C compiler such as `gcc`. To execute this scenario follow these steps:

1. Create a directory for the tuning (e.g., `./tuning/`) and copy the example scenario files located in the `examples` folder to the created directory:

```
mkdir ./tuning
cd ./tuning
# $IRACE_HOME is the installation directory of irace.
cp $IRACE_HOME/examples/acotsp/* ./tuning/
```

2. Download the training instances from https://iridia.ulb.ac.be/supp/IridiaSupp2016-003/index.html to the `./tuning/` directory.

3. Create the instance directory (e.g., `./tuning/Instances`) and decompress the instance files on it.

```
mkdir ./tuning/Instances/
cd ./tuning/
tar -xvf tsp-instances-training.tar.bz2 Instances/
```

4. Download the **ACOTSP** software from http://www.aco-metaheuristic.org/aco-code/ to the `./tuning/` directory and compile it.

```
cd ./tuning/
tar -xvf ACOTSP-1.03.tgz
cd ./tuning/ACOTSP-1.03
make
```

5. Create a directory for executing the experiments and execute **irace**:

```
mkdir ./tuning/acotsp-arena/
cd ./tuning/
# $IRACE_HOME is the installation directory of irace.
$IRACE_HOME/bin/irace
```

6. Or you can also execute **irace** from the R console using:

```
library("irace")
setwd("./tuning/")
irace_cmdline()
```

5 Defining a configuration scenario

5.1 Target algorithm parameters

The parameters of the target algorithm are defined by a parameter file as described in Section 5.1.7. Optionally, when executing **irace** from the R console, the parameters can be specified directly as an R object (see Section 5.1.8). For defining your parameters follow the guidelines provided in the following sections.

5.1.1 Parameter types

Each target parameter has an associated type that defines its domain and the way **irace** handles them internally. Understanding the nature of the domains of the target parameters is important to select appropriate types. The four basic types supported by **irace** are the following:

- *Real* parameters are numerical parameters that can take floating-point values within a given range. The range is specified as an interval '`(<lower bound>,<upper bound>)`'. This interval is closed, that is, the parameter value may eventually be one of the bounds. The possible values are rounded to a number of *decimal places* specified by the global option `digits` (Section 5.1.6). For example, given the default number of digits of 4, the values 0.12345 and 0.12341 are both rounded to 0.1234. Selected real-valued parameters can be optionally sampled on a logarithmic scale (base e).

- *Integer* parameters are numerical parameters that can take only integer values within the given range. Their range is specified as the range of real parameters and they can also be optionally sampled on a logarithmic scale (base e).
- *Categorical* parameters are defined by a set of possible values specified as '`(<value 1>, ..., <value n>)`'. The values are quoted or unquoted character strings. Empty strings and strings containing commas or spaces must be quoted.
- *Ordinal* parameters are defined by an *ordered* set of possible values in the same format as for categorical parameters. They are handled internally as integer parameters, where the integers correspond to the indexes of the values.

Boolean (or logical) parameters are best encoded as categorical ones with just two values rather than integer ones with domain $(0, 1)$. Some boolean parameters take an explicit value (0/1 or true/false) such as:

```
dlb "--dlb " c (0, 1)
```

Others are switches whose presence activates the parameter:

```
dlb "" c ("", "--dlb")
```

5.1.2 Parameter domains

For each target parameter, an interval or a set of values must be defined according to its type, as described above. There is no limit for the size of the set or the length of the interval, but keep in mind that larger domains could increase the difficulty of the tuning task. Choose always values that you consider relevant for the tuning. In case of doubt, we recommend to choose larger intervals, as occasionally best parameter settings may be not intuitive a priori. All intervals are considered as closed intervals.

It is possible to define parameters that will have always the same value. Such "*fixed*" parameters will not be tuned but their values are used when executing the target algorithm and they are affected by constraints defined on them. All fixed parameters must be defined as categorical parameters and have a domain of one element.

5.1.3 Parameter dependent domains

Domains that are dependent on the values of other parameters can be specified only for numerical parameters (both integer and real). To do so, the dependent domain must be expressed in function of another parameter, which must be a numerical parameter. The expression that defines a dependency must be written between quotes: `(value,"expression")` or `("expression",value)` or `("expression","expression")`.

The expressions can only use the following operators and R functions: `+`, `-`, `*`, `/`, `%%`, `min`, `max`, `round`, `floor`, `ceiling`, `trunc`. If you need to use an operator or function not listed here, please contact us.

The user must ensure that the defined domain is valid at all times since **irace** currently is not able to detect possible invalid domains based on the expressions provided.

If you have a parameter `p2` that is just a transformation of another `p1`, then instead of using a dependent domain (left-hand side of the following example), it will be better to create a dummy parameter that controls the transformation (right-hand side) and do the transformation within `target-runner`. For example:

```
# With dependent domains
p1 "" r (0, 100)
p2 "" r ("p1", "p1 + 10")
```

should be

```
# With a dummy parameter
p1    "" r (0, 100)
p2dum "" r (0, 10)
```

and `target-runner` will compute $\texttt{p2} = \texttt{p2dum} \cdot \texttt{p1}$.

5.1.4 Conditional parameters

Conditional parameters are active only when others have certain values. These dependencies define a hierarchical relation between parameters. For example, the target algorithm may have a parameter `localsearch` that takes values `(sa,ts)` and another parameter `ts-length` that only needs to be set if the first parameter takes precisely the value `ts`. Thus, parameter `ts-length` is conditional on `localsearch == "ts"`.

5.1.5 Forbidden parameter configurations

A line containing just `[forbidden]` ends the list of parameters and starts the list of forbidden expressions. Each line is a logical expression (in R syntax) containing parameter names as defined by the `parameterFile` (Section 5.1), values and logical operators. For a list of R logical operators see:

https://stat.ethz.ch/R-manual/R-devel/library/base/html/Syntax.html

If **irace** generates a parameter configuration that makes any of the logical expressions evaluate to `TRUE`, then the configuration is considered forbidden and it is never evaluated. This is useful when some combination of parameter values could cause the target algorithm to crash, consume excessive CPU time or memory, or when it is known that they do no produce satisfactory results.

Initial configuration (Section 5.5) that are forbidden will be discarded with a warning.

If the forbidden constraints provided are too strict, **irace** may produce the following error:

```
irace tried 100 times to sample from the model a configuration not forbidden
without success, perhaps your constraints are too strict?
```

In that case, it may be a good idea to reformulate the forbidden constraints as conditional parameters (Section 5.1.4), parameter-dependent domains (Section 5.1.3), repairing the configurations (Section 5.6) or post-processing within the target-algorithm (Section 10.7).

5.1.6 Global options

A line containing just `[global]` starts the definition of global options. The only global option currently implemented is `digits`, which controls the number of decimal digits for real valued parameters. Its default value is 4.

5.1.7 Parameter file format

For simplicity, the description of the parameters space is given as a table. Each line of the table defines a configurable parameter

```
<name> <label> <type> <domain> [ | <condition> ]
```

where each field is defined as follows:

`<name>` The name of the parameter as an unquoted alphanumeric string, e.g., '`ants`'.

`<label>` A *label* for this parameter. This is a string that will be passed together with the parameter to `targetRunner`. In the default `targetRunner` provided with the package (Section 5.2), this is the command-line switch used to pass the value of this parameter, for instance '`"--ants "`'.
The value of the parameter is concatenated *without separator* to the label when invoking `targetRunner`, thus *any whitespace in the label is significant*. Following the same example, when parameter `ants` takes value `5`, the default targetRunner will pass the parameter as `"--ants 5"`.

`<type>` The type of the parameter, either *integer*, *real*, *ordinal* or *categorical*, given as a single letter: '`i`', '`r`', '`o`' or '`c`'. Numerical parameters can be sampled using a natural logarithmic scale with '`i,log`' and '`r,log`' (without spaces) for integer and real parameters, respectively.

`<domain>` The range or set of values of the parameter delimited by parentheses, e.g., `(0,1)` or `(a,b,c,d)`. See also parameter dependent domains (Section 5.1.3).

`<condition>` An optional *condition* that determines whether the parameter is enabled or disabled, thus making the parameter conditional. If the condition evaluates to false, then no value is assigned to this parameter, and neither the parameter value nor the corresponding label are passed to `targetRunner`. The condition must follow the same syntax as those for specifying forbidden configurations (see below), that is, it must be a valid R logical expression[2]. The condition may contain the name of other parameters as long as the dependency graph does not contain any cycle. Otherwise, **irace** will detect the cycle and stop with an error.

Categorical and ordinal parameters are always treated as strings. Given a parameter like:

```
a "" c (0, 5, 10, 20)
```

then, a condition like `a >10` will be true when `a` is `5`, because comparisons between strings are lexicographic and `"10"` is sorted before `"5"`. As a work-around, you can convert the string to numeric in the condition with `as.numeric(a)`.

As an example, Figure 2 shows the parameters file of the **ACOTSP** scenario.

5.1.8 Parameters R format

The target parameters are stored in an R `list` that you can obtain from the R console using the following command:

```
parameters <- readParameters(file = "parameters.txt")
```

See the help of the `readParameters` function (`?readParameters`) for more information. The structure of the parameter list that is created is as follows:

[2]For a list of R operators see: https://stat.ethz.ch/R-manual/R-devel/library/base/html/Syntax.html

```
# 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    (5, 100)
nnls        "--nnls "          i    (5, 50)              | localsearch %in% c(1, 2, 3)
q0          "--q0 "            r    (0.0, 1.0)           | algorithm == "acs"
dlb         "--dlb "           c    (0, 1)               | localsearch %in% c(1,2,3)
rasrank     "--rasranks "      i    (1, "ants")          | algorithm == "ras"
elitistants "--elitistants "   i    (1, 750)             | algorithm == "eas"

[forbidden]
## Examples of valid logical operators are:
## ==  !=  >=  <=  >  <  &  |  !  %in%
(alpha == 0.0) & (beta == 0.0)
```

Figure 2: Parameter file (`parameters.txt`) for tuning **ACOTSP**.

`names`	Vector that contains the names of the parameters.
`types`	Vector that contains the type of each parameter 'i', 'c', 'r', 'o'.
`switches`	Vector that contains the labels of the parameters. e.g., 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 a set of values for categorical and ordinal 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.
`transform`	Vector that contains the transformation of each parameter. Currently, it can take values ‘‘’’ (no transformation, default) of ‘‘`log`’’ (natural logarithmic transformation).
`nbParameters`	An integer, the total number of parameters.
`nbFixed`	An integer, the number of parameters with a fixed value.
`nbVariable`	Number of variable (i.e., to be tuned) parameters.
`isDependent`	Logical vector that specifies which parameter defines a dependent domain.
`forbidden`	List of R logical expressions that cannot evaluate to `TRUE` for any evaluated configuration.

The following example shows the structure of the `parameters` R object for the `algorithm`, `ants` and `q0` parameters of the **ACOTSP** scenario:

```
> str(parameters, vec.len = 10)

Classes 'ParameterSpace', 'R6' <ParameterSpace>
  Public:
    .params: list
    as_character: function ()
    clone: function (deep = FALSE)
    conditions: list
    depends: list
    domains: list
    forbid_configurations: function (x)
    forbidden: NULL
    get: function (x)
    get_ordered: function ()
    hierarchy: 1 1 2
    initialize: function (..., forbidden = NULL, verbose = 0L)
    isFixed: FALSE FALSE FALSE
    names: algorithm ants q0
    names_fixed:
    names_numeric: ants q0
    names_variable: algorithm ants q0
    nbFixed: 0
    nbParameters: 3
    nbVariable: 3
    switches: -- --ants  --q0
    types: c i r
```

5.2 Target algorithm runner

The evaluation of a candidate configuration on a single instance is done by means of a user-given auxiliary program or, alternatively, a user-given R function. The function (or program name) is specified by the option `targetRunner`. The `targetRunner` must return the cost value (e.g., cost of the best solution found) of the evaluation; unless computing the cost requires information from all the configurations evaluated on an instance, e.g., when evaluating multi-objective algorithms with unknown normalisation bounds (see Section 5.3 for details).

The objective of **irace** is to minimize the cost value returned by the target algorithm. If you wish to maximize, you can multiply the cost by `-1` before returning it to **irace**.

5.2.1 Target runner as an executable program

When `targetRunner` is an auxiliary executable program, it is invoked for each candidate configuration, passing as arguments:

```
<id_configuration> <id_instance> <seed> <instance> [bound] <configuration>
```

`id_configuration`	an alphanumeric string that uniquely identifies a configuration;
`id_instance`	an alphanumeric string that uniquely identifies an instance;
`seed`	seed for the random number generator to be used for this evaluation, ignore the seed for deterministic algorithms;
`instance`	string giving the instance to be used for this evaluation;
`bound`	optional execution time bound. Only provided when the `boundMax` option is set in the scenario, see Section 10.3;
`configuration`	the pairs parameter label-value that describe this candidate configuration. Typically given as command-line switches to be passed to the executable program.

The experiment list shown in Section 5.2.2, would result in the following execution line:

```
target-runner 1 113 734718556 /home/user/instances/tsp/2000-533.tsp \
 --eas --localsearch 0 --alpha 2.92 --beta 3.06 --rho 0.6 --ants 80
```

The command line switches that describe the candidate configuration are constructed by appending to each parameter label (switch), *without separator*, the value of the parameter, following the order given in the parameter table. The program `targetRunner` must print a real number, which corresponds to the cost measure of the candidate configuration for the given instance and optionally its execution time (mandatory when `maxTime` is used and/or when the `capping` option is enabled). The working directory of `targetRunner` is set to the execution directory specified by the option `execDir`. This allows the user to execute independent runs of **irace** in parallel using different values for `execDir`, without the runs interfering with each other.

5.2.2 Target runner as an R function

When `targetRunner` is an R function, it is invoked for each candidate configuration as:

```
targetRunner(experiment, scenario)
```

where `experiment` is a list that contains information about configuration and instance to execute one experiment, and `scenario` is the scenario list. The structure of the `experiment` list is as follows:

`id_configuration`	an alphanumeric string that uniquely identifies a configuration;
`id_instance`	an alphanumeric string that uniquely identifies an instance;
`seed`	seed to be used for this evaluation;
`instance`	string giving the instance to be used for this evaluation;
`bound`	optional execution time bound;
`configuration`	1-row data frame with a column per parameter name;
`switches`	vector of parameter switches (labels) in the order of parameters used in `configuration`.

The following is an example of an experiment list for the **ACOTSP** scenario:

```
> print(experiment)
```

```
$id_configuration
[1] 1

$id_instance
[1] 119

$seed
[1] 120344916

$configuration
  algorithm localsearch alpha beta  rho ants nnls q0  dlb rasrank
1        as           0     1    1 0.95   10   NA NA <NA>      NA
  elitistants time
1          NA    5

$instance
[1] "./instances/2000-539.tsp"

$switches
       algorithm      localsearch            alpha             beta
            "--" "--localsearch "       "--alpha "        "--beta "
             rho             ants             nnls               q0
       "--rho  "        "--ants "        "--nnls "          "--q0 "
             dlb          rasrank      elitistants             time
        "--dlb "    "--rasranks " "--elitistants "        "--time "
```

You can find an example that calls MATLAB from R using this approach here: https://github.com/MLopez-Ibanez/irace/blob/master/inst/examples/matlab/scenario.txt.

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. A cost of `Inf` is accepted and results in the immediate rejection of the configuration (see Section 10.8).

If the scenario option `maxTime` is non-zero or if the `capping` option 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;

5.3 Target evaluator

Normally, `targetRunner` returns the cost of the execution of a candidate configuration (see Section 5.2). However, there are cases when the cost evaluation must be delayed until all candidate configurations in a race have been executed on a instance.

The `targetEvaluator` option defines an auxiliary program (or an R function) that allows postponing the evaluations of the candidate configurations. For each instance seen, the program targetEvaluator is only invoked after all the calls to targetRunner for all alive candidate configurations on the same instance have already finished.

When using targetEvaluator, targetRunner must not return the evaluation of the configuration. If maxTime is used, targetRunner must return only execution time.

As an example, targetEvaluator may be used to dynamically find normalization bounds for the output returned by an algorithm for each individual instance. In this case, targetRunner will save the output of the algorithm, then the first call to targetEvaluator will examine the output produced by all calls to targetRunner for the same instance, update the normalization bounds and return the normalized output. Subsequent calls to targetEvaluator for the same instance will simply return the normalized output.

A similar need arises when using quality measures for multi-objective optimization algorithms, such as the hypervolume, which typically require specifying reference points or sets. By using targetEvaluator, it is possible to dynamically compute the reference points or sets while **irace** is running. Examples are provided at `examples/hypervolume`. See also Section 10.2 for more information on how to tune multi-objective algorithms.

5.3.1 Target evaluator executable program

When targetEvaluator is an auxiliary executable program, it is invoked for each candidate with the following arguments:

```
<id_configuration> <id_instance> <seed> <instance> <num_configurations> <all_conf_id>
```

`id_configuration`	an alphanumeric string that uniquely identifies a configuration;
`id_instance`	an alphanumeric string that uniquely identifies an instance;
`seed`	seed to be used for this evaluation;
`instance`	string giving the instance to be used for this evaluation;
`num_configurations`	number of alive candidate configurations;
`all_conf_id`	list of IDs of the alive configurations separated by whitespace.

The targetEvaluator executable must print a numerical value corresponding to the cost measure of the candidate configuration on the given instance.

5.3.2 Target evaluator R function

When targetEvaluator is an R function, it is invoked for each candidate configuration as:

```
targetEvaluator(experiment, num_configurations, all_conf_id, scenario,
                target_runner_call)
```

where `experiment` is a list that contains information about one experiment (see Section 5.2.2), `num_configurations` is the number of configurations alive in the race, `all_conf_id` is the vector of IDs of the alive configurations, `scenario` is the scenario list and `target_runner_call` is the string of the targetRunner execution line.

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 `targetEvaluator` called an external program or function;

5.4 Training instances

The **irace** options `trainInstancesDir` and `trainInstancesFile` specify where to find the training instances. If you only set the value of `trainInstancesDir` (e.g., to `./Instances`), **irace** will consider all files within that directory as training instances.

Otherwise, the value of `trainInstancesFile` may specify a text file. The format of this file is one instance per line. Within each line, elements separated by white-space will be parsed as separate arguments to be supplied to `targetRunner`. This allows defining instance-specific parameter settings. Quoted strings will be parsed as a single argument. The following example shows a training instance file for the **ACOTSP** scenario:

```
# Example training instances file
100/100-1_100-2.tsp --time 1
100/100-1_100-3.tsp --time 2
100/100-1_100-4.tsp --time 3
```

Figure 3: Training instances file for tuning **ACOTSP**.

The value of `trainInstancesDir`, if set, is always prefixed to the instance name, that is, the instances names are treated as relative to this directory. For example, given the above file as `trainInstancesFile` and `trainInstancesDir="./Instances"`, then a possible invocation of `targetRunner` would be:

```
target-runner 1 4 5718 ./Instances/100/100-1_100-2.tsp --time 1 --alpha 2.92 ...
```

Training instances do not need to be files, **irace** just passes the elements of each line as arguments to `targetRunner`, thus each line may denote the name of a benchmark function or a label, plus instance-specific settings, that the target algorithm understands. Each line may even be the command-line parameters required to call an instance generator within `targetRunner`. When the instances do not represent actual files, then `trainInstancesDir` is usually set to the empty string (`--train-instances-dir=""`). For example,

```
# Example training instances file
rosenbrock_20 --function=12 --nvar 20
rosenbrock_30 --function=12 --nvar 30
rastrigin_20 --function=15 --nvar 20
rastrigin_30 --function=15 --nvar 30
```

Optionally, when executing **irace** from the R console, the list of instances might be provided explicitly by means of the variable `scenario$instances`. Thus, the previous example would be equivalent to:

```r
scenario$instances <- c("rosenbrock_20 --function=12 --nvar 20",
                        "rosenbrock_40 --function=12 --nvar 30",
                        "rastrigin_20 --function=15 --nvar 20",
                        "rastrigin_40 --function=15 --nvar 30")
```

By default, **irace** assumes that the target algorithm is stochastic (the value of the option `deterministic` is 0), thus, the same configuration can be executed more than once on the same instance and obtain different results. In this case, **irace** generates pairs `(instance,seed)` by generating a random seed for each instance. In other words, configurations evaluated on the same instance use the same random seed. This is a well-known variance reduction technique called *common random numbers* [14]. If all available pairs are used within a run of **irace**, new pairs are generated with different seeds, that is, a configuration evaluated more than once per instance will use different random seeds.

If `deterministic` is set to 1, then each instance will be used at most once per race. This setting should only be used for target algorithms that do not have a stochastic behavior and, therefore, executing the target algorithm on the same instance several times with different seeds does not make sense.

If `deterministic` is active and the number of training instances provided to **irace** is less than `firstTest` (default: 5), no statistical test will be performed on the race.

Finally, **irace** randomly re-orders the sequence of instances provided. This random sampling may be disabled by using the option `sampleInstances` (`--sample-instances 0`) if keeping the order provided in the instance file is important.

We advise to always sample instances to prevent biasing the tuning due to the instance order. See also Section 10.5

5.5 Initial configurations

The scenario option `configurationsFile` allows specifying a text file that contains an initial set of configurations to start the execution of **irace**. If the number of initial configurations supplied in the file is less than the number of configurations required by **irace** in the first iteration, additional configurations will be sampled uniformly at random.

The format of the configurations file is one configuration per line, and one parameter value per column. The first line must give the parameter name corresponding to each column (names must match those given in the parameters file). Each configuration must satisfy the parameter conditions (`NA` should be used for those parameters that are not enabled for a given configuration) and not be forbidden by the constraints that define forbidden configurations (Section 5.1.5), if any.

Figure 4 gives an example file that corresponds to the **ACOTSP** scenario.

```
## Initial candidate configuration for irace
algorithm localsearch alpha beta rho  ants nnls dlb q0 rasrank elitistants
as        0           1.0   1.0  0.95 10   NA   NA  0  NA      NA
```

Figure 4: Initial configuration file (`default.txt`) for tuning **ACOTSP**.

We advise to use this feature when a default configuration of the target algorithm exists or when different sets of good parameter values are known. This will allow **irace** to start the search from those parameter values and attempt to improve their performance.

5.6 Repairing configurations

In some problems, the parameter values require complex constraints that cannot be implemented by constraints defined in the parameter space (Section 5.1.5). The scenario option repairConfiguration can be set to a user-defined R function that takes a single configuration generated by irace and returns a "*repaired*" configuration, thus allowing the implementation of any rules necessary to satisfy arbitrary constraints on parameter values. The repairConfiguration function is called after generating a configuration and before checking for forbidden configurations. The first argument is a 1-row data.frame with parameter names as the column names and the second argument is the parameters list (Section 5.1.8). An example that makes all real-valued parameters sum up to one would be:

```r
repairConfiguration = function(configuration, parameters)
{
  isreal <- names(which(parameters$types[colnames(configuration)] == "r"))
  # This ignores 'digits'
  c_real <- unlist(configuration[isreal])
  c_real <- c_real / sum(c_real)
  configuration[isreal] <- c_real
  return(configuration)
}
```

The following example forces three specific parameters to be in increasing order:

```r
repairConfiguration = function(configuration, parameters)
{
 columns <- c("p1","p2","p3")
 # cat("Before"); print(configuration)
 configuration[columns] <- sort(unlist(configuration[columns], use.names=FALSE))
 # cat("After"); print(configuration)
 return(configuration)
}
```

The above code can be specified directly in the scenarioFile, by default scenario.txt.

6 Parallelization

A single run of **irace** can be done much faster by executing the calls to targetRunner (the runs of the target algorithm) in parallel. There are four ways to parallelize a single run of **irace**:

1. **Parallel processes**: The option parallel executes multiple calls to targetRunner in parallel within a single computer, by means of the **parallel** R package. For example, adding --parallel N to the command line of **irace** will launch in parallel up to N calls of the target algorithm. When using this option within a computing cluster, **irace** will be submitted as a *job* in some way that tells the cluster to "reserve" N CPUs (or tasks depending on the cluster) within a single cluster node (a single machine).

2. **MPI**: By enabling the option mpi, calls to targetRunner will be executed in parallel by using the message passing interface (MPI) protocol (requires the **Rmpi** R package). In this case, the option parallel controls the number of slave nodes used by **irace**. For example, adding --mpi 1 --parallel N to the command-line will create N slaves + 1 master, and execute up to N calls of targetRunner in parallel.

   The user is responsible for setting up the required MPI environment. MPI is commonly available in computing clusters and requires launching **irace** in some particular way. An example script for using MPI mode in a SGE cluster is given at `$IRACE_HOME/bin/parallel-irace-mpi`.

By default, **irace** dynamically balances the load among nodes, however, this may significantly increase communication overhead in some parallel environments and disabling `loadBalancing` may be faster.

3. **Batch queue mode**: Some computing clusters work by submitting jobs to a batch queue and waiting for the jobs to finish. With the option `batchmode` (`--batchmode [sge|pbs|torque|slurm]`), **irace** will launch in parallel as many calls of `targetRunner` as possible and use a cluster-specific method to wait for jobs to finish. In this mode, `parallel` controls how many jobs are queued and should be set to the queue limit of your cluster. If your cluster type is not supported or not working as expected, please submit a pull request (`https://github.com/MLopez-Ibanez/irace/pulls`) adding support to your cluster type. See the examples in `$IRACE_HOME/examples/batchmode-cluster/`.

   In batchmode, **irace** runs in the submission node of the cluster, hence, **irace** is not submitted to the cluster as a job (that is, neither **qsub** nor **squeue** should be used to invoke **irace** itself). The user must call the appropriate job submission command (e.g., **qsub** or **squeue**) from `targetRunner` with the appropriate settings for their cluster, that is, `targetRunner` submits one job to the cluster and prints a single string: The job ID that allows **irace** to determine the status of the running job. Moreover, the use of a separate `targetEvaluator` script is required to collect the results of `targetRunner` and return them to **irace**.

4. `targetRunnerParallel`: This option allows users to fully control the parallelization of the execution of `targetRunner`. Its value must be an R function that will be invoked by **irace** as follows:

```
targetRunnerParallel(experiments, exec_target_runner, scenario, target_runner)
```

   where `scenario` is the list describing the configuration scenario (Section 5); `experiments` is a list that describes the configurations and instances to be executed (see Section 5.2 for a description); `target_runner` is the function that calls the target algorithm and it is the same as `targetRunner`, if the latter is a function, or it is a call to `target_runner_default`, if `targetRunner` is the path to an executable; and `exec_target_runner` is an internal function within **irace** that takes care of executing `target_runner`, check its output and, possibly, retry in case of error (see `targetRunnerRetries`). The `targetRunnerParallel` function should call the given `target_runner` function for each element in the `experiments` list, possibly using `exec_target_runner` as a wrapper. A trivial example would be:

```
targetRunnerParallel <- function(experiments, exec_target_runner, scenario,
                                 target_runner)
{
  lapply(experiments, exec_target_runner, scenario = scenario,
         target_runner = target_runner)
}
```

   However, the user is free to set up the calls in any way, perhaps implementing its own replacement for `target_runner` and/or `exec_target_runner`. The user may load and call other R packages, such as **batchtools** (`https://mllg.github.io/batchtools/`).

The only requirement is that the targetRunnerParallel function must return a list of the same length as experiments, where each element is the output expected from the corresponding call to targetRunner (see Section 5.2).

The following is an example of the output of a call to targetRunnerParallel with 2 experiments, in which the execution time is not reported:

```
   print(output)

## [[1]]
## [[1]]$cost
## [1] 40755522
##
## [[1]]$time
## numeric(0)
##
##
## [[2]]
## [[2]]$cost
## [1] 33549023
##
## [[2]]$time
## numeric(0)
```

The best option will depend on the resources available to you. Option 1 is usually the fastest and simplest to setup. Running on a node (machine) with 128 CPUs will be faster than running on 8 nodes with 16 CPUs because the communication between nodes required by MPI can be very slow depending on the cluster. Option 2 may be faster if irace generates more configurations per iteration than the number of CPUs of a single node. However, depending on the configuration of your cluster, requesting many CPUs may require waiting in the queue a long time. Option 3 may be the slowest since irace has to check the queue frequently. However, irace will start running experiments as soon as 1 CPU is available, thus option 3 may actually finish earlier than the other options if there is always some CPUs available in the cluster but the queue for requesting many CPUs at once is very long.

As a rule-of-thumb, if you only have access to a single machine, then you only need option 1. If you have access to a computing cluster with multiple machines, then use option 1 with the maximum number of CPUs that a single node has in your computing cluster. If that number is 64 or more, it should be enough unless a single run of irace evaluates thousands of configurations. Otherwise, investigate option 2. If option 2 does not work, then investigate option 3.

7 Testing (Validation) of configurations

Once the tuning process is finished, **irace** returns a set of configurations corresponding to the elite configurations at the end of the run, ordered from best to worst. In order to evaluate the generality of these configurations without looking at their performance on the training set, **irace** offers the possibility of evaluating these configurations on a test instance set, typically different from the training set used during the tuning phase. These evaluations will use the same settings for parallel execution, targetRunner and targetEvaluator.

The test instances can be specified by the options testInstancesDir and/or testInstancesFile, or by setting directly the variable scenario$testInstances. These options behave similarly to their counterparts for the training instances (Section 5.4). In particular, each test instance is assigned a different seed in the same way as done for the training instances. In principle, **irace** evaluates each configuration on each testing instance just once, because evaluating one run on n instances is always better than evaluating n' runs on n/n' instances [2]. However, if the number of instances is limited, one can always duplicate instances as needed in the testInstancesFile, and **irace** will assign a different random seed to each instance. An example of the output produced by **irace** when testing is shown in Fig. 5.

The options testNbElites and testIterationElites control which configurations are evaluated during the testing phase. In particular, setting testIterationElites = 1 will test not only the final set of elite configurations (those returned at the end of the training phase), but also the set of elites at the end of each race (iteration). The option testNbElites limits the maximum number of configurations considered within each set. Some examples:

- testIterationElites = 0; testNbElites = 1 means that only the best configuration found during the run of **irace**, the final best, will be used in the testing phase.
- testIterationElites = 1; testNbElites = 1 will test, in addition to the final best, the best configuration found at each iteration.
- testIterationElites = 1; testNbElites = 2 will test the two best configurations found at each iteration, in addition to the final best and second-best configurations.

The testing can be also (re-)executed at a later time by using the following R command (but you may need to override testNbElites and testIterationElites):

```
testing_fromlog(logFile = "./irace.Rdata", testNbElites = 1)
```

The above line will load the scenario setup from logFile to perform the testing. The testing results will be stored in the R object iraceResults$testing, which is saved in the file specified by scenario$logFile. The structure of the object is described in Section 9.2. For examples on how to analyse the results see Section 9.3.

Another alternative is to test a specific set of configurations using the command-line option --only-test as follows:

```
irace --only-test configurations.txt
```

where configurations.txt has the same format as the set of initial configurations (Section 5.5).

8 Recovering irace runs

Problems like power cuts, hardware malfunction or the need to use computational power for other tasks may occur during the execution of **irace**, terminating a run before completion. At the end of each iteration, **irace** saves an R data file (logFile, by default "./irace.Rdata") that not only contains information about the tuning progress (Section 9.2), but also internal information that allows recovering an incomplete execution.

To recover an incomplete **irace** run, set the option recoveryFile to the log file previously produced, and **irace** will continue the execution from the last saved iteration. The state of the random generator is saved and loaded, therefore, as long as the execution is continued in the same machine, the obtained results will be exactly the same as executing **irace** in one step (external factors, such as CPU load and disk caches, may affect the target algorithm and that may affect the results). You can specify the recoveryFile from the command-line or from the scenario file, and execute **irace** as described in Section 4. For example, from the command-line use:

```
irace --recovery-file "./irace-backup.Rdata"
```

When recovering a previous run, **irace** will try to save data on the file specified by the logFile option. Thus, you must specify different files for logFile and recoveryFile. Before recovering, we strongly advise to rename the saved R data file as in the example above, which uses "irace-backup.Rdata".

Do not change anything in the log file or the scenario file before recovering, as it may have unexpected effects on the recovered run of **irace**. In case of doubt, please contact us first (Section 13). In particular, it is not possible to continue a run of **irace** by recovering with a larger budget. Results will **not** be the same as running **irace** from the start with the largest budget. An alternative is to use the final configurations from one run as the initial configurations of a new run.

If your scenario uses targetEvaluator (Section 5.3) and targetEvaluator requires files created by targetRunner, then recovery will fail if those files are not present in the execDir directory. This can happen, for example, if you recover from a different directory than the one from which irace was initially executed, or when execDir is set to a temporary directory for every irace run. Thus, you need to copy the contents of the previous execDir into the new one.

9 Output and results

During its execution, **irace** prints information about the progress of the tuning in the standard output. Additionally, after each iteration, an R data file is saved (logFile option) containing the state of **irace**.

9.1 Text output

Figure 6 shows the output, up to the end of the first iteration, of a run of elitist **irace** applied to the **ACOTSP** scenario with 1000 evaluations as budget.

First, **irace** gives the user a warning informing that it has found a file with the default scenario filename and it will use it. Then, general information about the selected **irace** options is printed:

- nbIterations indicates the minimum number of iterations **irace** has calculated for the scenario. Depending on the development of the tuning the final iterations that are executed can be more.
- minNbSurvival indicates the minimum number of alive configurations that are required to continue a race. When less configurations are alive the race is stopped and a new iteration begins.
- nbParameters is the number of parameters of the scenario.
- seed is the number that was used to initialize the random number generator in **irace**.
- confidence level is the confidence level of the statistical test.
- budget is the total number of evaluations available for the tuning.

```
...

# Testing of elite configurations: 5
# Testing iteration configurations: TRUE
# 2023-10-01 13:35:17 BST: Testing configurations (in no particular order): 2 29 3 20 50 34 47 86 74 111 106 92 123 134 130 119
    algorithm localsearch  alpha   beta    rho ants nnls     q0 dlb rasrank elitistants
2         acs           3 1.1275 3.3469 0.6471   36   43 0.9053   0      NA          NA
29        ras           3 4.5152 5.6811 0.7818   11   43     NA   0      13          NA
3         eas           3 3.7246 5.5153 0.5998   75   29     NA   1      NA          91
20        acs           2 1.4734 0.1808 0.4304    5   39 0.5665   1      NA          NA
50        ras           3 2.2804 4.5294 0.5508   18   32     NA   1      12          NA
34        acs           3 2.2044 2.4923 0.7243   12   29 0.4628   0      NA          NA
47        acs           2 1.7046 6.3908 0.3256    5   36 0.2288   0      NA          NA
86        ras           3 3.9808 3.4401 0.2191   20   25     NA   1      15          NA
74       mmas           3 1.1135 0.6356 0.3765   11   33     NA   1      NA          NA
111       acs           2 1.2881 6.4311 0.5859   10   23 0.2126   0      NA          NA
106       ras           3 3.2756 5.5035 0.9277   18   32     NA   0      10          NA
92        acs           2 1.6234 9.2153 0.1101    8   40 0.3412   0      NA          NA
123       acs           2 1.0584 5.8121 0.5587   10   35 0.2575   0      NA          NA
134       acs           2 1.6144 7.8972 0.1999    8   33 0.1415   1      NA          NA
130       acs           2 1.5899 6.9391 0.5529    8   26 0.3796   0      NA          NA
119       acs           2 1.4432 6.4746 0.4582    5   28 0.1931   0      NA          NA
# 2023-10-01 13:48:55 BST: Testing results (column number is configuration ID in no particular order):
         seeds        2       29        3       20       50       34       47       86       74      111      106       92      123
1t  1385446146 23609115 23432130 23400390 23403445 23383391 23439897 23380458 23455774 23355888 23382496 23417159 23404000 23492199 23425
2t  1396979195 23327035 23208321 23186931 23288822 23181225 23228238 23265383 23201401 23279356 23188848 23171998 23314132 23244891 23207
3t   448912041 23111373 23246027 23092265 23130590 23016527 23056027 23070519 23084400 23041364 23118905 23063197 23115280 23069064 23094
4t   732530909 23128507 23159085 23119773 23290945 23063257 23095963 23110872 23151621 23053285 23133797 23139894 23148536 23101345 23084
5t   947545849 23340533 23347633 23298513 23274844 23236071 23274590 23246473 23292452 23232446 23242908 23264111 23219377 23258853 23234
6t   195435663 23532942 23470894 23496016 23533573 23423201 23538541 23526779 23500842 23497417 23525281 23464461 23510384 23454321 23478
7t   798649446 23440321 23430267 23331305 23422861 23375048 23454558 23491515 23405251 23509688 23386282 23508539 23438775 23432504 23438
8t  1692971486 23376183 23262794 23300201 23286312 23258955 23368034 23301552 23263607 23231022 23254721 23271971 23236607 23227995 23280
9t   896414478 23375100 23296796 23375905 23293303 23293631 23428815 23292599 23376861 23314826 23312995 23369543 23333975 23355279 23352
10t 1371337352 23222214 23163714 23110039 23126033 23089358 23198436 23080692 23146608 23106011 23092382 23087101 23108507 23096788 23126
# 2023-10-01 13:48:55 BST: Finished testing
```

Figure 5: Sample text output of **irace** when evaluating on test instances.

- `time budget` is the maximum execution time available for the tuning.
- `mu` is a value used for calculating the minimum number of iterations.
- `deterministic` indicates if the target algorithm is assumed to be deterministic.

  At each iteration, information about the progress of the execution is printed as follows:

- `experimentsUsedSoFar` is the number of experiments from the total budget that have been used up to the current iteration.
- `timeUsed` is the execution time used so far in the experiments. Only available when reported in the `targetRunner` (activate it with the `maxTime` option).
- `remainingBudget` is the number of experiments that have not been used yet.
- `timeEstimate` estimation of the mean execution time. This is used to calculate the remaining budget when `maxTime` is used.
- `currentBudget` is the number of evaluations **irace** has allocated to the current iteration.
- `nbConfigurations` is the number of configurations **irace** will use in the current iteration. In the first iteration, this number of configurations include the initial configurations provided; in later iterations, it includes the elite configurations from the previous iterations.

After the iteration information, a table shows the progress of the iteration execution. Each row of the table gives information about the execution of an instance in the race. The first column contains a symbol that describes the results of the statistical test:

`|x|` No statistical test was performed for this instance. The options `firstTest` and `eachTest` control on which instances the statistical test is performed.

`|-|` Statistical test performed and configurations have been discarded. The column `Alive` gives an indication of how many configurations have been discarded.

`|=|` Statistical test performed and no configurations have been discarded. This means **irace** needs to evaluate more instances to identify the best configurations.

`|!|` This indicator exists only for the elitist version of **irace**. It indicates that the statistical test was performed and some elite configurations appear to show bad performance and could be discarded but they are kept because of the elitist rules. See option `elitist` in Section 11 for more information.

Other columns have the following meaning:

`Instance`: Index of `(instance,seed)` pair executed. This number corresponds to the row in the data frame returned by `get_instanceID_seed_pairs()`. See Section 9.2 for more information. This index is different from the instance ID passed to `targetRunner`.

`Bound`: Only when `capping` is enabled. Execution time used as bound for the execution of new candidate configurations.

`Alive`: Number of configurations that have not been discarded after the statistical test was performed.

`Best`: ID of the best configuration according to the instances seen so far in this race (i.e., not including previous iterations).

```
#------------------------------------------------------------------------------
# irace: An implementation in R of (Elitist) Iterated Racing
# Version: 4.0.3037934
# Copyright (C) 2010-2020
# Manuel Lopez-Ibanez     <manuel.lopez-ibanez@manchester.ac.uk>
# Jeremie Dubois-Lacoste
# Leslie Perez Caceres    <leslie.perez.caceres@ulb.ac.be>
#
# This is free software, and you are welcome to redistribute it under certain
# conditions.  See the GNU General Public License for details. There is NO
# WARRANTY; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
#
# irace builds upon previous code from the race package:
#     race: Racing methods for the selection of the best
#     Copyright (C) 2003 Mauro Birattari
#------------------------------------------------------------------------------
# installed at: /home/manu/R/x86_64-pc-linux-gnu-library/4.1/irace
# called with: --parallel 2
# 2024-11-25 19:54:46 GMT: Reading parameter file '/home/manu/work/irace/git/devel-examples/vignette-example/parameters.txt'.
# 2024-11-25 19:54:46 GMT: 1 expression(s) specifying forbidden configurations read.
# 2024-11-25 19:54:46 GMT: Read 1 configuration(s) from file '/home/manu/work/irace/git/devel-examples/vignette-example/default.txt'
# 2024-11-25 19:54:46 GMT: Initialization
# Elitist race
# Elitist new instances: 1
# Elitist limit: 2
# nbIterations: 5
# minNbSurvival: 5
# nbParameters: 11
# seed: 687542627
# confidence level: 0.95
# budget: 1000
# mu: 5
# deterministic: FALSE

# 2024-11-25 19:54:46 GMT: Iteration 1 of 5
# experimentsUsedSoFar: 0
# remainingBudget: 1000
# currentBudget: 200
# nbConfigurations: 33
# Markers:
     x No test is performed.
     c Configurations are discarded only due to capping.
     - The test is performed and some configurations are discarded.
     = The test is performed but no configuration is discarded.
     ! The test is performed and configurations could be discarded but elite configurations are preserved.
     . All alive configurations are elite and nothing is discarded.

+-+-----------+-----------+-----------+----------------+-----------+--------+-----+----+------+
| |   Instance|      Alive|       Best|       Mean best| Exp so far|  W time|  rho|KenW|  Qvar|
+-+-----------+-----------+-----------+----------------+-----------+--------+-----+----+------+
|x|          1|         33|          3|     32651716.00|         33|00:01:34|   NA|  NA|    NA|
|x|          2|         33|          3|     32758597.50|         66|00:01:35|+0.96|0.98|0.0011|
|x|          3|         33|          3|     32897333.67|         99|00:01:43|+0.96|0.97|0.0010|
|x|          4|         33|          3|     32873388.50|        132|00:01:34|+0.97|0.97|0.0018|
|-|          5|          3|          3|     32913189.20|        165|00:01:33|+0.40|0.52|0.1580|
+-+-----------+-----------+-----------+----------------+-----------+--------+-----+----+------+
Best-so-far configuration:           3    mean value:      32913189.20
Description of the best-so-far configuration:
  .ID. algorithm localsearch  alpha   beta   rho ants nnls q0 dlb rasrank elitistants time .PARENT.
3    3       ras           3 2.4626 3.3474 0.794    6    7 NA   0      86          NA    5       NA

# 2024-11-25 20:02:48 GMT: Elite configurations (first number is the configuration ID; listed from best to worst according to the sum of
   algorithm localsearch  alpha   beta    rho ants nnls     q0 dlb rasrank elitistants time
3        ras           3 2.4626 3.3474 0.7940    6    7     NA   0      86          NA    5
16       acs           3 0.9001 7.7224 0.2372   16   10 0.7089   0      NA          NA    5
25       ras           3 4.8064 5.5349 0.0825   45   49     NA   1      65          NA    5
# 2024-11-25 20:02:48 GMT: Iteration 2 of 5
# experimentsUsedSoFar: 165
```

Figure 6: Sample text output of **irace**.

`Mean best`: Mean cost value of the best configuration across the instances seen so far in this race (not globally). Equivalent to the concept of "iteration-best" in other algorithms.

`Exp so far`: Number of experiments performed so far.

`W time`: Wall-clock time spent on this instance.

`rho`, `KenW`, and `Qvar`: Spearman's rank correlation coefficient rho, Kendall's concordance coefficient W, and a variance measure described in [16], respectively, of the configurations across the instances evaluated so far in this iteration. These measures evaluate how consistent is the performance of the configurations across the instances. Values close to 1 for `rho` and `KenW` and values close to 0 for `Qvar` indicate that the scenario is highly homogeneous. For heterogeneous scenarios, we provide advice in Section 10.5.

Finally, **irace** outputs the best configuration found and a list of the elite configurations. The elite configurations are configurations that did not show statistically significant difference during the race; they are ordered according to their mean performance on the executed instances.

9.2 R data file (`logFile`)

The R data file created by **irace** (by default as `irace.Rdata`, see option `logFile`) contains an object called `iraceResults`. You can load this file in the R console with:

```
logfile <- system.file(package="irace", "exdata", "irace-acotsp.Rdata", mustWork=TRUE)
iraceResults <- read_logfile(logfile)
```

The `iraceResults` object is a list, and the elements of a list can be accessed in R by using the `$` or `[[]]` operators:

```
> iraceResults$irace_version

[1] "4.0.3037934"
```

The `iraceResults` list contains the following elements:

- `scenario`: The scenario R object containing the **irace** options used for the execution. See Section 11 and the help of the **irace** package; open an R console and type: `?defaultScenario`. See Section 11 for more information.

- `parameters`: The parameters R object containing the description of the target algorithm parameters. See Section 5.1.

- `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 final column (`.PARENT.`) is the identifier of the configuration from which the current configuration was sampled; and the remaining columns correspond to the parameter values; each column is named as the parameter name specified in the parameter object.

  ```
  > head(iraceResults$allConfigurations)
  ```

```
  .ID. algorithm localsearch  alpha   beta    rho ants nnls     q0
1    1        as           0 1.0000 1.0000 0.9500   10   NA     NA
2    2        as           1 4.9626 8.3474 0.2990   28   30     NA
3    3       ras           3 2.4626 3.3474 0.7940    6    7     NA
4    4       acs           0 3.7126 5.8474 0.5465   13   NA 0.8964
5    5      mmas           2 1.2126 0.8474 0.0515   60   18     NA
6    6       ras           1 0.5876 4.5974 0.9178   19   36     NA
   dlb rasrank elitistants time .PARENT.
1 <NA>      NA          NA    5       NA
2    1      NA          NA    5       NA
3    0      86          NA    5       NA
4 <NA>      NA          NA    5       NA
5    1      NA          NA    5       NA
6    1      24          NA    5       NA
```

- `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.`).

```
> print(iraceResults$allElites)

[[1]]
[1]  3 16 25

[[2]]
[1]  3 16 25 54

[[3]]
[1] 68 87

[[4]]
[1] 117  68  91 106

[[5]]
[1] 147 119 117 136 135

[[6]]
[1] 158 163 162

[[7]]
[1] 158 163 162 165

[[8]]
[1] 158 163 162 165 170

[[9]]
[1] 158 163
```

The configurations are ordered by mean performance, that is, the ID of the best configuration corresponds to the first ID. To obtain the values of the parameters of all elite configurations found by **irace** use:

```
> logfile <- system.file(package="irace", "exdata", "irace-acotsp.Rdata", mustWork=TRUE)
> getFinalElites(logfile, n = 0)

    .ID. algorithm localsearch  alpha   beta    rho ants nnls q0 dlb
158  158       ras           3 2.3916 1.0655 0.7257    9   10 NA   1
163  163        as           3 3.0685 2.0363 0.7901    9   12 NA   1
    rasrank elitistants time .PARENT.
158      82          NA    5      119
163      NA          NA    5      136
```

- `iterationElites`: A vector containing the best candidate configuration ID of each iteration. The best configuration found corresponds to the last one of this vector.

```
> print(iraceResults$iterationElites)

[1]   3   3  68 117 147 158 158 158 158
```

  One can obtain the full configuration with:

```
> last <- length(iraceResults$iterationElites)
> id <- iraceResults$iterationElites[last]
> getConfigurationById(iraceResults, ids = id)

    .ID. algorithm localsearch  alpha   beta    rho ants nnls q0 dlb
158  158       ras           3 2.3916 1.0655 0.7257    9   10 NA   1
    rasrank elitistants time .PARENT.
158      82          NA    5      119
```

- `rejectedConfigurations`: A vector containing the rejected configurations IDs. These correspond to configurations that produced failed executions and were ignored by **irace** during the configuration process. See Section 10.8 to enable the detection of such configurations.

- `experiments`: A matrix with configurations as columns and instances as rows. Column names correspond to the internal identifier of the configuration (`allConfigurations$.ID.`). The results of a particular configuration can be obtained using:

```
> # As an example, we use the best configuration found
> best_config <- getFinalElites(iraceResults, n = 1)
> best_id <- as.character(best_config$.ID.)
> # Obtain the results of the best configuration
> all_exp <- iraceResults$experiments[, best_id]
> # all_exp is a vector and names(all_exp) is the (instance,seed) index.
> all_exp
```

```
       1        2        3        4        5        6        7
32543016 32674165 33071851 32581927 32888629 33048372 32468751
       8        9       10       11       12       13       14
32994695 32653004 32527692 32940567 32674577 32668525 33071355
      15
32957205

> # Obtain the results of the first and best configurations
> all_exp <- iraceResults$experiments[, c("1", best_id)]
> # all_exp is a matrix: colnames(all_exp) is configurationID and
> # rownames(all_exp) is the (instance,seed) index.
> all_exp

          1      158
1  40755522 32543016
2  41680348 32674165
3  41364641 33071851
4  40664730 32581927
5  41007962 32888629
6        NA 33048372
7        NA 32468751
8        NA 32994695
9        NA 32653004
10       NA 32527692
11       NA 32940567
12       NA 32674577
13       NA 32668525
14       NA 33071355
15       NA 32957205
```

When a configuration was not executed on an instance, its value is `NA`. A configuration may not be executed on an instance because: (1) it was not created yet when the instance was used, or (2) it was discarded by the statistical test and not executed on subsequent instances, or (3) the race terminated before this instance was considered.

Row names correspond to the row index of the `(instance,seed)` pairs in the data frame returned by `get_instanceID_seed_pairs()`. The instanceID and seed used for a particular experiment can be obtained with:

```
> # As an example, we get instanceID, seeds and instances of the experiments
> # of the best configuration.
> # We could get the indexes of the instances on which at least one
> # configuration was executed:
> pair_index <- which(apply(!is.na(all_exp), 1L, any))
> # or the instances on which all configurations were executed:
> pair_index <- which(apply(!is.na(all_exp), 1L, all))
> # but in this example we get the indexes of the instances executed for
> # the best configuration.
> pair_index <- which(!is.na(all_exp[, best_id]))
> instanceID <- get_instanceID_seed_pairs(iraceResults)[["instanceID"]][pair_index]
> # or get the seeds
> get_instanceID_seed_pairs(iraceResults)[["seed"]][pair_index]

 [1]  120344916 1498426593 1324006684  156117387 2123556176  975149182
 [7]  657774990 1688886839 1722597766  545710096  685987118  654417054
[13] 1203404683 1011189918 2070570017

> # or obtain the actual instances.
> iraceResults$scenario$instances[instanceID]

 [1] "./instances/2000-539.tsp" "./instances/2000-532.tsp"
 [3] "./instances/2000-533.tsp" "./instances/2000-637.tsp"
 [5] "./instances/2000-437.tsp" "./instances/2000-22.tsp"
 [7] "./instances/2000-630.tsp" "./instances/2000-224.tsp"
 [9] "./instances/2000-923.tsp" "./instances/2000-221.tsp"
[11] "./instances/2000-440.tsp" "./instances/2000-740.tsp"
[13] "./instances/2000-735.tsp" "./instances/2000-131.tsp"
[15] "./instances/2000-126.tsp"

> # If the instances are of atomic type (integers, floating-point numbers or
> # character strings), the above is similar to:
> get_instanceID_seed_pairs(iraceResults, index = pair_index, instances=TRUE)

    instanceID       seed                 instance
         <int>      <int>                   <char>
 1:        119  120344916 ./instances/2000-539.tsp
 2:        112 1498426593 ./instances/2000-532.tsp
 3:        113 1324006684 ./instances/2000-533.tsp
 4:        137  156117387 ./instances/2000-637.tsp
 5:         97 2123556176 ./instances/2000-437.tsp
 6:         31  975149182  ./instances/2000-22.tsp
 7:        130  657774990 ./instances/2000-630.tsp
 8:         25 1688886839 ./instances/2000-224.tsp
 9:        183 1722597766 ./instances/2000-923.tsp
10:         22  545710096 ./instances/2000-221.tsp
11:        100  685987118 ./instances/2000-440.tsp
12:        160  654417054 ./instances/2000-740.tsp
13:        155 1203404683 ./instances/2000-735.tsp
14:         11 1011189918 ./instances/2000-131.tsp
15:          6 2070570017 ./instances/2000-126.tsp
```

- `experimentLog`: A matrix with columns `iteration,instance,configuration`. This matrix contains the log of all the experiments that **irace** performs during its execution. The `instance` column refers to the index of the data frame returned by `get_instanceID_seed_pairs()`. When `capping` is enabled a column `bound` is added to log the execution bound applied for each execution.

- `softRestart`: A logical vector that indicates if a soft restart was performed on each iteration. If `FALSE`, then no soft restart was performed. See option `softRestart` in Section 11.
- `state`: A list that contains the state of **irace**, the recovery (Section 8) is done using the information contained in this object. The probabilistic model of the last elite configurations can be found here by doing:

```
> # As an example, we get the model probabilities for the
> # localsearch parameter.
> iraceResults$state$model["localsearch"]

$localsearch
$localsearch$`158`
[1] 4.927852e-05 4.927852e-05 4.927852e-05 9.998522e-01

$localsearch$`163`
[1] 4.927852e-05 4.927852e-05 4.927852e-05 9.998522e-01

$localsearch$`162`
[1] 4.927852e-05 4.927852e-05 4.927852e-05 9.998522e-01

$localsearch$`165`
[1] 4.927852e-05 4.927852e-05 4.927852e-05 9.998522e-01

> # The order of the probabilities corresponds to:
> iraceResults$scenario$parameters$domains$localsearch

[1] "0" "1" "2" "3"
```

  The example shows a list that has one element per elite configuration (ID as element name). In this case, `localsearch` is a categorical parameter and it has a probability for each of its values.

- `testing`: A list that contains the testing results. The list contains the following elements:
  - `experiments`: Matrix of experiments in the same format as the `iraceResults$experiments` matrix. The column names indicate the candidate configuration identifier and the row names contain the name of the instances.

```
> # Get the results of the testing
> iraceResults$testing$experiments

           3       16       25       54       68       87      117
1t  33038703 33210450 33197215 33123566 32910890 32833215 32790987
2t  32826546 32868347 32936877 32869167 32624174 32689448 32641747
3t  33263371 33361627 33367816 33346995 33112700 33234481 33127815
4t  32989521 33132403 33092369 33125274 32900111 32878571 32887939
5t  32838591 33067819 33220556 33134342 32801520 32915718 32848386
6t  32762995 33040031 32833200 32906000 32855991 32871604 32576206
7t  32988662 33266114 33312415 33072790 33007492 33114622 32929590
```

```
8t    33233782 33173476 33263436 33111162 33133010 32931182 32791745
9t    33105732 33189429 33276723 33233416 33009362 33048057 32936535
10t   32896170 32964601 33066887 33037373 32776358 32892243 32822586
11t   32923676 32939408 32978789 32910829 32746586 32834798 32818140
12t   32784366 33033690 33077305 33092155 32931737 32793969 32854368
13t   33206700 33387763 33436444 33365937 33328987 33171934 33189468
14t   33060973 33218435 33184801 33106470 33029238 32960623 32905906
15t   33166941 33220666 33280307 33253955 32971439 32930800 32937933
16t   32797571 32956667 32916825 32862351 32621355 32623956 32497202
17t   32931334 33128016 33098168 33088625 32868160 32810604 32890631
18t   32831688 32899725 32960081 32909228 32656821 32633442 32590724
19t   32944271 33159671 33079853 33104326 32917181 32875636 32860227
20t   32966988 32977020 33140445 33098661 32793694 32917364 32724034
21t   33168840 33076254 33094865 33122180 32935360 32908403 32847629
22t   32861107 33186847 33126009 33027808 32834772 32843447 32861419
23t   32796360 32909290 32873717 32835387 32630655 32707085 32632067
24t   32880210 33026120 33104366 33120366 32887775 32916351 32780689
25t   32970467 33288737 33282554 33269982 32910512 33010224 32851243
26t   32992892 33146513 33222813 33192971 32922817 33122902 32830006
27t   33264382 33357116 33337360 33313169 33118978 33109715 33004475
28t   32901992 32906898 33119831 33023159 32784069 32879636 32705243
29t   32972637 33135352 33131620 33152147 32987724 32829349 32757116
30t   33040299 33075287 33131918 33087965 32808668 32739020 32797403
31t   32992349 33087780 33201967 33113209 32772137 32926380 32789415
32t   33155387 33055083 33158723 33155576 33030373 32976933 32909107
33t   32825563 32885651 32928450 32973675 32611243 32790726 32598400
34t   32813990 33000143 33075408 32962461 32670285 32605058 32703468
35t   32830554 33006227 32993599 32916346 32670828 32742699 32615125
36t   32973931 33077502 33008674 33029675 32859035 32829651 32729024
37t   33177011 33230178 33307832 33168250 32992385 33088864 32828793
38t   33374921 33441677 33517977 33506821 33286319 33271704 33082391
39t   32722390 32856025 32856881 32808252 32633047 32513872 32519392
40t   33166452 33241530 33240870 33344384 33000623 33107487 32925090
41t   33334294 33385797 33370637 33325028 33081916 33073207 33105467
42t   32715319 33059430 33045491 33035400 32870719 32923965 32711042
43t   32806128 33093236 33004727 33050147 32938023 32706779 32646345
44t   32902925 33071485 33105353 33006103 32806905 32867805 32714588
45t   33279920 33380374 33333662 33403257 33122175 33273456 33069725
46t   33090806 33287365 33333530 33270710 32884761 33032582 33002168
47t   33010731 33291748 33317082 33329815 32960273 33121431 32889066
48t   32650053 32833708 32946704 32897712 32613550 32635809 32582945
49t   32573296 32872127 32702523 32745193 32583567 32576887 32430751
50t   33086491 33173417 33223544 33229069 32924439 33072724 32894041
51t   33044473 33126681 33275879 33237276 33028077 33000132 32910956
52t   32829314 32987275 32965841 32966570 32719768 32708909 32620431
53t   32948989 33123576 33159203 33160675 32914623 33121419 32858773
54t   32879637 33065249 33020062 32958108 32768367 32737964 32702370
55t   33138112 33133080 33140868 33113637 32930177 33003860 32931502
```

```
56t  33026761 33120873 33152542 33061267 32784404 32842912 32808567
57t  32914391 33119818 33171633 33196985 32896643 32897154 32844778
58t  33202414 33223587 33197887 33279478 33099821 33024787 33027906
59t  32998813 33350453 33329382 33306953 33095095 33117461 33002269
60t  32931446 33175929 33129972 33191595 32915230 32907851 32724218
61t  33066792 33135827 33162255 33111192 32775148 32943134 32861208
62t  33368475 33227646 33447877 33389249 33050255 33038428 33151186
63t  32845603 33157916 33133823 33119567 32904939 32848892 32806365
64t  33517097 33500309 33624037 33599192 33300840 33356357 33255860
65t  32877149 33046097 32981391 32969449 32792838 32707896 32658949
66t  33288242 33371259 33359833 33350435 32988403 32962891 33004271
67t  33010654 32986762 32952068 32840913 32637351 32771170 32716152
68t  33043204 33231466 33089910 33212922 32934373 32802236 32814023
69t  32966726 33196212 33215590 33141456 33040707 33031561 32915592
70t  33241122 33332303 33288091 33311883 32983633 32930417 33034328
71t  32639109 32828977 33014768 32990584 32797103 32803064 32706630
72t  32978266 33013911 33058820 32946320 32765124 32799630 32689662
73t  32792737 32889597 32973399 32983784 32722627 32795664 32725089
74t  33030645 33177413 33204897 33095335 32820730 32848311 32833608
75t  33337559 33367042 33351792 33348168 33171014 33153205 33025102
76t  33043380 33272295 33206835 33066524 32909105 32978007 32809090
77t  33097223 33139296 33154939 33148511 32853313 32839742 32873862
78t  32719675 33032572 33064998 33007466 32714289 32968271 32691012
79t  32377650 32655602 32737639 32792339 32518971 32530025 32435940
80t  32640804 32735488 32835398 32890935 32607919 32651514 32734211
81t  32673364 32815739 32717598 32685503 32470187 32448802 32430571
82t  32881583 33066504 33070451 32998880 32725392 32917111 32817007
83t  32979339 33149979 33061518 33151660 32861296 32938218 32887838
84t  32656106 32682589 32748761 32667712 32546906 32606089 32409978
85t  32660589 32725338 32829617 32831603 32435481 32578347 32332162
86t  32914581 33020564 33096555 33107840 32875223 32810018 32819913
87t  32998095 33130354 33276270 33014762 32875607 32898124 32938137
88t  33240635 33364758 33393929 33375514 33180213 33188048 33009342
89t  33157922 33524016 33509024 33510589 33208448 33224163 33280988
90t  32950195 33029385 33100773 33030184 32867572 32937339 32749579
91t  33065825 33198265 33182151 33102457 32901548 33001100 32964457
92t  33348838 33516048 33477660 33523286 33193907 33340254 33148893
93t  32988035 33143746 33139153 33199519 32903315 32998793 32904211
94t  32969392 33190780 33128433 33144811 32963125 33012925 32908914
95t  32934066 33094589 33119384 33135110 32876757 32800639 32790952
96t  33153265 33353927 33316234 33370563 33107495 33229517 33095068
97t  33038071 33277081 33229092 33222150 32906559 33102724 32899936
98t  32709146 32828546 32895890 32916259 32557845 32551376 32521112
99t  32863968 32898042 32990796 32883133 32672866 32634488 32668269
100t 32808211 33137544 33159815 33110727 32826357 32993666 32682907
101t 32886670 33104764 33000295 32891135 32842584 32825360 32804749
102t 32708971 32934760 32866062 32808126 32683548 32679706 32613437
103t 32817994 32910705 32943640 32873556 32655571 32771822 32654815
```

```
104t 33367932 33492783 33430770 33410862 33284970 33249374 33129912
105t 32632079 32853085 32830541 32826516 32482029 32422439 32487784
106t 32945815 33109700 33047757 33079102 32778968 32806104 32718928
107t 33222262 33209377 33107388 33097237 32907531 32899564 32799514
108t 32909035 33032588 33045855 32999718 32861210 32901131 32827971
109t 33100920 33094295 33117824 32978946 32817514 32958048 32793739
110t 32832632 33006678 33110667 33020814 32856948 32848775 32715253
111t 33067801 33133541 33118571 33143746 32908853 32851856 32784727
112t 32910256 32951914 33027311 32880226 32602262 32852930 32607999
113t 33280715 33554227 33446125 33385224 33229003 33183065 32994570
114t 33372779 33613118 33580073 33472133 33225602 33286451 33187125
115t 32956748 33049375 33180206 33184058 32944157 32864878 32810654
116t 33257508 33416215 33487975 33459103 33137831 33229032 33123347
117t 32840343 32744544 32732976 32849247 32584549 32642050 32484718
118t 33100478 33370382 33314101 33274755 32882431 32965807 32939087
119t 32979579 33269056 33176927 33294069 33013331 32891909 32935983
120t 32893163 33182227 33042117 33061984 32779801 32867529 32774869
121t 32661818 32768535 32855872 32801143 32547702 32658807 32546278
122t 33008219 33159658 33147563 33179819 32865271 32985154 32834192
123t 33229974 33349775 33395696 33412518 33221314 33131575 33127063
124t 32945119 33202046 33249450 33184052 33005633 32958010 32871509
125t 33062630 33085362 33227536 33200998 32925894 33004185 32886308
126t 32969854 33205449 33196261 33202417 32954887 32975573 32874063
127t 33072225 33234621 33256021 33283264 32979079 33016886 32894380
128t 33250657 33307606 33264725 33326357 33053477 32983349 32980378
129t 33195408 33421073 33388995 33362908 33110045 33110454 33025290
130t 32999460 33285144 33182633 33289205 33020297 32949768 32867574
131t 33181631 33324134 33264146 33211620 33043025 33120526 32948558
132t 32856182 33047368 33047318 32896389 32686715 32742023 32674331
133t 33303301 33412585 33419575 33382373 33145784 33187694 33147338
134t 32704421 32745005 32770211 32730781 32618242 32791243 32434867
135t 32621650 32853204 32844095 32775135 32631812 32527810 32521573
136t 33036455 33109400 33147747 33142926 32893702 32825886 32726888
137t 33197362 33294283 33349873 33258334 33100593 33166070 32849203
138t 32622428 32798628 32898344 32866698 32756313 32607034 32545088
139t 32819128 33042721 33003247 32936899 32734722 32685264 32786556
140t 32692704 32729559 32765001 32772786 32525272 32620589 32467692
141t 33038583 33008484 33138682 33200915 32934058 32937635 32873029
142t 32822954 33090737 33072107 32996011 32594869 32819207 32593961
143t 32610929 32756433 32765247 32809350 32437536 32574432 32439756
144t 32972265 33272663 33276787 33216307 32922134 33052316 33029162
145t 33230162 33327280 33275613 33268071 33120188 33214944 33047466
146t 32660466 32772474 32655077 32703160 32452790 32516961 32387594
147t 33200222 33371496 33337618 33391088 33078038 33208230 33110577
148t 33152008 33422758 33411071 33328005 33034980 33057225 33160963
149t 33149101 33150217 33197595 33185211 33014593 32885657 32962924
150t 33071071 33172814 33253317 33224100 32904526 33074232 32818847
151t 33555136 33693372 33707389 33612554 33440226 33403522 33322940
```

```
152t 33000379 33225077 33270575 33284530 32979879 33046265 32886167
153t 32623641 32844813 32887646 32807520 32561937 32712478 32612664
154t 32842465 32917121 33003116 32979419 32723740 32916313 32708746
155t 33274665 33439732 33375514 33398589 33142881 33225290 33152595
156t 32891125 32956745 32921473 32953759 32788841 32833203 32631707
157t 33052073 33216486 33123206 33158517 32871766 33017189 32954924
158t 32811498 32957305 33030752 32884647 32783181 32794616 32594519
159t 33120008 33186903 33224866 33199550 32983815 32976536 32993818
160t 33547061 33739329 33564394 33647749 33300094 33424670 33265299
161t 32440723 32740006 32748421 32720407 32545200 32603553 32472464
162t 32744212 32957574 32964782 32971271 32687183 32636958 32660433
163t 33333804 33379692 33394507 33476752 33228317 33242077 33113879
164t 33327490 33439049 33552236 33531273 33245176 33374448 33189962
165t 32609445 32672470 32767928 32706039 32448947 32429082 32476826
166t 32639696 32870702 32780715 32781584 32606998 32588705 32611115
167t 32783247 32982443 33017018 32957110 32834755 32761473 32678150
168t 32847575 32999627 32911270 32988530 32694783 32798324 32647867
169t 33139259 33249678 33317381 33331821 33078915 33159710 32968459
170t 32854588 33192591 33108997 33076374 32870335 32938895 32934416
171t 33246621 33252046 33285742 33315980 33107991 32997495 32959633
172t 32961549 33123816 33056776 33038321 32759464 32822281 32806865
173t 32996179 33013922 33031152 33068796 32746939 32794870 32786721
174t 33226836 33415408 33347776 33323141 33096926 33094912 33018274
175t 33111220 33206291 33156534 33182978 32933010 32991399 32915564
176t 33249636 33250036 33329737 33253722 32989223 32897345 33009985
177t 33008891 33084590 33201920 33156212 32797028 32934242 32828626
178t 32878805 33020799 33093322 33084925 32795968 32752628 32740078
179t 32646503 32845084 32745854 32824332 32632114 32629080 32521357
180t 32674825 32873071 32862944 32759745 32534136 32541751 32516383
181t 32850319 32950066 32866053 32968706 32763688 32756827 32738199
182t 33218253 33308463 33219559 33278453 33025261 33076222 32908127
183t 32697662 33006884 33036142 32953937 32596810 32710933 32754289
184t 32862478 33023048 32914798 32938656 32685321 32785372 32781088
185t 32595761 32801929 32860092 32779297 32607240 32667733 32570536
186t 32966334 33039338 33047835 33065588 32823377 32881674 32805092
187t 33098549 33369202 33281551 33318871 33097823 33164848 33107920
188t 33280082 33275116 33166120 33176402 32988730 33071291 32864527
189t 32668250 32818660 32799327 32808254 32532122 32684430 32559146
190t 32541976 32771214 32678680 32819969 32374080 32519709 32443986
191t 32746826 32895023 32898392 32871802 32724602 32569206 32600537
192t 32948246 33122775 33153405 33252871 32941934 32949618 32916466
193t 33100179 33195435 33217470 33259674 32952051 32962538 32825110
194t 32717978 32934358 32894919 32849614 32680648 32624933 32575092
195t 32870460 33147094 33058335 33120929 32767121 32891678 32929044
196t 33105193 33348233 33344388 33389375 33101543 33222801 33074403
197t 33529421 33750244 33674918 33766472 33490986 33511014 33417700
198t 32751437 32947364 33033499 33047464 32569876 32749116 32691772
199t 32615383 32761385 32897447 32856877 32666471 32523476 32505884
```

```
200t 33199053 33265199 33187423 33202350 33033979 33083991 32939322
           91      106      147      119      136      135      158
1t   32924214 32909015 32882904 32821827 32828921 32857601 32825807
2t   32643971 32719554 32842719 32648073 32528251 32680593 32585924
3t   33030897 33095600 33046975 33009633 33112738 33103012 32961811
4t   32797137 32822275 32838583 32804693 32832932 32806554 32796337
5t   32893736 32782406 32877116 32778155 32824632 32863179 32783936
6t   32679420 32714514 32635050 32644831 32719519 32551692 32702853
7t   32901202 32927288 32991040 32984877 32904976 33022101 32881336
8t   32971069 32851928 33028676 32870363 32950316 32948519 32841456
9t   32928492 32914799 32955356 32863927 32938271 32895526 32829362
10t  32754490 32806029 32862203 32786524 32826282 32921051 32700440
11t  32794081 32872132 32689494 32641707 32778300 32732494 32768835
12t  32811314 32768101 32766950 32789508 32769457 32687085 32602851
13t  33120410 33148927 33247547 33123469 33121931 33033844 33004010
14t  32888551 32855335 32880054 32846583 32907219 32885836 32837339
15t  32955289 33049419 32982147 32859892 32897300 32899533 32832704
16t  32424314 32691248 32678949 32500118 32505872 32609490 32479914
17t  32851486 32855187 32819224 32778126 32708719 32694860 32770742
18t  32717882 32605197 32524007 32521751 32639273 32563701 32587039
19t  32883040 32796984 32986070 32760283 32855854 32821698 32702435
20t  32819488 32798914 32847972 32626343 32850948 32770973 32794000
21t  32818217 33019918 32889617 32727284 32895744 32847560 32807547
22t  32814064 32887242 32939757 32799212 32837724 32763760 32710954
23t  32652033 32672573 32537363 32486111 32540810 32660929 32524910
24t  32820695 32680739 32834052 32780286 32764874 32795808 32733216
25t  32968286 33052463 32979485 32950498 32965038 32901310 33017929
26t  32998031 32915930 32986305 32989363 32895619 32928638 32904952
27t  33124237 33110487 33088529 33011869 33176961 33004143 33073829
28t  32835696 32716251 32774961 32641984 32826894 32670128 32739398
29t  32867621 32817385 32863392 32789435 32837260 32759538 32845772
30t  32781089 32781690 32784193 32711203 32672881 32722046 32861372
31t  32653591 32864274 32862562 32705521 32796043 32747271 32707416
32t  32987990 33002899 32955037 33020525 32879747 32955729 32922223
33t  32665346 32660427 32539935 32518891 32640941 32562508 32461524
34t  32792472 32795626 32788121 32609227 32664142 32662125 32551600
35t  32673889 32718573 32683042 32629415 32680598 32679491 32782373
36t  32820944 32830225 32780749 32918426 32779438 32641469 32764964
37t  32982789 32973022 32932215 32859528 32936209 32928801 32983786
38t  33113790 33190834 33347009 33162093 33342828 33074357 33219203
39t  32493604 32591315 32586550 32513276 32605877 32418616 32468004
40t  32927541 32973027 33107771 32940967 32993696 33003291 32973255
41t  33119958 32983146 33101170 32952703 33153494 33020820 32965260
42t  32838841 32897911 32804846 32714765 32866415 32668852 32749635
43t  32725686 32757441 32778606 32708573 32849655 32686597 32751507
44t  32739221 32682157 32699927 32865924 32780141 32842751 32609311
45t  33178116 33167457 33152273 33061724 33091598 33193665 33116107
46t  32957661 32906569 33028097 32949773 32942687 33057334 32892567
```

```
47t  32978246 32985069 33041902 32803089 32992093 32889478 32993423
48t  32575299 32560392 32740574 32551002 32616114 32588293 32722428
49t  32573677 32512420 32585607 32488016 32473577 32461521 32488567
50t  32913464 32910841 32982102 32897963 32885370 32968540 32920213
51t  32944515 32945969 32929278 32795389 32863373 32806900 32891612
52t  32644835 32813633 32733154 32668523 32667440 32595373 32622397
53t  32763452 32839382 32826646 32704397 32716608 32806427 32770934
54t  32648990 32655267 32736523 32730851 32849751 32712938 32580391
55t  32946225 32782618 32847432 32833561 32837244 32775064 32844567
56t  32854606 32843855 32833511 32845411 32610293 32793399 32928335
57t  32897418 32970985 32877806 32839208 32808191 32827355 32897100
58t  32890243 32938815 33018277 32801315 33006549 32854243 32900493
59t  33033905 33121677 32984175 32968795 33078572 33046764 32858135
60t  32721187 32750046 32906165 32831108 32865914 32776232 32771110
61t  32740053 32746452 32739941 32830146 32813025 32703092 32785338
62t  33180594 33241442 33200459 33099488 33081116 33067570 33229336
63t  32837626 32772868 32694593 32752156 32783985 32711667 32774802
64t  33288083 33253009 33246875 33269478 33256828 33175632 33241427
65t  32689253 32731310 32737599 32663333 32699106 32748977 32504216
66t  33023290 33065380 33116689 32962945 33052656 33130669 32891539
67t  32670536 32838334 32787907 32707907 32669343 32632239 32788914
68t  32774600 32751633 32858059 32783592 32838937 32783082 32775650
69t  33060253 33073074 32874325 32873282 32981078 32925125 32941791
70t  32885311 33191331 33007943 32956223 33008797 32924826 32970057
71t  32762390 32731514 32805192 32649218 32647698 32749853 32743344
72t  32785285 32707196 32816183 32731277 32766229 32734708 32724007
73t  32732566 32631902 32689790 32720891 32605094 32639156 32554252
74t  32855817 32892769 32848102 32786861 32847212 32707943 32705740
75t  33056348 33069339 33084073 33000916 33025489 33039153 33072656
76t  32889866 32854826 32847118 32952492 32805298 32868276 32873170
77t  32848673 32751590 32789592 32755893 32885099 32754055 32926112
78t  32742977 32682924 32729993 32730452 32754995 32766174 32611267
79t  32498174 32582539 32515181 32355962 32422269 32392618 32344841
80t  32497146 32559153 32685867 32492647 32501901 32628984 32444957
81t  32379802 32406097 32344721 32364092 32480011 32299212 32331749
82t  32832394 32762009 32713391 32650662 32694544 32713098 32566965
83t  32857902 32920251 32823823 32812310 32816338 32786118 32912115
84t  32547266 32408558 32494835 32399702 32375351 32456290 32377925
85t  32370085 32401475 32437837 32461951 32541634 32443627 32357098
86t  32844212 32787802 32778746 32749455 32807657 32728279 32793416
87t  32890874 32866728 32907040 32821193 32921377 32842446 32731387
88t  33014525 33101354 33096528 32979471 33014907 32955443 33064271
89t  33129782 33182197 33311788 33180605 33236500 33201496 33060760
90t  32722161 32835398 32823473 32702567 32726273 32659532 32854021
91t  32921515 32948795 32973068 32802117 32881381 32738337 32855572
92t  33147051 33257659 33228707 33157602 33222288 33189295 33252161
93t  32923233 32841757 32986202 32814337 32887775 32840639 32789736
94t  32874499 32869950 32928296 32813747 32858142 32906912 32917178
```

```
95t  32933889 32805435 32894463 32817851 32798014 32824217 32913238
96t  33104459 32964375 33142305 32909476 33043963 33040049 33081087
97t  32910281 32963265 33002065 32885040 32826498 32960011 32867261
98t  32579811 32588969 32442274 32482495 32411760 32572358 32463292
99t  32673811 32691842 32570153 32606980 32697706 32630363 32776186
100t 32713922 32857629 32890567 32646784 32762044 32625584 32779357
101t 32698242 32781886 32853730 32749899 32753098 32758688 32690565
102t 32636210 32563336 32591805 32486342 32600523 32569473 32509184
103t 32605839 32495383 32699498 32561099 32555260 32514928 32458052
104t 33261030 33166514 33230529 33139009 33150063 33170669 33003634
105t 32500850 32545633 32527679 32444688 32438309 32390304 32458050
106t 32797472 32749477 32731495 32722169 32684041 32804392 32707684
107t 32859804 32927565 32862039 32811555 32845517 32804147 32710078
108t 32771617 32756999 32823554 32754595 32749009 32653662 32658797
109t 32940483 32713251 32896544 32784359 32796074 32787274 32745218
110t 32703028 32764323 32738865 32620397 32752701 32717149 32767088
111t 32820461 32945030 32890402 32788783 32827005 32832694 32777721
112t 32772259 32745727 32749279 32654309 32636224 32679487 32683339
113t 33053841 33104041 33056376 33049890 33193311 33166784 33072937
114t 33148632 33234949 33248190 33108709 33196009 33144609 33212734
115t 32885916 32847103 32826763 32881108 32899184 32793898 32933746
116t 33089982 33079130 33141002 33075142 33121848 33091303 33135188
117t 32605408 32508564 32417342 32492597 32482079 32483627 32615745
118t 32982987 32892607 33120904 32893842 32956905 32952153 32991219
119t 33032900 32968819 32844870 32907601 32898749 32991168 32880824
120t 32824338 32792839 32777467 32743439 32783959 32750023 32662170
121t 32573273 32377076 32559667 32494143 32663932 32474572 32504134
122t 32815325 32971985 32882804 32920512 32843736 32874982 32874567
123t 32999553 33049625 33140917 33016040 33050724 32968182 32929041
124t 33034566 32934273 32863748 32919037 32906620 32890358 32991501
125t 32932122 32931857 33013450 32759325 32828479 32810081 32785180
126t 32938385 32883087 32944293 32886492 32949745 32832698 32843739
127t 32961217 32998096 33010303 32976580 33032443 32991406 32834256
128t 33004843 32830446 33045626 32944814 32953404 32897597 32868490
129t 33148058 33122698 33066279 32951850 33078560 33052980 33078454
130t 32932008 32836081 33030266 32921303 33014223 33039890 32845210
131t 32963483 33002983 33010954 32876549 33049174 32904163 32890095
132t 32744183 32670555 32640695 32628118 32784048 32655776 32607690
133t 33296155 33142586 33161601 33039689 33031463 33044520 33044191
134t 32420209 32537556 32543015 32457445 32499550 32453215 32519413
135t 32631316 32489333 32539771 32544069 32468082 32505747 32468231
136t 32836253 32870272 32880373 32720480 32821001 32856249 32965380
137t 32922778 32957722 33012933 32917183 33009381 32990920 32988459
138t 32608160 32566800 32679628 32522185 32571630 32651337 32562285
139t 32694433 32623655 32813595 32646382 32612519 32683485 32730088
140t 32567446 32557485 32588534 32423232 32567878 32554910 32493548
141t 32956271 32804037 32851673 32837839 32862113 32860024 32917381
142t 32704744 32878141 32740736 32666213 32763245 32713852 32725716
```

```
143t 32542457 32413105 32490804 32394323 32496198 32430935 32416426
144t 32889655 32881477 32850531 32869184 32902636 32832863 32967873
145t 33004823 33011985 33065403 32936308 33103789 33029276 32915924
146t 32378311 32385609 32340813 32364811 32420587 32455607 32320227
147t 33127091 33038229 33157206 33104439 33048991 33063126 33096052
148t 33027072 33085978 33087578 32991717 33062375 32987759 33113948
149t 33018658 32935129 32892007 32838536 32926934 32937205 32916693
150t 32917687 32906528 32817981 32739841 32950272 32910785 32780057
151t 33504439 33421902 33453562 33242292 33318493 33301156 33422501
152t 32921913 32969305 33047057 32832627 33079834 32863737 32987979
153t 32517333 32596480 32506778 32568391 32637878 32510923 32541983
154t 32638220 32656736 32583059 32603100 32701877 32683350 32577306
155t 33070743 33069112 33070616 33059798 33098959 33006914 33038739
156t 32653924 32836751 32765013 32651765 32674204 32690074 32752570
157t 32953585 32862614 32922659 32821351 32767373 32866361 32829074
158t 32735531 32729621 32724953 32561577 32745974 32729592 32643150
159t 32962932 33077085 32941259 32902092 32908147 32895781 32872300
160t 33321093 33530558 33436149 33269697 33271238 33214644 33283747
161t 32457093 32474083 32465939 32413626 32428815 32381949 32316349
162t 32652916 32645734 32672949 32575277 32719605 32602290 32749295
163t 33042791 33238990 33106546 33035078 33138865 33053317 33047270
164t 33315681 33254261 33195241 33190893 33332144 33192447 33208732
165t 32604704 32495515 32576894 32552433 32277493 32365147 32425186
166t 32462907 32457137 32499287 32513419 32546253 32447253 32478730
167t 32668165 32735589 32803700 32740986 32766226 32586054 32647895
168t 32708136 32627538 32746656 32649112 32607662 32575338 32668021
169t 33046134 33018701 33050882 32970314 32988183 33018641 33061378
170t 32781088 32958200 33072626 32692249 32943497 32719154 32810994
171t 33040159 33049930 33037305 32852104 32996959 33029152 32795733
172t 32745259 32639285 32778017 32662774 32625821 32782432 32750675
173t 32738223 32782076 32802303 32753602 32707701 32697572 32745595
174t 33073647 33172239 33001589 33137659 32945408 32996641 33031802
175t 33027166 32805936 32921227 32880729 32836173 32899041 32899886
176t 32975589 32934455 33001515 32958183 32963760 32915866 32827757
177t 32874689 32977760 32849596 32884044 32942630 32902629 32928383
178t 32820648 32703312 32785730 32617057 32719205 32684900 32863584
179t 32505781 32539103 32505840 32498442 32478205 32428373 32450931
180t 32490405 32665397 32590672 32456949 32593415 32583922 32424074
181t 32646796 32701185 32725309 32727456 32757054 32760084 32689496
182t 32973317 33024508 33067941 32997920 32921794 32986721 32981070
183t 32690495 32740161 32651812 32714399 32634990 32674339 32650092
184t 32769449 32691254 32766680 32702737 32775397 32629050 32635692
185t 32514658 32597414 32595217 32603976 32481790 32565845 32528324
186t 32655616 32892452 32833308 32706015 32738881 32777400 32712625
187t 33130737 32983070 32964461 32953968 33086109 33053737 32969755
188t 32978460 32858319 33097832 32999418 32955414 33015782 32915192
189t 32628924 32579456 32590757 32526647 32611510 32427237 32417185
190t 32316302 32384059 32315501 32384301 32486575 32333557 32277231
```

```
191t 32627651 32587379 32639907 32503882 32635559 32506095 32534472
192t 33041858 32829126 32909371 32912866 32833432 32886504 32854363
193t 32948783 32961098 32896445 32822553 32862416 32858768 32883479
194t 32649524 32721832 32551332 32517280 32595698 32484519 32517033
195t 32858988 32862323 32972196 32729199 32823109 32827964 32857745
196t 33053088 33109692 33174450 33017199 33054344 33030277 32992571
197t 33436446 33493198 33384892 33468010 33404456 33353053 33228914
198t 32624442 32558015 32776274 32532653 32610337 32543988 32562350
199t 32682410 32640441 32624976 32488399 32510136 32540766 32390400
200t 32996175 33033523 32989113 32876829 32889121 32935379 32958837
          163      162      165      170
1t   32823567 32992382 32882530 32846969
2t   32506011 32570995 32497569 32588425
3t   33094750 32973878 33139288 33149564
4t   32697020 32860947 32799402 32856848
5t   32786920 32889240 32832076 32727551
6t   32591806 32705428 32648085 32709565
7t   32912696 32936862 32838966 32998861
8t   32836317 32889165 32926502 33004821
9t   32814855 32916962 32969567 32858883
10t  32746960 32748835 32751656 32677123
11t  32670244 32909463 32927859 32660614
12t  32713229 32764342 32731378 32864351
13t  32997761 33051422 33155544 33030999
14t  32806372 32879113 32799427 32771248
15t  32919281 33008357 32920845 32964483
16t  32626722 32542801 32640142 32578028
17t  32814354 32737048 32841801 32714341
18t  32616497 32580742 32705777 32579802
19t  32824596 32825610 32805092 32844194
20t  32713121 32736796 32691512 32748379
21t  32763322 32824006 32839595 32921956
22t  32736086 32804362 32866591 32729084
23t  32511082 32677718 32630429 32543675
24t  32721861 32747507 32673548 32783609
25t  32916575 32987635 32888635 32844112
26t  32895388 32927334 32954405 32892729
27t  32937829 33030864 32993530 33021785
28t  32748376 32688115 32641559 32741664
29t  32737556 32804577 32829438 32826573
30t  32769743 32666669 32767243 32712441
31t  32787500 32826750 32735448 32796719
32t  32771293 32897685 32946818 32954257
33t  32629409 32589760 32673309 32559411
34t  32553967 32644957 32741005 32658954
35t  32611435 32661334 32653414 32575524
36t  32668913 32693989 32824762 32723684
37t  32956939 32948398 32990658 32947290
```

```
38t  33248058 33147282 33316729 33169687
39t  32536482 32569199 32521530 32646972
40t  32992315 33105169 32960935 32979683
41t  32950043 32994327 33078070 33113226
42t  32748632 32783674 32719487 32659830
43t  32716920 32639519 32767986 32711244
44t  32739253 32854991 32862522 32680752
45t  33035037 33190223 33126125 33011192
46t  33065212 33000238 33054963 32927513
47t  32862895 32958519 32979340 32924736
48t  32553160 32504024 32567508 32511630
49t  32358938 32361801 32461703 32466543
50t  32824817 32951590 33073157 32848187
51t  32843140 32751226 32888929 32910742
52t  32651986 32672661 32678351 32682825
53t  32713850 32842819 32746325 32810802
54t  32689965 32833609 32898396 32700715
55t  32816226 32859733 32791550 32894612
56t  32760173 32750910 32732005 32737245
57t  32884693 32766408 32904626 32968109
58t  32904213 32892813 33085020 33014201
59t  32955572 33048980 33038604 32927321
60t  32815003 32897019 32796239 32856887
61t  32673432 32772628 32883123 32834771
62t  32985616 33194754 33055105 33144762
63t  32809394 32717873 32785659 32744469
64t  33284362 33214951 33272777 33181590
65t  32583503 32653519 32576844 32569702
66t  32894631 33054949 33054330 33054346
67t  32638601 32668233 32701394 32689645
68t  32740284 32915974 32784130 32788055
69t  32833715 32913301 32853528 32947537
70t  32945821 32941705 32931653 32956925
71t  32725272 32700080 32773241 32607194
72t  32650900 32791074 32770594 32691415
73t  32559589 32635860 32652608 32565619
74t  32826958 32852349 32752175 32773028
75t  33069336 33039286 33102312 33024710
76t  32835400 32825849 32835730 32774505
77t  32786378 32827013 32881486 32797504
78t  32549741 32801379 32694142 32699232
79t  32269087 32493138 32479711 32392035
80t  32466453 32673470 32636844 32554597
81t  32307077 32381029 32447008 32375237
82t  32605011 32735045 32746630 32836964
83t  32813337 32763509 32828185 32767801
84t  32294049 32470430 32466241 32421935
85t  32421819 32454380 32436214 32346531
```

```
86t  32825957 32901318 32853696 32720655
87t  32863792 32815607 32747097 32818059
88t  32956348 33050002 33032114 33070342
89t  33144092 33124159 33217436 33126280
90t  32721319 32719046 32832542 32788879
91t  32879404 32734708 32883451 33028743
92t  33214994 33223505 33143731 33248342
93t  32791960 32925625 32808899 32917480
94t  32875735 32797847 32984441 32828649
95t  32880222 32875564 32853823 32878715
96t  33016415 32993791 32971470 33028943
97t  32908948 32882066 32988060 32851163
98t  32397538 32478100 32473423 32494319
99t  32530293 32632333 32737905 32607958
100t 32754233 32766085 32800855 32728066
101t 32577940 32681880 32673360 32772242
102t 32591110 32471218 32586052 32555318
103t 32601796 32646806 32604377 32580964
104t 33184435 33147234 33183708 33283994
105t 32389272 32458951 32469800 32530660
106t 32738682 32700166 32740320 32761791
107t 32814239 32864951 32771222 32848046
108t 32643883 32688627 32747746 32766744
109t 32732550 32833585 32901994 32784563
110t 32707263 32733008 32732493 32708609
111t 32786543 32753718 32794712 32826583
112t 32716653 32787894 32823050 32585056
113t 33053301 33056906 33111122 32993885
114t 33196844 33115631 33243163 33174471
115t 32804858 32871019 32876787 32828224
116t 33138457 33230300 33127511 33095009
117t 32491161 32624104 32536452 32506470
118t 32896671 32936503 32897470 32912304
119t 32851812 33006182 32957172 33052447
120t 32800966 32784991 32844515 32735524
121t 32464498 32499209 32477273 32572536
122t 32838733 32769627 32911895 32818783
123t 32998880 33044568 33050046 33004198
124t 32828775 32824652 32898866 32987968
125t 32919436 32805721 33006666 32851873
126t 32815391 32961295 32981849 32838081
127t 32912750 33067392 32905393 32956155
128t 32954443 32893677 33075327 32925330
129t 33054560 33012013 32999888 33046008
130t 32902154 32952478 32962895 32829748
131t 32936753 32956750 33001212 33045546
132t 32664692 32629265 32716516 32743565
133t 33076157 33130259 33129716 33138108
```

```
134t 32483226 32487159 32497109 32516426
135t 32476817 32516629 32590928 32549558
136t 32830220 32810645 32776088 32685287
137t 32900963 32872338 33038926 33023329
138t 32635129 32566770 32541950 32578474
139t 32658835 32784524 32803723 32625358
140t 32521827 32547298 32568520 32467780
141t 32684883 32863857 32753944 32930078
142t 32685914 32689403 32647612 32770951
143t 32482545 32494298 32526033 32530907
144t 32848018 33031761 32913325 32864008
145t 32949134 33041582 33066211 33036242
146t 32367766 32350577 32334322 32294149
147t 33106881 33150527 33078750 33095214
148t 32994196 33010385 33099142 33064344
149t 32893386 32769993 32921433 32876270
150t 32791440 32825104 32848160 32773755
151t 33313004 33303181 33343747 33415263
152t 32894100 32976021 32898754 32896219
153t 32483061 32587766 32517309 32503969
154t 32617376 32636164 32711398 32529161
155t 33053383 33136502 33109650 33114389
156t 32717037 32582006 32765023 32619531
157t 32809569 32993867 32897942 32826500
158t 32638614 32687244 32667945 32612230
159t 32926289 32960920 33012455 32975680
160t 33293180 33374465 33300147 33332840
161t 32432976 32515426 32491102 32402608
162t 32606197 32595879 32630811 32580147
163t 33231431 33233841 33070079 33044045
164t 33235697 33254819 33234844 33290563
165t 32354383 32436162 32402542 32263648
166t 32466768 32518324 32629074 32503125
167t 32646280 32705428 32671112 32807068
168t 32640127 32725978 32666827 32563414
169t 33048949 33011136 32930200 33009141
170t 32735528 32846948 32736534 32832408
171t 32970956 33038251 32971065 32978839
172t 32723239 32666577 32755603 32764196
173t 32625937 32807554 32817308 32824152
174t 32949029 33116713 33021505 33081460
175t 32929460 32813449 32852508 32863099
176t 32827915 33017291 33063027 32903071
177t 32747505 32806537 32872467 32796907
178t 32665306 32738230 32711670 32599282
179t 32443789 32522926 32443388 32524216
180t 32526721 32430979 32540734 32534197
181t 32621323 32769537 32719730 32629381
```

```
182t 32937722 32941509 32948932 32958315
183t 32544302 32707069 32698156 32575462
184t 32654858 32680668 32662385 32683234
185t 32651996 32539142 32564446 32585327
186t 32690809 32602380 32753344 32749287
187t 32960693 33074978 33098062 32992022
188t 32935945 32855854 32904026 33004423
189t 32628482 32556307 32493073 32505253
190t 32381894 32380896 32377659 32432010
191t 32546415 32743536 32661613 32528793
192t 32878447 32911611 32827318 32880504
193t 32915702 32826546 32859399 32840517
194t 32508866 32660784 32641787 32605386
195t 32741920 32773797 32829703 32848321
196t 33096728 33094829 32966562 33008693
197t 33447525 33385435 33466437 33365536
198t 32623459 32577273 32640498 32597290
199t 32494719 32452478 32508466 32544772
200t 32825910 32956914 32923798 32913778
```

– `seeds`: The seeds used for the experiments, each seed corresponds to each instance in the rows of the test `experiments` matrix.

```
> # Get the seeds used for testing
> iraceResults$testing$seeds

        1t         2t         3t         4t         5t         6t 
2102888981 1373895548  737168275 1746854571  138389571  786192030 
        7t         8t         9t        10t        11t        12t 
1178378408 1091144016  487896768  947627276 1244452644   16831254 
       13t        14t        15t        16t        17t        18t 
1096983435  945727373  496130080  461233007  608196975 1606584204 
       19t        20t        21t        22t        23t        24t 
 521504677 1431690252  529426667 1789253406 1075199038  212980354 
       25t        26t        27t        28t        29t        30t 
1053133819 1241062382  699613509  792559700 1968946615  153013166 
       31t        32t        33t        34t        35t        36t 
 275111045 2064313849 1834264287  900370477 2075000813  949056843 
       37t        38t        39t        40t        41t        42t 
1293439853 1382645237 1992954192 1221484178  854308369 1131908672 
       43t        44t        45t        46t        47t        48t 
1725861088  792373773 1599641910  276520080 1396259076 1341093643 
       49t        50t        51t        52t        53t        54t 
1365056851  992822221  628348915  315587582 2039354376  382197123 
       55t        56t        57t        58t        59t        60t 
1290426930  992839633 1422553503  555312338  972317384 1176005400 
       61t        62t        63t        64t        65t        66t 
 987978749  932768075 2129850937  870245330 2141031147 1512246576 
       67t        68t        69t        70t        71t        72t 
1206098133 1495543539 1749774608  756951873 1990335629 1311241583 
```

```
       73t        74t        75t        76t        77t        78t 
 869345563  674592297  692563853 1529228658 2135517163  495789009 
       79t        80t        81t        82t        83t        84t 
 733248345 1097131571  783813028 1224369123 1515029220   55726933 
       85t        86t        87t        88t        89t        90t 
1861855068  811669657 1755606982 2013216975 1987920792 1838008341 
       91t        92t        93t        94t        95t        96t 
1523003566  536585773  488967397 1900892596 1685647815 1879534497 
       97t        98t        99t       100t       101t       102t 
 615239714 1485687956  301667990 1577996667  116014708 2050723150 
      103t       104t       105t       106t       107t       108t 
   4917828   85907529   14505414  404338521 1782177479 1110666514 
      109t       110t       111t       112t       113t       114t 
1459351962 1990218731   25878560  450214056 1373087355 1924677682 
      115t       116t       117t       118t       119t       120t 
1753820896 1490257498   75946130 1754309743  190939470 1660044325 
      121t       122t       123t       124t       125t       126t 
1149569885 1742140820 1314955828 1365508241  780241093  246869782 
      127t       128t       129t       130t       131t       132t 
  46474598  476633877  598751590  319979099 1580796383  248208077 
      133t       134t       135t       136t       137t       138t 
1932600244 1041134102 1434253441 1475284765 1818558707   38684667 
      139t       140t       141t       142t       143t       144t 
1280701788  151090926  885745127  102783810 1697563041  598510230 
      145t       146t       147t       148t       149t       150t 
 157544092 1915278990 1565428700 1822827654 2109298646  628215893 
      151t       152t       153t       154t       155t       156t 
1189017503 1122233252 1793546828  297070912 1833341799  877101869 
      157t       158t       159t       160t       161t       162t 
 227494448  851827281  997089374  907501177 1689878855  340557566 
      163t       164t       165t       166t       167t       168t 
1875446491  831244715  917718233 1976068837 1567828607   89334968 
      169t       170t       171t       172t       173t       174t 
 544836869 1289090481    5620582  934856202  491565145  167509626 
      175t       176t       177t       178t       179t       180t 
 779494849 1604133301  275015508 1229007180 2137191717  582176268 
      181t       182t       183t       184t       185t       186t 
1597734639  991015576  112871509  409269595 1912347056  280108552 
      187t       188t       189t       190t       191t       192t 
 195538207 1297690789  628268213 1327172460  432898400 1239162891 
      193t       194t       195t       196t       197t       198t 
1065354274 1597708233 1918246550 2136562957 1356069411 1394989400 
      199t       200t 
 192068973 1171789662 
```

In the example, instance `1000-1.tsp` is executed with seed 2102888981.

9.3 Analysis of results

The final configurations returned by **irace** are the elites of the final race. They are reported in decreasing order of performance, that is, the best configuration is reported first.

If testing is performed, you can further analyze the resulting best configurations by performing statistical tests in R:

```
> results <- iraceResults$testing$experiments
> # Wilcoxon paired test
> conf <- gl(ncol(results), # number of configurations
+            nrow(results), # number of instances
+            labels = colnames(results))
> pairwise.wilcox.test (as.vector(results), conf, paired = TRUE, p.adj = "bonf")

Pairwise comparisons using Wilcoxon signed rank test with continuity correction

data:  as.vector(results) and conf

    3       16      25      54      68      87      117     91
16  < 2e-16 -       -       -       -       -       -       -
25  < 2e-16 1.00000 -       -       -       -       -       -
54  < 2e-16 1.00000 0.02513 -       -       -       -       -
68  < 2e-16 < 2e-16 < 2e-16 < 2e-16 -       -       -       -
87  2.0e-13 < 2e-16 < 2e-16 < 2e-16 0.00015 -       -       -
117 < 2e-16 < 2e-16 < 2e-16 < 2e-16 4.5e-12 < 2e-16 -       -
91  < 2e-16 < 2e-16 < 2e-16 < 2e-16 0.00046 9.5e-14 0.01554 -
106 < 2e-16 < 2e-16 < 2e-16 < 2e-16 0.00127 4.8e-12 0.06613 1.00000
147 < 2e-16 < 2e-16 < 2e-16 < 2e-16 1.00000 6.0e-08 1.8e-07 1.00000
119 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 8.8e-08 1.9e-15
136 < 2e-16 < 2e-16 < 2e-16 < 2e-16 4.1e-11 < 2e-16 1.00000 1.00000
135 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 0.07762 8.6e-10
158 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 0.00766 3.4e-10
163 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 2.9e-07 < 2e-16
162 < 2e-16 < 2e-16 < 2e-16 < 2e-16 2.5e-11 < 2e-16 1.00000 0.00983
165 < 2e-16 < 2e-16 < 2e-16 < 2e-16 1.0e-09 < 2e-16 1.00000 1.00000
170 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 < 2e-16 0.47329 1.9e-07
    106     147     119     136     135     158     163     162
16  -       -       -       -       -       -       -       -
25  -       -       -       -       -       -       -       -
54  -       -       -       -       -       -       -       -
68  -       -       -       -       -       -       -       -
87  -       -       -       -       -       -       -       -
117 -       -       -       -       -       -       -       -
91  -       -       -       -       -       -       -       -
106 -       -       -       -       -       -       -       -
147 1.00000 -       -       -       -       -       -       -
119 5.3e-14 < 2e-16 -       -       -       -       -       -
136 1.00000 9.4e-05 9.0e-10 -       -       -       -       -
135 2.5e-08 2.2e-16 0.77675 0.00118 -       -       -       -
158 3.1e-10 1.8e-15 1.00000 0.00016 1.00000 -       -       -
```

```
163 7.4e-16 < 2e-16 1.00000 2.0e-11 0.22694 1.00000 -       -
162 0.02323 2.8e-07 3.1e-06 1.00000 0.22983 0.02838 1.4e-06 -
165 1.00000 0.00011 1.1e-09 1.00000 0.00030 1.0e-04 3.9e-13 1.00000
170 3.6e-07 6.5e-14 0.03559 0.04126 1.00000 1.00000 0.01823 1.00000
    165
16  -
25  -
54  -
68  -
87  -
117 -
91  -
106 -
147 -
119 -
136 -
135 -
158 -
163 -
162 -
165 -
170 0.05975

P value adjustment method: bonferroni
```

The Kendall concordance coefficient (`W`) and the Spearman's rho can be applied over data that has the characteristics of the data obtained in the testing, that is a full matrix where all configurations are executed in all instances. `W` can show if the configurations tested have an homogeneous performance on the used instances set. If evidence of an heterogeneous scenario found we recommend to make some adjustments in the **irace** options as described in Section 10.5.

```
> irace:::concordance(iraceResults$testing$experiments)

$kendall.w
[1] 0.6089698

$spearman.rho
[1] 0.6070048
```

It is also possible, as shown in Fig. 7, to plot the performance on the test set of the best-so-far configuration over the number of experiments as follows:

```
# Get summary data from the logfile.
irs <- irace_summarise(iraceResults)
# Get number of iterations
iters <- irs$n_iterations
# Get number of experiments (runs of target-runner) up to each iteration
fes <- cumsum(table(iraceResults$state$experiment_log[["iteration"]]))
# Get the mean value of all experiments executed up to each iteration
```

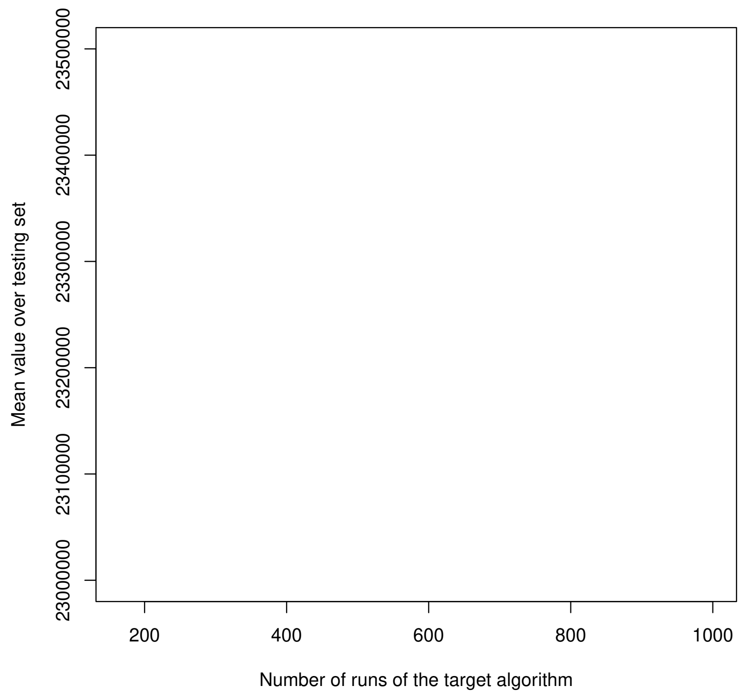

Figure 7: Testing set performance of the best-so-far configuration over number of experiments. Label of each point is the configuration ID.

```r
# for the best configuration of that iteration.
elites <- as.character(iraceResults$iterationElites)
values <- colMeans(iraceResults$testing$experiments[, elites])
stderr <- function(x) sqrt(var(x)/length(x))
err <- apply(iraceResults$testing$experiments[, elites], 2L, stderr)
plot(fes, values, type = "s",
     xlab = "Number of runs of the target algorithm",
     ylab = "Mean value over testing set", ylim=c(23000000,23500000))
points(fes, values, pch=19)
arrows(fes, values - err, fes, values + err, length=0.05, angle=90, code=3)
text(fes, values, elites, pos = 1)
```

The **irace** package also provides an implementation of the ablation method [6]. See Section 10.9.

Finally, more advanced visualizations of the behavior of **irace** are provided by the ACVIZ software package [4], which is available at `https://github.com/souzamarcelo/acviz`. See an example in Fig. 8.

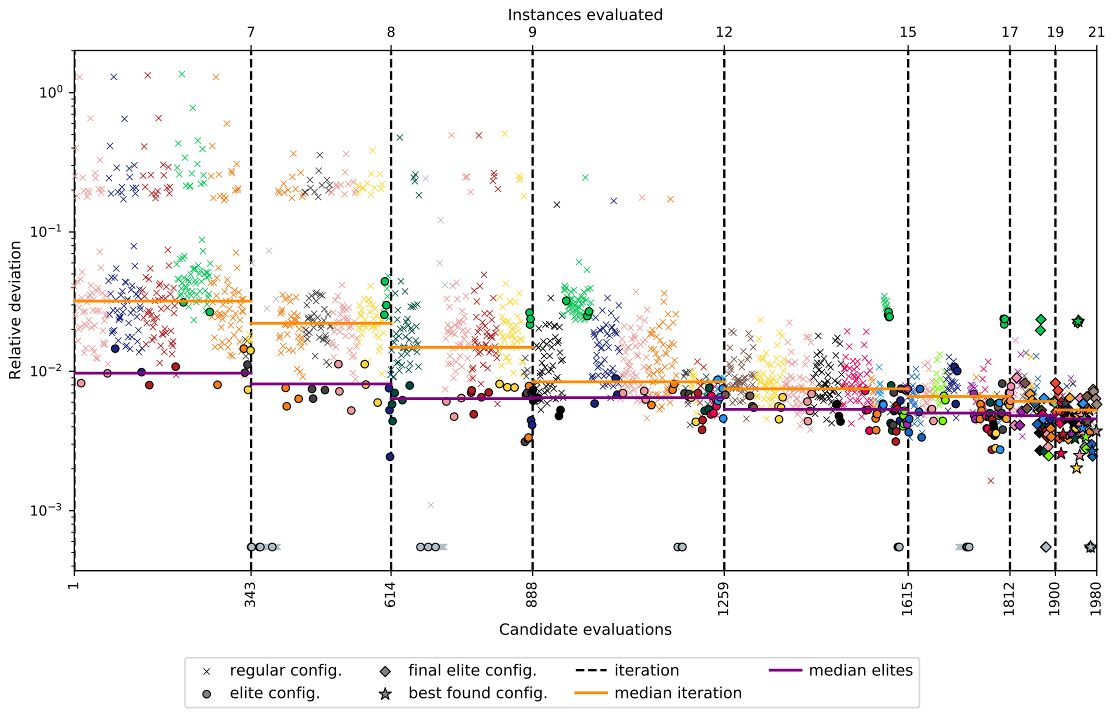

Figure 8: Visualization produced by ACVIZ [4].

10 Advanced topics

10.1 Tuning budget

Before setting the budget for a run of **irace**, please consider the number of parameters that need to be tuned, available processing power and available time. The optimal budget depends on the difficulty of the tuning scenario, the size of the parameter space and the heterogeneity of the instances. Typical values range from 1 000 to 100 000 runs of the target algorithm, although smaller and larger values are also possible. Currently, **irace** does not detect whether the given budget allows generating all possible configurations. In such a case, the use of *iterated* racing is unnecessary: One can simply perform a single race of all configurations (see FAQ in Section 12.11).

Irace provides two options for setting the total tuning budget (`maxExperiments` and `maxTime`). The option `maxExperiments` limits the number of executions of `targetRunner` performed by **irace**. The option `maxTime` limits the total time of the `targetRunner` executions. When this latter option is used, `targetRunner` must return the evaluation cost together with the execution time (`"cost time"`).

When the goal is to minimize the computation time of an algorithm, and you wish to use `maxTime` as the tuning budget, `targetRunner` must return the time also as the evaluation cost, that is, return the time twice as `"time time"`.

When using `targetEvaluator` and using `maxTime` as tuning budget, `targetRunner` just returns the time (`"time"`) and `targetEvaluator` returns the cost.

When using maxTime, **irace** estimates the execution time of each targetRunner execution before the configuration. The amount of budget used for the estimation is set with the option budgetEstimation (default is 2%). The obtained estimation is adjusted after each iteration using the obtained results and it is used to estimate the number of experiments that can be executed. Internally, **irace** uses the number of remaining experiments to adjust the number of configurations tested in each race.

10.2 Multi-objective tuning

Currently, **irace** only optimizes one cost value at a time, which can be solution cost, computation time or any other objective that is returned to **irace** by the targetRunner. If the target algorithm is multi-objective, it will typically return not a single cost value, but a set of objective vectors (typically, a Pareto front). For tuning such a target algorithm with **irace**, there are two alternatives. If the algorithm returns a single vector of objective values, they can be aggregated into one single number by using, for example, a weighted sum. Otherwise, if the target algorithm returns a set of objective vectors, a unary quality metric (e.g., the hypervolume) may be used to evaluate the quality of the set.[3]

The use of aggregation or quality metrics often requires normalizing the different objectives. If normalization bounds are known a priori for each instance, normalized values can be computed by targetRunner. Otherwise, the bounds may be dynamically computed while running **irace**, by using targetEvaluator. In this case, targetRunner will save the output of the algorithm, then the first call to targetEvaluator will examine the output produced by all calls to targetRunner for the same instance, update the normalization bounds and return the normalized output. Subsequent calls to targetEvaluator for the same instance will simply return the normalized output. A similar approach can be used to dynamically compute the reference points or sets often required by unary quality metrics.

For more information about defining a targetEvaluator, see Section 5.3. Examples of tuning a multi-objective target algorithm using the hypervolume can be found in the examples at $IRACE_HOME/examples/hypervolume and $IRACE_HOME/examples/moaco.

10.3 Tuning for minimizing computation time

When using **irace** for tuning algorithms that only report computation time to reach a target, targetRunner should return the execution time of a configuration instead of solution cost. When using maxTime as the budget, this means that targetRunner must return twice the execution time since the first value is the minimization objective and the second value is used to track the budget consumed.

Starting from version 3.0, **irace** includes an elitist racing procedure that implements an **adaptive capping mechanism** [15]. Adaptive capping [8] is a configuration technique that avoids the execution of long runs of the target algorithm, focusing the configuration budget in the evaluation of the best configurations found. This is done by bounding the execution time of each configuration based on the best performing candidate configurations.

To use adaptive capping, the capping option must be enabled and the elitist irace option must be selected. When evaluating candidate configurations on an instance, **irace** calculates an execution bound based on the execution times of the elite configurations. The boundType option defines how the performance of the elite configurations is defined to obtain the execution bound. The default value of boundType calculates the performance (p_i^s) of each elite configuration (s) as the mean execution time of the instances already executed in the race and the currently executed

[3]An implementation is publicly available at http://lopez-ibanez.eu/hypervolume [7]

instance (i). The cappingType option specifies the measure used to obtain the elite configurations bound. By default, the execution bound is calculated as the median of the execution times of the elite configurations:

$$b_i = \text{Median}_{\theta_s \in \Theta^{\text{elite}}} \{p_i^s\} \tag{1}$$

The execution bound for new configurations (j) is calculated by multiplying the elite configurations bound by the number of instances (i) in the execution list and subtracting the mean execution time of the instances executed by the candidate:

$$k_i^{'j} = b_i \cdot i + b^{\min} - p_{i-1}^j \cdot (i-1) \tag{2}$$

A small constant $b^{\min}$ is added to account for time measurements errors. These settings are also used to apply a dominance elimination criterion together with the statistical test elimination. The domination criterion is defined as:

$$b_i + b^{\min} < p_i^j \tag{3}$$

When elite configurations dominate new configurations, these are eliminated from the race.

The default statistical test when capping is enabled is `t-test`. This test is more appropriate to configure algorithms for optimizing runtime (see Section 10.6).

The execution bound is constantly adjusted by **irace** based on the best configurations times, nevertheless, a maximum execution time ($b^{\max}$) is never exceeded. This maximum execution time must be defined in the configuration scenario when capping is enabled. To specify the maximum execution bound for the target runner executions use the boundMax option. The final execution bound (k_i^j) is calculated by:

$$k_i^j = \begin{cases} b^{\max} & \text{if } k_i^{'j} > b^{\max}, \\ \min\{b_i, b^{\max}\} & \text{if } k_i^{'j} \leq 0, \\ k_i^{'j} & \text{otherwise;} \end{cases} \tag{4}$$

Additionally, the boundDigits option defines the precision of the time bound provided by **irace**, the default setting is 0.

Timed out executions occur when the maximum execution bound (boundMax) is reached and the algorithm has not achieved successful termination or a defined quality goal. In this case, it is a common practice to apply a penalty known as PARX, in which timeouts are penalized by multiplying boundMax by a constant X. The constant X may be set using the boundPar option. Bounded executions are executions that do not achieve successful termination or a defined quality goal in the execution bound (k_i^j) set by **irace**, which is smaller than boundMax. The boundAsTimeout option replaces the evaluation of bounded executions by the boundMax value. More details about the implementation of adaptive capping can be found in Pérez Cáceres et al. [15].

Note that bounded executions are not timed out executions and thus, they will not be penalized by PARX.

Penalized evaluations of timed out and bounded executions are only used for the elimination tests and the comparison between the quality of configurations. To calculate execution bounds and computation budget consumed, **irace** uses only unpenalized execution times. The unpenalized execution time must be provided by the target runner or target evaluator as described in Section 5.2 and Section 5.3 .

More advanced capping methods that are applicable to minimizing solution cost are available when combining irace with the `capopt` package described by De Souza et al. [5].

10.4 Hyper-parameter optimization of machine learning methods

The **irace** package can also be used for model selection and hyper-parameter optimization of machine learning (ML) methods. We will next explain a possible setup for one given dataset and using 10-fold cross-validation (CV). Generalizing to multiple datasets and different resampling strategies, e.g. leave-one-out, is straightforward.

First, split the dataset into training, to be used by **irace**, and testing, to be used for evaluating the performance of the configuration returned by **irace**. A typical split could be 70% and 30%, respectively.

The training set is used by **irace** to perform 10-fold CV, that is, the data is split into 10 folds. A single run of the `targetRunner` will use 9 folds for training and the remaining fold for validation. Splitting the data into folds can be done at each call of `targetRunner` or before running **irace**, however, it is important that the split is always the same for every call of the `targetRunner`, i.e., the content of the folds does not change, only which folds are used for training and validation will change.

The setup of **irace** should be as follows:

- `trainInstancesFile="train-instances.txt"`, where this file contains one number per line from 1 to 10. This number will tell the `targetRunner` which fold should be used for validation.
- `trainInstancesDir=""`, because the folds are the "instances" and you do not have actual instance files. If you want to pass the name of the dataset to the `targetRunner`, you can specify it either at each line of `"train-instances.txt"`, directly in the `targetRunner`, or as a fixed parameter in the `parameterFile`.
- `deterministic=1` unless it really makes sense to train more than once the same ML model on the same data. If it makes sense, then your `targetRunner` should use the seed passed by **irace** to seed the ML model before training.
- `sampleInstances=0` because the folds should already be generated by randomly sampling the dataset.
- `testType="t-test` because the performance metrics in ML are typically the mean of the CV results, which assumes that the performance are close to normally distributed.
- `firstTest=2` because **irace** should discard configurations very aggressively looking for maximum generality.

Finally, your `targetRunner` needs to be able to do the following:

- Receive from **irace** the hyper-parameter settings, the dataset name and a fold number (the "instance"). Let us use fold 3 as an example.
- Train the ML model on the whole training set minus fold 3, then validate (score) the model on fold 3 and return the score to **irace** (negated if the score must be maximized, because **irace** assumes minimization). Since each fold is different, each instance should give a different result. Each row in the table printed by **irace** should print something different; otherwise, something is wrong in your setup.

The above is actually 10 times faster than doing 10-fold CV for each call to `targetRunner`, thus, you should assign to **irace** 10 times the budget than what would be assigned to other methods that do a complete 10-fold CV at each step.

10.5 Heterogeneous scenarios

We classify a scenario as homogeneous when the target algorithm has a consistent performance regarding the instances; roughly speaking, good configurations tend to perform well and bad configurations tend to perform poorly on all instances of the problem. By contrast, in heterogeneous scenarios, the target algorithm has an inconsistent performance on different instances, that is, some configurations perform well for a subset of the instances, while they perform poorly for a different subset.

When facing a heterogeneous scenario, the first question should be whether the objective of tuning is to find configurations that perform reasonably well over all instances, even if that configuration is not the best ones in any particular instance (a generalist). If this is not the goal, then it would be better to partition instances into more similar subsets and execute **irace** separately on each subset. This will lead to a portfolio of algorithm configurations, one for each subset, and algorithm selection techniques can be used to select the best configuration from the portfolio when facing a new instance.

To make sure **irace** is not misled by results on few instances, it may be useful to increase the number of instances executed before doing a statistical test using the option `firstTest`, e.g., `--first-test 10` (default value is 5), in order to see more instances before discarding configurations. The option `elitistNewInstances` in elitist **irace** (option `elitist`) can be used to increase the number of new instances executed in each iteration, e.g., `--elitist-new-instances 5` (default value is 1).

If finding an overall good configuration for all the instances is the objective, then we recommend that instances are randomly sampled (option `sampleInstances`), unless one can provide the instances in a particular order that does not bias the tuning towards any subset.

If instances are easily categorized in different classes, then we recommend to create "blocks" of instances in `trainInstancesFile`, where each block should contain one instance from each class. Then set the option `blockSize` to the number of classes within each block, so that **irace** will always see a complete block of instances before eliminating configurations. The value of `blockSize` will multiply the effective values of `firstTest` and `eachTest`. Randomly sampling instances (`sampleInstances=1`) will randomly sample the blocks but not break the blocks.

While executing **irace**, the homogeneity of the scenario can be observed by examining the values of Spearman's rank correlation coefficient and Kendall's concordance coefficient in the text output of **irace**. See Section 9.1 for more information.

10.6 Choosing the statistical test

The statistical test used in **irace** identifies statistically bad performing configurations that can be discarded from the race in order to save budget. Different statistical tests use different criteria to compare the cost of the configurations, which has an effect on the tuning results.

Irace provides two types of statistical tests (option `testType`). Each test has different characteristics that are beneficial for different goals:

- Friedman test (`F-test`): This test uses the ranking of the configurations to analyze the differences between their performance. This makes the test suitable for scenarios where the scale of the performance metric is not as important to assess configurations as their relative ranking. This test is also indicated when the distribution of the mean performances deviates greatly from a normal distribution. For example, the ranges of the performance metric on different instances may be completely difference and comparing the performance of different configurations using the mean over multiple instances may be deceiving. We recommend to use the `F-test` (default when `capping` is not enabled) when tuning for solution cost and whenever the best performing algorithm should be among the best in as many instances as possible.

- Student's t-test (`t-test`): This test uses the mean performance of the configurations to analyze the differences between the configurations.[4] This makes the test suitable for scenarios where the differences between values obtained for different instances are relevant to assess good configurations. We recommend using t-test, in particular, when the target algorithm is minimizing computation time and, in general, whenever the best configurations should obtain the best average solution cost.

The confidence level of the tests may be adjusted by using the option `confidence`. Increasing the value of `confidence` leads to a more strict statistical test. Keep in mind that a stricter test will require more budget to identify which configurations perform worse. A less strict test discards configurations faster by requiring less evidence against them and, therefore, it is more likely to discard good configurations.

10.7 Complex parameter space constraints

Some parameters may have complex dependencies. Ideally, parameters should be defined in the way that is more likely to help the search performed by **irace**. For example, when tuning a branch and bound algorithm, one may have the following parameters:

- branching (`b`) that takes values in `{0,1,2,3}`, where 0 indicates no branching will be used and the rest are different types of branching.
- stabilization (`s`) that takes values in `{0,1,2,3,4,5,6,7,8,9,10}`, of which for `b=0` only `{0,1,2,3,4,5}` are relevant.

In this case, it is not possible to describe the parameter space by defining only two parameters for **irace**. An extra parameter must be introduced as follows:

```
# name    label   type   range                    condition
b         "-b "   c      (0,1,2,3)
s1        "-s "   c      (0,1,2,3,4,5)            | b == "0"
s2        "-s "   c      (0,1,2,3,4,5,6,7,8,9,10) | b != "0"
```

Parameters whose values depend on the value of other parameters may also require using extra parameters or changing the parameters and processing them in `targetRunner`. For example, given the following parameters:

- Population size (`p`) takes the integer values $[1, 100]$.
- Selection size (`s`) takes the same values but no more than the population size, that is [1,`p`].

In this case, it is possible to describe the parameters `p` and `s` using surrogate parameters for **irace** that represent a ratio of the original interval as follows:

[4]The t-test does not require that the performance values follow a normal distribution, only that the distribution of sample means does. In practice, the t-test is robust despite large deviations from the assumptions.

```
# name    label   type   range
p        "-p "    i      (1,100)
s_f      "-s "   r      (0.0,1.0)
```

and targetRunner must calculate the actual value of s as $\min(\max(\mathrm{round}(\texttt{s\_f} \cdot \texttt{p}, 1)), 100)$. For example, if the parameter p has value 50 and the surrogate parameter s_f has value 0.3, then s will have value 15.

The processing within targetRunner can also split and join parameters. For example, assume the following parameters:

```
# name    label   type   range
m         "-m "   i      (1,250)
e         "-e "   r      (0.0,2.0)
```

These parameters could be used to define a value $\texttt{m} \cdot 10^{\texttt{e}}$ for another parameter (--strength) not known by **irace**. Then, targetRunner takes care of parsing -m and -e, computing the strength value and passing the parameter --strength together with its value to the target algorithm.

More complex parameter space constraints may be implemented by means of the repairConfiguration function (Section 5.6).

10.8 Unreliable target algorithms and immediate rejection

There are some situations in which the target algorithm may fail to execute correctly. By default, **irace** stops as soon as a call to targetRunner or targetEvaluator fails, which helps to detect bugs in the target algorithm. Sometimes the failure cannot be fixed because it is due to system problems, network issues, memory limits, bugs for which no fix is available, or fixing them is impossible because there is no access to the source code.

In those cases, if the failure is caused by random errors or transient system problems, one may wish to ignore the error and try again the same call in the hope that it succeeds. The option targetRunnerRetries indicates the number of times a targetRunner execution is repeated if it fails. Use this option only if you know additional repetitions could be successful.

If the target algorithm consistently fails for a particular set of configurations, these configurations may be declared as forbidden (Section 5.1.5) so that **irace** avoids them. On the other hand, if the configurations that cause the problem are unknown, the targetRunner should return Inf so that **irace** immediately rejects the failing configuration. This immediate rejection should be used with care according to the goals of the tuning. For example, a configuration that crashes on a particular instance, e.g., by running out of memory, might still be considered acceptable if it gives very good results on other instances. The configurations which were rejected during the execution of **irace** are saved in the Rdata output file (see Section 9.2).

If the configuration budget is specified in total execution time (maxTime option), immediate rejected executions must provide the cost and time (which must be Inf 0). Nevertheless, rejected configurations will be excluded from the execution time estimation and the execution bound calculation.

10.9 Ablation Analysis

The ablation method [6] takes two configurations (source and target) and generates a sequence of configurations that differ between each other just in one parameter, where parameter values in source are replaced by values from target. The sequence can be seen as a "path" from the source to the target configuration. This can be used to find new better "intermediate" configurations or to analyse the impact of the parameters in the performance.

To perform ablation, you can use the `ablation()` R function or the `ablation` command-line executable (see more details below). You may specify the IDs of the source and target configurations. By default, the source is taken as the first configuration evaluated by **irace** and the target as the best overall configuration found. Use the function `plotAblation` to visualize the ablation results (Fig. 9).

```r
ablog <- ablation("irace.Rdata", src = 1, target = 60)
plotAblation(ablog)
```

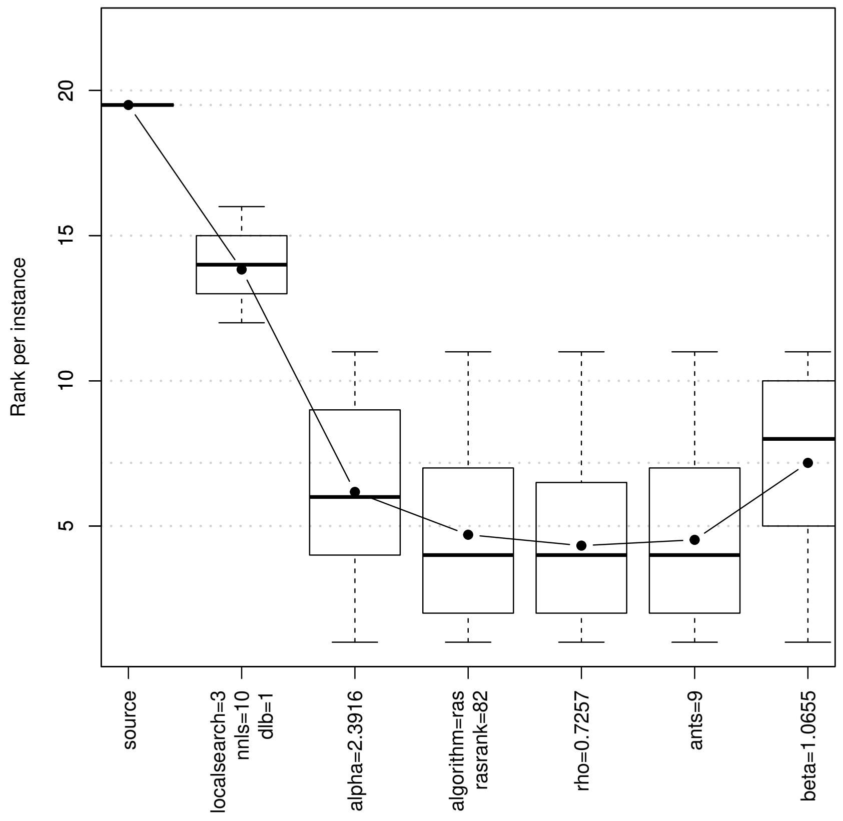

Figure 9: Example of plot generated by `plotAblation()`.

The function returns a list containing the following elements:

`configurations`: A dataframe of configurations tested during ablation.

`instances`: The instances used for the ablation.

`scenario`: Scenario options provided by the user.

`trajectory`: Best configuration IDs at each step of the ablation.

`best`: Best overall configuration found.

We also provide a command-line executable (`ablation.exe` in Windows) that allows you to perform ablation without launching R. It is installed in the same location as the **irace** command-line executable and has the following options:

```
#------------------------------------------------------------------------------
# ablation: An implementation in R of Ablation Analysis
# Version: 4.0.886dd4c
# Copyright (C) 2020--2022
# Manuel Lopez-Ibanez     <manuel.lopez-ibanez@manchester.ac.uk>
# Leslie Perez Caceres    <leslie.perez.caceres@ulb.ac.be>
#
# This is free software, and you are welcome to redistribute it under certain
# conditions.  See the GNU General Public License for details. There is NO
# WARRANTY; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
#------------------------------------------------------------------------------
# installed at: /tmp/RtmpxyWmP6/Rinstcb624a8474d9/irace
# called with: --help
-l,--log-file            Path to the (.Rdata) file created by irace from which
                          the "iraceResults" object will be loaded.
-S,--src                 Source configuration ID. Default: 1.
-T,--target              Target configuration ID. By default the best
                          configuration found by irace.
-P,--params              Specific parameter names to be used for the ablation
                          (separated with commas). By default use all
-t,--type                Type of ablation to perform: "full" will execute each
                          configuration on all "--n-instances" to determine the
                          best-performing one; "racing" will apply racing to
                          find the best configurations. Default: full.
-n,--nrep                Number of replications per instance used in "full"
                          ablation. Default: 1.
   --seed                 Integer value to use as seed for the random number
                          generation. Default: 1234567.
-o,--output-file         Log file to save the ablation log. If "", the results
                          are not saved to a file. Default: log-ablation.Rdata.
   --instances-file       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.
```

10.10 Postselection race

After the configuration process is finished it is possible perform a postselection race by specifying the **irace** option `postselection` with value larger than 0. This option will perform a post-selection race of the set of best configurations of each iteration. The budget assigned for this race is obtained using the `postselection` option which defines a percentage of the **irace** configuration budget. This budget is not considered in the total configuration budget that is, these evaluations are extra computation.

The execution of the postselection race add an element (`psrace.log`) to the `iraceResults` list saved in the **irace** log file. The postselection log consists of a list with the following elements:

`configurations`: Configurations used in the postselection race.

`instances`: Instances used in the in the postselection race.

`maxExperiments`: Configuration budget assigned for the postselection race.

`experiments`: Matrix of experiments in the same format as the `iraceResults$experiments` matrix. The column names indicate the candidate configuration identifier and the row names contain the name of the instances.

`elites`: Elite configurations obtained in the postselection race.

Optionally, it is possible to perform a postselection race with all elite configurations of the iterations or selecting a set of configurations from `iraceResults$allConfigurations`.

```r
# Execute all elite configurations in the iterations
psRace("irace.Rdata", max_experiments = 0.5, elites=TRUE)
# Execute a set of configurations IDs providing budget
psRace("irace.Rdata", conf_ids = c(34, 87, 102, 172, 293), max_experiments = 500)
```

10.11 Parameter importance analysis using PyImp

The **PyImp**[5] tool developed by the AutoML group[6] supports various parameter importance analysis methods using surrogate models. Given a performance dataset of an algorithm configuration scenario, a Random Forest is built to predict performance of all algorithm configurations. Parameter importance analyses are then applied on the prediction model. The model serves as a surrogate for the original target algorithm, so that the algorithm does not need to be executed during the analyses. Three analysis methods are supported, namely fANOVA [10] (functional analysis of variance), forward selection [9], and ablation analysis with surrogates [1]. Note that the **irace** package directly supports ablation (without surrogate models) analysis with and without racing (Section 10.9). Although ablation analysis without surrogates may be more time-consuming, results of the surrogate version may be less accurate than the non-surrogate one.

[5] https://github.com/automl/ParameterImportance
[6] https://www.automl.org/

The `.Rdata` dataset generated by **irace** can be used as input for **PyImp**. The package `irace2pyimp`[7] is able to convert an `irace.Rdata` file into the input format required by **PyImp**. The conversion can be accessed either through the R console (function `irace2pyimp`), or via command line executable `irace2pyimp` (or `irace2pyimp.exe` in Windows) provided by the package. You can find the location of the executable by running this command in R: `system.file(file.path("bin", "irace2pyimp"), package="irace2pyimp", mustWork=TRUE)` or `system.file(file.path("bin", "irace2pyimp.exe"), package="irace2pyimp", mustWork=TRUE)` in Windows.

To see the usage of the executable, please run: `irace2pyimp --help`. For more information on the R function `irace2pyimp`, type in the R console: `?irace2pyimp`.

Given as input an `irace.Rdata` file, the script will generate the following output files:

- `params.pcs`: a text file containing the parameter space definition.
- `runhistory.json`: a JSON file containing the list of algorithm configurations evaluated during the tuning and the performance data obtained.
- `traj_aclib2.json`: a JSON file containing the best configurations after each iteration of **irace**. The last configuration will be used as the target configuration in ablation analysis.
- `scenario.txt`: a text file containing the definition of the tuning scenario.
- `instances.txt`: a text file containing the list of instances.
- `features.csv`: a .csv file containing instance features. If no instance features are provided, the index of each instance will be used as a feature.

PyImp can then be called using the files listed above as input. Several examples on how to use the script and call **PyImp** can be found at `system.file("/examples/",package="irace2pyimp")`.

11 List of command-line and scenario options

Most **irace** options can be specified in the command line using a flag or in the **irace** scenario file using the option name (or setting their value in the `scenario` list passed to the various R functions exported by the package). This section describes the various **irace** options that can be specified by the user in this way.

Relative filesystem paths (e.g., `../scenario/`) given in the command-line are relative to the current working directory (the directory at which **irace** is invoked). However, paths given in the scenario file are relative to the directory containing the scenario file. See also Table 1.

11.1 General options

`--help` *flag:* `-h` *or* `--help` *default:*
Show the list of command-line options of **irace**.

`--version` *flag:* `-v` *or* `--version` *default:*
Show the version of **irace**.

[7] https://github.com/ndangtt/irace2pyimp

`--check` *flag:* `-c` *or* `--check` *default:*
: Check that the scenario and parameter definitions are correct and test the execution of the target algorithm. See Section 4.

`--init` *flag:* `-i` *or* `--init` *default:*
: Initialize the working directory with the template config files. This copies the files in `$IRACE_HOME/templates` to the working directory without overwriting the files with the same names as those of the template files.

`scenarioFile` *flag:* `-s` *or* `--scenario` *default:* `./scenario.txt`
: File that contains the scenario setup and other irace options. All options listed in this section can be included in this file. See `$IRACE_HOME/templates/` for an example. Relative file-system paths specified in the scenario file are relative to the scenario file itself.

`execDir` *flag:* `--exec-dir` *default:* `./`
: Directory where the target runner will be run. The default execution directory is the current directory.

The execution directory must exist before executing **irace**, it will not be created automatically.

`logFile` *flag:* `-l` *or* `--log-file` *default:* `./irace.Rdata`
: File to save tuning results as an R dataset. The provided path must be either an absolute path or relative to `execDir`. See Section 9.2 for details on the format of the R dataset.

`debugLevel` *flag:* `--debug-level` *default:* `0`
: Level of information to display in the text output of **irace**. A value of 0 silences all debug messages. Higher values provide more verbose debug messages. Details about the text output of **irace** are given in Section 9.1.

`seed` *flag:* `--seed` *default:*
: Seed to initiallize the random number generator. The seed must be a positive integer. If the seed is `""` or `NULL`, a random seed will be generated.

`repairConfiguration` *default:*
: User-defined R function that takes a configuration generated by **irace** and repairs it. See Section 5.6 for details.

`postselection` *flag:* `--postselection` *default:* `1`
: Perform a postselection race after the execution of **irace** to consume all remaining budget. Value 0 disables the postselection race. See Section 10.10.

`aclib` *flag:* `--aclib` *default:* `0`
: Enable/disable AClib mode. This option enables compatibility with `GenericWrapper4AC` (`https://github.com/automl/GenericWrapper4AC/`) as `targetRunner` script.

11.2 Elitist irace

`elitist` *flag:* `-e` *or* `--elitist` *default:* `1`
: Enable/disable elitist **irace**.

In the **elitist** version of `irace` [12], elite configurations are not discarded from the race until non-elite configurations have been executed on the same instances as the elite configurations.

Each race begins by evaluating all configurations on a number of new instances. This number is defined by the option `elitistNewInstances`. After the new instances have been evaluated, configurations are evaluated on instances seen in the previous race. Elite configurations already have results for most of these previous instances and, therefore, do not need to be re-evaluated. Finally, after configurations have been evaluated on all these instances, the race continues by evaluating additional new instances.

The statistical tests can be performed at any moment during the race according to the setting of the options `firstTest` and `eachTest`. The elitist rule forbids discarding elite configurations, even if the show poor performance, until the last of the previous instances is seen in the race.

The **non-elitist** version of **irace** can discard elite configurations at any point of the race, instances are not re-used from one race to the next, and new instances are sampled for each race.

`elitistNewInstances` *flag:* `--elitist-new-instances` *default:* `1`
Number of new instances added to each race before evaluating instances from previous races (only for elitist **irace**).

If `deterministic` is `TRUE` then the number of `elitistNewInstances` will be reduced or set to `0` once all instances have been evaluated.

`elitistLimit` *flag:* `--elitist-limit` *default:* `2`
Maximum number of statistical tests performed without successful elimination after all instances from the previous race have been evaluated. If the limit is reached, the current race is stopped. Only valid for elitist **irace**. Use `0` to disable the limit.

11.3 Internal irace options

`sampleInstances` *flag:* `--sample-instances` *default:* `1`
Enable/disable the sampling of the training instances. If the option `sampleInstances` is disabled, the instances are used in the order provided in the `trainInstancesFile` or in the order they are read from the `trainInstancesDir` when`trainInstancesFile` is not provided. For more information about training instances see Section 5.4.

`softRestart` *flag:* `--soft-restart` *default:* `1`
Enable/disable the soft-restart strategy that avoids premature convergence of the probabilistic model. When a sampled configuration is *similar* to its parent configuration, the probabilistic model of these configurations is soft restarted. The soft-restart mechanism is explained in the **irace** paper [12]. The similarity of categorical and ordinal parameters is given by the hamming distance, and the option `softRestartThreshold` defines the similarity of numerical parameters.

`softRestartThreshold` *flag:* `--soft-restart-threshold` *default:* `1e-04`
Soft restart threshold value for numerical parameters.

`nbIterations` *flag:* `--iterations` *default:* `0`
Maximum number of iterations to be executed. Each iteration involves the generation of new configurations and the use of racing to select the best configurations. By default (with 0), **irace** calculates a *minimum* number of iterations as $N^{\text{iter}} = \lfloor 2 + \log_2 N^{\text{param}} \rfloor$,

where N^{param} is the number of non-fixed parameters to be tuned. Setting this parameter may make **irace** stop sooner than it should without using all the available budget. We recommend to use the default value.

`nbExperimentsPerIteration` *flag:* `--experiments-per-iteration` *default:* `0`
Number of runs of the target algorithm per iteration. By default (when equal to 0), this value changes for each iteration and depends on the iteration index and the remaining budget. Further details are provided in the **irace** paper [12]. We recommend to use the default value.

`minNbSurvival` *flag:* `--min-survival` *default:* `0`
Minimum number of configurations needed to continue the execution of each race (iteration). If the number of configurations alive in the race is not larger than this value, the current iteration will stop and a new iteration will start, even if there is budget left to continue the current race. By default (when equal to 0), the value is calculated automatically as $\lfloor 2 + \log_2 N^{\text{param}} \rfloor$, where N^{param} is the number of non-fixed parameters to be tuned.

`nbConfigurations` *flag:* `--num-configurations` *default:* `0`
The number of configurations that will be raced at each iteration. By default (when equal to 0), this value changes for each iteration and depends on `nbExperimentsPerIteration`, the iteration index and `mu`. The precise details are given in the **irace** paper [12]. We recommend to use the default value.

`mu` *flag:* `--mu` *default:* `5`
Parameter used to define the number of configurations to be sampled and evaluated at each iteration. The number of configurations will be calculated such that there is enough budget in each race to evaluate all configurations on at least $\mu + \min(5, j)$ training instances, where j is the index of the current iteration. The value of μ will be adjusted to never be lower than the value of `firstTest`. We recommend to use the default value and, if needed, adjust `firstTest`and `eachTest`, instead.

11.4 Target algorithm parameters

`parameterFile` *flag:* `-p` *or* `--parameter-file` *default:* `./parameters.txt`
File that contains the description of the parameters of the target algorithm. See Section 5.1.

11.5 Target algorithm execution

`targetRunner` *flag:* `--target-runner` *default:* `./target-runner`
Executable or R function that evaluates a configuration of the target algorithm on a particular instance. See Section 5.2 for details.

`targetRunnerLauncher` *flag:* `--target-runner-launcher` *default:*
Executable that will be used to launch the target runner, when `targetRunner` cannot be executed directly (e.g., a Python script in Windows).

`targetCmdline` *flag:* `--target-cmdline` *default:* `{configurationID} {instanceID} {seed} {instance} {bound} {targetRunnerArgs}`
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`). Example:

```
targetRunner="./real_target_runner.py"
targetRunnerLauncher="python"
targetCmdLine="-m {targetRunner} {configurationID} {instanceID}\
 --seed {seed} -i {instance} --cutoff {bound} {targetRunnerArgs}"
```

`targetRunnerRetries` *flag:* `--target-runner-retries` *default:* 0
Number of times to retry a call to `targetRunner` if the call failed.

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

`targetRunnerData` *default:*
Optional data passed to `targetRunner`. This is ignored by the default `targetRunner` function, but it may be used by custom `targetRunner` functions to pass persistent data around.

`targetRunnerParallel` *default:*
Optional R function to provide custom parallelization of `targetRunner`. See Section 6 for more information.

`targetEvaluator` *flag:* `--target-evaluator` *default:*
Optional script or R function that returns a numerical value for an experiment after all configurations have been executed on a given instance using `targetRunner`. See Section 5.3 for details.

`deterministic` *flag:* `--deterministic` *default:* 0
Enable/disable deterministic target algorithm mode. If the target algorithm is deterministic, configurations will be evaluated only once per instance. See Section 5.4 for more information.

If the number of instances provided is less than the value specified for the option `firstTest`, no statistical test will be performed.

`parallel` *flag:* `--parallel` *default:* 0
Number of calls of the `targetRunner` to execute in parallel. Values 0 or 1 mean no parallelization. For more information on parallelization, see Section 6.

`loadBalancing` *flag:* `--load-balancing` *default:* 1
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. See Section 6.

`mpi` *flag:* `--mpi` *default:* 0
Enable/disable use of **Rmpi** to execute the `targetRunner` in parallel using MPI protocol. When `mpi` is enabled, the option `parallel` is the number of slave nodes. See Section 6.

`batchmode` *flag:* `--batchmode` *default:* `0`

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`). See Section 6.

11.6 Initial configurations

`configurationsFile` *flag:* `--configurations-file` *default:*

File containing a table of initial configurations. If empty or `NULL`, **irace** will not use initial configurations. See Section 5.5.

The provided configurations must not violate the constraints described in `parameterFile` and `forbiddenFile`.

11.7 Training instances

`trainInstancesDir` *flag:* `--train-instances-dir` *default:*

Directory where training instances are located; either absolute path or relative to current directory. See Section 5.4.

`trainInstancesFile` *flag:* `--train-instances-file` *default:*

File that contains a list of instances and optionally additional parameters for them. See Section 5.4.

The list of instances in `trainInstancesFile` is interpreted as file-system paths relative to `trainInstancesDir`. When using an absolute path or instances that are not files, set `trainInstancesDir=""`.

11.8 Tuning budget

`maxExperiments` *flag:* `--max-experiments` *default:* `0`

The maximum number of runs (invocations of `targetRunner`) that will be performed. It determines the maximum budget of experiments for the tuning. See Section 10.1.

`minExperiments` *flag:* `--min-experiments` *default:*

The minimum number of runs (invocations of `targetRunner`) that will be performed. If this option is set, then `maxExperiments` is ignored and the actual budget will depend on the number of parameters and `minSurvival`, but it will not be smaller than this value. See Section 10.1.

`maxTime` *flag:* `--max-time` *default:* `0`

The maximum total time for the runs of `targetRunner` that will be performed. The mean execution time of each run is estimated in order to calculate the maximum number of experiments (see option `budgetEstimation`). When `maxTime` is positive, then `targetRunner` **must** return the execution time as its second output. This value and the one returned by `targetRunner` must use the same units (seconds, minutes, iterations, evaluations, . . .). See Section 10.1.

`budgetEstimation` *flag:* `--budget-estimation` *default:* `0.05`

Fraction (smaller than 1) of the budget used to estimate the mean execution time of a configuration. Only used when `maxTime` > 0. See Section 10.1.

`minMeasurableTime` *flag:* `--min-measurable-time` *default:* `0.01`
Minimum time unit that is still (significantly) measureable.

11.9 Statistical test

`testType` *flag:* `--test-type` *default:*
Specifies the statistical test used for elimination:

`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).

We recommend to not use corrections for multiple comparisons because the test typically becomes too strict and the search stagnates. See Section 10.6 for details about choosing the statistical test most appropriate for your scenario.

The default setting of `testType` is `F-test` unless the `capping` option is enabled in which case, the default setting is `t-test`.

`firstTest` *flag:* `--first-test` *default:* `5`
Specifies how many instances are evaluated before the first elimination test.

The value of `firstTest` must be a multiple of `eachTest`.

`eachTest` *flag:* `--each-test` *default:* `1`
Specifies how many instances are evaluated between elimination tests.

`confidence` *flag:* `--confidence` *default:* `0.95`
Confidence level for the elimination test.

11.10 Adaptive capping

`capping` *flag:* `--capping` *default:*
Enable the use of adaptive capping. Capping is enabled by default if `elitist` is active, `maxTime` > 0 and `boundMax` > 0. When using this option, **irace** provides an execution bound to each target algorithm execution (See Section 5.2). For more details about this option See Section 10.3.

`cappingType` *flag:* `--capping-type` *default:* `median`
Specifies the measure used to define the execution bound:

`median` (the median of the performance of the elite configurations)

`mean` (the mean of the performance of the elite configurations)

`best` (the best performance of the elite configurations)

`worst` (the worst performance of the elite configurations).

`boundType` *flag:* `--bound-type` *default:* `candidate`
Specifies how to calculate the performance of elite configurations for the execution bound:

`candidate` (performance of candidates is aggregated across the instances already executed)

`instance` (performance of candidates on each instance).

`boundMax` *flag:* `--bound-max` *default:* `0`
Maximum execution bound for `targetRunner`. It must be specified when capping is enabled.

`boundDigits` *flag:* `--bound-digits` *default:* `0`
Precision used for calculating the execution time. It must be specified when capping is enabled.

`boundPar` *flag:* `--bound-par` *default:* `1`
Penalty used for PARX. This value is used to penalize timed out executions, see Section 10.3.

`boundAsTimeout` *flag:* `--bound-as-timeout` *default:* `1`
Replace the configuration cost of bounded executions with `boundMax`. See Section 10.3.

11.11 Recovery

`recoveryFile` *flag:* `--recovery-file` *default:*
Previously saved **irace** log file that should be used to recover the execution of **irace**; either absolute path or relative to the current directory. If empty or `NULL`, recovery is not performed. For more details about recovery, see Section 8.

11.12 Testing

`--only-test` *flag:* `--only-test` *default:*
Run the configurations contained in the file provided as argument on the test instances. See Section 7.

`testInstancesDir` *flag:* `--test-instances-dir` *default:*
Directory where testing instances are located, either absolute or relative to the current directory.

`testInstancesFile` *flag:* `--test-instances-file` *default:*
File containing a list of test instances and, optionally, additional parameters for them.

`testNbElites` *flag:* `--test-num-elites` *default:* `1`
Number of elite configurations returned by irace that will be tested if test instances are provided. For more information about the testing, see Section 7.

`testIterationElites` *flag:* `--test-iteration-elites` *default:* `0`
Enable/disable testing the elite configurations found at each iteration.

12 FAQ (Frequently Asked Questions)

12.1 Is irace minimizing or maximizing the output of my algorithm?

By default, **irace** considers that the value returned by `targetRunner` (or by `targetEvaluator`, if used) should be **minimized**. In case of a maximization problem, one can simply multiply the value by -1 before returning it to irace. This is done, for example, when maximizing the hypervolume (see the last lines in `$IRACE_HOME/examples/hypervolume/target-evaluator`).

12.2 Are experiments with irace reproducible?

Short answer: Yes, under some conditions.
Long answer: According to the terminology described by López-Ibáñez et al. [13], we define *repeatability* as "*exactly repeating the original experiment, generating precisely the same results*". Following this definition, a run of **irace** is repeatable under the following conditions:

- Same version of **irace**.
- Same version of R (different versions of R may change the behavior of functions used by **irace**).
- The behavior of `targetRunner` is deterministic or exactly reproducible for the same instance, parameter configuration and random seed. Make sure that `targetRunner` uses the seed provided by **irace** to initialize all random number generators used. If the result of `targetRunner` depends on CPU-time, wall-clock time or system load in any way, then `targetRunner` is not reproducible and neither will be **irace**.
- Same random seed (`seed`) given to **irace**.
- Same scenario options (Section 11). Although some options should not affect reproducibility (e.g., `debugLevel`), maintaining a list of such options will be a huge effort, thus the safest assumption is that any change may break reproducibility.
- Same parameter space (Section 5.1), including types, domains, conditions and forbidden configurations. The order of the parameters may also affect reproducibility (the name of the parameters should not) because it affects the order in which random numbers are used.
- Same training instances provided and in the same order (Section 5.4). Even if the instances are sampled randomly (`sampleInstances`), a different initial order will produce a different sample even with the same random seed.
- Same initial configurations (Section 5.5), if any.

12.3 Is it possible to configure a MATLAB algorithm with irace?

Definitely. There are three main ways to achieve this:

1. Edit the `targetRunner` script to call MATLAB in a non-interactive way. See the MATLAB documentation, or the following links.[8][9] You would need to pass the parameter received by `targetRunner` to your MATLAB script.[10][11] There is a minimal example in `$IRACE_HOME/examples/matlab/`.

[8]`http://stackoverflow.com/questions/1518072/suppress-start-message-of-matlab`
[9]`http://stackoverflow.com/questions/4611195/how-to-call-matlab-from-command-line-and-print-to-stdout-before-exiting`
[10]`https://www.mathworks.com/matlabcentral/answers/97204-how-can-i-pass-input-parameters-when-running-matlab-in-batch-mode-in-windows`
[11]`https://stackoverflow.com/questions/3335505/how-can-i-pass-command-line-arguments-to-a-standalone-matlab-executable-running`

2. Call MATLAB code directly from R using the **matlabr** package (https://cran.r-project.org/package=matlabr). This is a better option if you are experienced in R. Define `targetRunner` as an R function instead of a path to a script. The function should call your MATLAB code with appropriate parameters.

3. Another possibility is calling MATLAB directly from a different programming language and write `targetRunner` in that programming language, for example, in Python (see examples in `$IRACE_HOME/examples/target-runner-python/`).[12]

12.4 My program works perfectly on its own, but not when running under irace. Is irace broken?

Every time this was reported, it was a difficult-to-reproduce bug, i.e., a Heisenbug, in the program (target algorithm), not in **irace**. To detect such bugs, we recommend that you use, within `targetRunner`, a memory debugger (e.g., `valgrind`) to run your program. For example, if your program is executed by `targetRunner` as:

```
${EXE} ${FIXED_PARAMS} -i ${INSTANCE} ${CONFIG_PARAMS} 1> ${STDOUT} 2> ${STDERR}
```

then replace that line with:

```
valgrind --error-exitcode=1 ${EXE} ${FIXED_PARAMS} -i ${INSTANCE} \
  ${CONFIG_PARAMS} 1> ${STDOUT} 2> ${STDERR}
```

If there are bugs in your program, they will appear in `$STDERR`, thus do not delete those files. Memory debuggers will significantly slowdown your code, so use them only as a means to find what is wrong with your target algorithm. Once you have fixed the bugs, you should remove the use of `valgrind`.

12.5 irace seems to run forever without any progress, is this a bug?

Every time this problem was reported, the issue was in the target algorithm and not in **irace**. Some ideas for debugging this problem:

- Check that the target algorithm is really not running nor paused nor sleeping nor waiting for input-output.
- Use `debugLevel=3` to see how **irace** calls `target-runner`, run the same command outside **irace** and verify that it terminates.
- Add some output to your algorithm that reports at the very end the runtime and exit code. Verify that this output is printed when **irace** calls your algorithm.
- In `target-runner`, print something to a log file *after* calling your target algorithm. Verify that this output appears in the log file when **irace** is running.
- Set a maximum timeout when calling your target algorithm from `target-runner` (see FAQ 12.6).

[12] https://www.mathworks.com/help/matlab/matlab_external/call-matlab-functions-from-python.html
https://www.mathworks.com/help/matlab/matlab_external/call-user-script-and-function-from-python.html

12.6 My program may be buggy and run into an infinite loop. Is it possible to set a maximum timeout?

We are not aware of any way to achieve this using R. However, in GNU/Linux, it is easy to implement by using the `timeout` command[13] in `targetRunner` when invoking your program.

12.7 When using the mpi option, irace is aborted with an error message indicating that a function is not defined. How to fix this?

Rmpi does not work the same way when called from within a package and when called from a script or interactively. When **irace** creates the slave nodes, the slaves will load a copy of **irace** automatically. If the slave nodes are on different machines, they must have **irace** installed. If **irace** is not installed system-wide, R needs to be able to find **irace** on the slave nodes. This is usually done by setting `R_LIBS`, `.libPaths()` or by loading **irace** using `library()` or `require()` with the argument "`lib.loc`". The settings on the master are not applied to the slave nodes automatically, thus the slave nodes may need their own settings. After spawning the slaves, it is too late to modify those settings, thus modifying the shell variable `R_LIBS` seems the only valid way to tell the slaves where to find **irace**.

If the path is set correctly and the problem persists, please check these instructions:

1. Test that **irace** and **Rmpi** work. Run **irace** on a single machine (submit node), without calling `qsub`, `mpirun` or a similar wrapper around **irace** or R.

2. Test loading **irace** on the slave nodes. However, jobs submitted by `qsub`/`mpirun` may load R packages using a different mechanism from the way it happens if you log directly into the node (e.g., with `ssh`). Thus, you need to write a little R program such as:

   ```
   library(Rmpi)
   mpi.spawn.Rslaves(nslaves = 10)
   paths <- mpi.applyLB(1:10, function(x) {
     library(irace); return(path.package("irace")) })
   print(paths)
   ```

   Submit this program to the cluster like you would submit **irace** (using `qsub`, `mpirun` or whatever program is used to submit jobs to the cluster).

3. In the script `bin/parallel-irace-mpi`, the function `irace_main()` creates an MPI job for our cluster. You may need to speak with the admin of your cluster and ask them how to best submit a job for MPI. There may be some particular settings that you need. **Rmpi** normally creates log files; but **irace** suppresses those files unless `debugLevel` > 0.

Please contact us (Section 13) if you have further problems.

12.8 Error: 4 arguments passed to `.Internal(nchar)` which requires 3

This is a bug in R 3.2.0 on Windows. The solution is to update your version of R.

[13] http://man7.org/linux/man-pages/man1/timeout.1.html

12.9 Warning: `In read.table(filename, header = TRUE, colClasses = "character", : incomplete final line found by ...`

This is a warning given by R when the last line of an input file does not finish with the newline character. The warning is harmless and can be ignored. If you want to suppress it, just open the file and press the `ENTER` key at the end of the last line of the file to end the final line with a newline.

12.10 How are relative filesystem paths interpreted by irace?

The answer depends on where the path appears. Relative paths may appear as the argument of command-line options, as the value of options given in the scenario file, or within various scripts, functions or instance files. Table 1 summarizes how paths are translated from relative to absolute.

Table 1: Translation of relative to absolute filesystem paths.

Relative path appears as ...	**... is relative to ...**
a string within `trainInstancesFile`	`trainInstancesDir`
a string within `testInstancesFile`	`testInstancesDir`
code within `targetRunner` or `targetEvaluator`	`execDir`
the value of `logFile` or `--log-file`	`execDir`
the value of other options in the scenario file	the directory containing the scenario file
the value of other command-line options	invocation (working) directory of **irace**

12.11 My parameter space is small enough that irace could generate all possible configurations; however, irace generates repeated configurations and/or does not generate some of them. Is this a bug?

Typically, **irace** is applied to parameter spaces that are much larger than what can be explored within the budget given. Thus, **irace** does not try to detect whether all possible configurations can be evaluated for the given budget and it does not waste computation time to check for repeated configurations. Thus, if the parameter space is actually very small, the initial random sampling performed by **irace** may generate repeated configurations and/or never generate some configurations, which is not ideal. If you still want to use (non-iterated) racing, the recommended approach is to provide all configurations explicitly to **irace** (Section 5.5) and execute a single race (`nbIterations`=1) with exactly the number of configurations provided (e.g., `nbConfigurations`=240). A future version of **irace** may automatically detect this case and switch to non-iterated racing without having to set additional options. Future versions may also implement computationally cheap checks for repeated configurations.[14]

12.12 On Windows and using `target-runner.py` (a **Python** file), I get the error "`target-runner.py is not executable`"

The issue is that `.py` files are not executable on their own and you need `python.exe` to read the `.py` file and execute it. Linux knows how to do this if the first line of the file is "`#!/usr/bin/python`", however, Windows doesn't know how to do it. In Windows you have 2 options:

[14] If you are interested in implementing this, please contact us!

- Create a `target-runner.bat` file that contains a line similar to (see `templates/windows/target-runner.bat`):

```
C:\path\to\python.exe C:\path\to\target-runner.py %instance% %seed% \
  %candidate_parameters% 1>%stdout% 2>%stderr%
```

- Or convert `target-runner.py` into an `.exe` file, for example, using `auto-py-to-exe`[15], so that you do not need a `.bat` file.

12.13 Error in `socketConnection("localhost", port = port, server = TRUE, lock = TRUE, :` can not open the connection

This error may arise if you activate the `parallel` option of **irace** and your `targetRunner` or `targetEvaluator` tries to setup a parallel cluster or execute code in parallel in a way that interacts badly with the parallel mechanism in R. In this case, you need to either investigate yourself if there is a way for the two parallel mechanisms to co-exist or, if that is not possible, disable parallelism in **irace** or in your code. Note that packages or software used by your `targetRunner` may have a parallel mechanism enabled by default and unknown to you. This is definitely NOT a bug in **irace**.

12.14 irace does not print the call to the `targetRunner` with `debugLevel`=2 when using the `parallel`

This is a limitation of Windows or Rstudio. Running without `parallel` should work. Unfortunately, we cannot fix this limitation in **irace**. If you need to understand how **irace** calls `targetRunner` when running in parallel, you can implement a logging mechanism able to handle parallelism directly inside the `targetRunner`.

13 Resources and contact information

More information about the package can be found on the **irace** webpage:

`https://iridia.ulb.ac.be/supp/IridiaSupp2016-003/index.html`

For questions and suggestions please contact the development team through the **irace** package Google group:

`https://groups.google.com/d/forum/irace-package`

or by sending an email to:

irace-package@googlegroups.com

14 Acknowledgements

We would like to thank all the people that directly or indirectly have collaborated in the development and improvement of **irace**: Prasanna Balaprakash, Zhi (Eric) Yuan, Franco Mascia, Alberto Franzin, Anthony Antoun, Esteban Diaz Leiva, Federico Caselli, Pablo Valledor Pellicer, André de Souza Andrade, and Nguyen Dang (`nttd@st-andrews.ac.uk`).

[15]`https://pypi.org/project/auto-py-to-exe/`

Bibliography

[1] A. Biedenkapp, M. T. Lindauer, K. Eggensperger, F. Hutter, C. Fawcett, and H. H. Hoos. Efficient parameter importance analysis via ablation with surrogates. In S. P. Singh and S. Markovitch, editors, *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press, Feb. 2017. doi: 10.1609/aaai.v31i1.10657.

[2] M. Birattari. On the estimation of the expected performance of a metaheuristic on a class of instances. how many instances, how many runs? Technical Report TR/IRIDIA/2004-001, IRIDIA, Université Libre de Bruxelles, Belgium, 2004.

[3] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-race and iterated F-race: An overview. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss, editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. Springer, Berlin, Germany, 2010. doi: 10.1007/978-3-642-02538-9_13.

[4] M. De Souza, M. Ritt, M. López-Ibáñez, and L. Pérez Cáceres. ACVIZ: A tool for the visual analysis of the configuration of algorithms with irace. *Operations Research Perspectives*, 8: 100186, 2021. doi: 10.1016/j.orp.2021.100186.

[5] M. De Souza, M. Ritt, and M. López-Ibáñez. Capping methods for the automatic configuration of optimization algorithms. *Computers & Operations Research*, 139:105615, 2022. doi: 10.1016/j.cor.2021.105615.

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

[7] C. M. Fonseca, L. Paquete, and M. López-Ibáñez. An improved dimension-sweep algorithm for the hypervolume indicator. In *Proceedings of the 2006 Congress on Evolutionary Computation (CEC 2006)*, pages 1157–1163. IEEE Press, Piscataway, NJ, July 2006. doi: 10.1109/CEC.2006.1688440.

[8] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, Oct. 2009. doi: 10.1613/jair.2861.

[9] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Identifying key algorithm parameters and instance features using forward selection. In P. M. Pardalos and G. Nicosia, editors, *Learning and Intelligent Optimization, 7th International Conference, LION 7*, volume 7997 of *Lecture Notes in Computer Science*, pages 364–381. Springer, Heidelberg, 2013. doi: 10.1007/978-3-642-44973-4_40.

[10] F. Hutter, H. H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning, ICML 2014*, volume 32, pages 754–762, 2014. URL `https://proceedings.mlr.press/v32/hutter14.html`.

[11] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011. URL `http://iridia.ulb.ac.be/IridiaTrSeries/link/IridiaTr2011-004.pdf`. Published in Operations Research Perspectives [12].

[12] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, and M. Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016. doi: 10.1016/j.orp.2016.09.002.

[13] M. López-Ibáñez, J. Branke, and L. Paquete. Reproducibility in evolutionary computation. *ACM Transactions on Evolutionary Learning and Optimization*, 1(4):1–21, 2021. doi: 10.1145/3466624.

[14] C. C. McGeoch. Analyzing algorithms by simulation: Variance reduction techniques and simulation speedups. *ACM Computing Surveys*, 24(2):195–212, 1992. doi: 10.1145/130844.130853.

[15] L. Pérez Cáceres, M. López-Ibáñez, H. H. Hoos, and T. Stützle. An experimental study of adaptive capping in irace. In R. Battiti, D. E. Kvasov, and Y. D. Sergeyev, editors, *Learning and Intelligent Optimization, 11th International Conference, LION 11*, volume 10556 of *Lecture Notes in Computer Science*, pages 235–250. Springer, Cham, Switzerland, 2017. doi: 10.1007/978-3-319-69404-7_17.

[16] M. Schneider and H. H. Hoos. Quantifying homogeneity of instance sets for algorithm configuration. In Y. Hamadi and M. Schoenauer, editors, *Learning and Intelligent Optimization, 6th International Conference, LION 6*, volume 7219 of *Lecture Notes in Computer Science*, pages 190–204. Springer, Heidelberg, 2012. doi: 10.1007/978-3-642-34413-8_14.

Appendix A Installing R

This section gives a quick R installation guide that will work in most cases. The official instructions are available at https://cran.r-project.org/doc/manuals/r-release/R-admin.html

A.1 GNU/Linux

You should install R from your package manager. On a Debian/Ubuntu system it will be something like:

```
sudo apt-get install r-base
```

Once R is installed, you can launch R from the Terminal and from the R prompt install the **irace** package (see Section 3.2).

A.2 OS X

You can install R directly from a CRAN mirror.[16] Alternatively, if you use homebrew, you can just brew the R formula from the science tap (unfortunately it does not come already bottled so you need to have Xcode[17] installed to compile it):

```
brew tap homebrew/science
brew install r
```

Once R is installed, you can launch R from the Terminal (or from your Applications), and from the R prompt install the **irace** package (see Section 3.2).

A.3 Windows

You can install R from a CRAN mirror.[18] We recommend that you install R on a filesystem path without spaces, special characters or long names, such as `C:\R`. Once R is installed, you can launch the R console and install the **irace** package from it (see Section 3.2).

Appendix B targetRunner troubleshooting checklist

If the `targetRunner` script fails to return the output expected by **irace**, it can be sometimes difficult to diagnose where the problem lies. The more descriptive errors provided by your script, the easier it will be to debug it. If `targetRunner` enters an infinite loop, irace will wait indefinitely (see FAQ in Section 12.6). If you are using temporary files to redirect the output of your algorithm, check that these files are properly created. We recommend to follow the structure of the example file (`target-runner`) provided in `$IRACE_HOME/templates`. The following error examples are based on that example file.

In case of failure of `targetRunner`, **irace** will print an error on its output describing which execution of `targetRunner` failed. Follow this checklist to detect where the problem is:

1. Make sure that your `targetRunner` script or program is at the specified location. If you see this error:

[16] https://cran.r-project.org/bin/macosx/
[17] Xcode download webpage: https://developer.apple.com/xcode/download/
[18] https://cran.r-project.org/bin/windows/

```
Error: == irace == target runner './tuning/target-runner' does not exist
```

it means that **irace** cannot find the `target-runner` file. Check that the file is at the path specified by the error.

2. Make sure that your `targetRunner` script is an executable file and the user running **irace** has permission to execute it. The following errors:

   ```
   Error: == irace == target runner './tuning/target-runner' is a directory,
   not a file
   ```

   or

   ```
   Error: == irace == target runner './tuning/target-runner' is not executable
   ```

   mean that your `targetRunner` is not an executable file. In the first case, the script is a folder and therefore there must be a problem with the name of the script. In the second case, you must make the file executable, which in GNU/Linux can be done by:

   ```
   chmod +x ./tuning/target-runner
   ```

3. If your `targetRunner` script calls another program, make sure it is at the location described in the script (variable `EXE` in the examples and templates). A typical output for such an error is:

   ```
   Error: == irace == running command ''./tuning/target-runner' 1 8 676651103
   ./tuning/Instances/1000-16.tsp --ras --localsearch 2 --alpha 4.03 --beta 1.89
   --rho  0.02 --ants 37 --nnls 48 --dlb 0 --rasranks 15 2>\&1' had status 1
   == irace == The call to target_runner_default was:
   ./tuning/target-runner 1 8 676651103 ./tuning/Instances/1000-16.tsp --ras
   --localsearch 2 --alpha 4.03 --beta 1.89 --rho  0.02 --ants 37 --nnls 48
   --dlb 0 --rasranks 15
   == irace == The output was:
   Tue May  3 19:00:37 UTC 2016: error: ./bin/acotsp: not found or not executable
   (pwd: ./tuning/acotsp-arena)
   ```

   You may test your script by copying the command line shown in the error and executing `target-runner` directly on the execution directory (`execDir`). In this case, the command line is:

   ```
   ./tuning/target-runner 1 8 676651103 ./tuning/Instances/1000-16.tsp --ras \
    --localsearch 2 --alpha 4.03 --beta 1.89 --rho 0.02 --ants 37 --nnls 48 \
    --dlb 0 --rasranks 15
   ```

   This executes the `targetRunner` script as **irace** does. The output of this script must be only one number.

4. If your `targetRunner` file is an executable script in Python, R, Perl, Bash or some other interpreted programming language, make sure that the interpreter specified in the first line of the file exists at the correct location. For example, if the first line of `target-runner.py` is:

   ```
   #|/usr/bin/python
   ```

   Then make sure that `/usr/bin/python` exists and it is executable. Otherwise, you will get an error such as:

```
Error: == irace == error in running command
```

5. Check that your `targetRunner` script is actually returning one number as output. For example:

   ```
   Error: == irace == The output of './tuning/target-runner 1 25 365157769
    ./tuning/Instances/1000-31.tsp --ras --localsearch 1 --alpha 0.26 --beta
    6.95 --rho 0.69 --ants 56 --nnls 10 --dlb 0 --rasranks 7' is not numeric!
   == irace == The output was:
   Solution: 24479793
   ```

   In the example above, the output of `target-runner` is "`Solution:  24479793`", which is not a number. If `target-runner` is parsing the output of the target algorithm, you need to verify that the code only parses the solution cost value.

6. Check that your `targetRunner` script is creating the output files for your algorithm. If you see an error as:

   ```
   == irace == The output was: Tue May  3 19:41:40 UTC 2016:
   error: c1-9.stdout: No such file or directory
   ```

   The output file of the execution of your algorithm has not been created (check permissions) or has been deleted before the result can be read.

7. Other errors can produce the following output:

   ```
   == irace == The output was: Tue May  3 19:49:06 UTC 2016:
   error: c1-23.stdout: Output is not a number
   ```

   This might be because your `targetRunner` script is not executing your algorithm correctly. To further investigate this issue, comment out the line that eliminates the temporary files that saves the output of your algorithm. Similar to this one

   ```
   rm -f "${STDOUT}" "${STDERR}"
   ```

   Execute directly the `targetRunner` command-line that is provided in the error message, look in your execution directory for the files that are created. Check the `.stderr` file for errors and the `.stdout` file to see the output that your algorithm produces.

8. Some command within `targetRunner` may not be working correctly. In that case, you must debug the commands individually exactly as **irace** executes them. In order to find where the problem is, print the commands to a log file before executing them. For example:

   ```
   echo "$EXE ${FIXED_PARAMS} -i $INSTANCE ${CONFIG_PARAMS}" >> ${STDERR}.log
   $EXE ${FIXED_PARAMS} -i $INSTANCE ${CONFIG_PARAMS} 1> ${STDOUT} 2> ${STDERR}
   ```

   then look at the `$STDERR.log` file corresponding to the `targetRunner` call that failed and execute/debug the last command there.

9. If the language of your operating system, the `target-runner` or the target algorithm is not English, **irace** may not be able to recognize the numbers generated by `target-runner`. We recommend that you run **irace**, the `target-runner` and the target algorithm under an English locale (or make sure that their languages and number format are compatible).

10. It is possible that transient bugs in the target algorithm are only visible when running within **irace**, and all commands within `targetRunner` appear to work fine when executed directly in the command-line outside **irace**. See FAQ in Section 12.4) for suggestions on how to detect such bugs.

11. If your `targetRunner` script works when running irace with `parallel=0` but it fails when using higher number of cores, this may be due to any number of reasons:

    - If you submit jobs through a queuing system, the running environment when using the queuing system may not be the same as when you launch **irace** yourself. The queuing system may also send the job to different machines depending on the number of CPUs requested. One way to test this is to submit the failing execution of `targetRunner` to the queuing system, and specifically to any problematic machine.
    - When using MPI, some calls to `targetRunner` may run on different computers than the one running the master **irace** process. See FAQ in Section 12.7.
    - Does `targetRunner` read or create intermediate files? These files may cause a race condition when two calls to `targetRunner` happen at the same time. You have to make sure that parallel runs of `targetRunner` do not interfere with each other's files.
    - Maybe these files consume too much memory or fill the filesystem when there are simultaneous `targetRunner` calls? Moreover, queuing systems have stricter limits for computing nodes than for the submit/host node.
    - Does the machine or the queuing system impose any limits on number of processes or CPU/memory/filesystem usage per job? Such limits may only trigger when more than one process is executed in parallel, killing the `targetRunner` process before it has a chance to print anything useful. In that case, **irace** may not detect the the program finished unexpectedly, only that the expected output was not printed.

Appendix C targetEvaluator troubleshooting checklist

Even if `targetRunner` appears to work, the use of `targetEvaluator` may lead to other problems. The same checklist of `targetRunner` can be followed here. In addition, we list here other potential problems unique to `targetEvaluator`:

1. If `targetEvaluator` fails only in the second or later iteration, this may because output files or data generated by a previous call to `targetRunner` are missing. Elite configurations are never re-executed on the same instance and seed pair, that is, **irace** will call only once `targetRunner` for each pair of configuration ID and instance ID. However, `targetEvaluator` is always re-executed, which takes into account any updated information (normalization bounds, reference sets/points, best-known values, etc.). Thus, any files or data generated by `targetRunner` for a given configuration must remain available to `targetEvaluator` as long as that configuration is alive. The list of alive configurations is passed to `targetEvaluator`, which may decide then which data to keep or remove.

Appendix D Glossary

Parameter tuning: Process of searching good settings for the parameters of an algorithm under a particular tuning scenario (instances, execution time, etc.).

Scenario: Settings that define an instance of the tuning problem. These settings include the algorithm to be tuned (target), budget for the execution of the target algorithm (execution time, evaluations, iterations, etc.), set of problem instances and all the information that is required to perform the tuning.

Target algorithm: Algorithm whose parameters will be tuned.

Target parameter: Parameter of the target algorithm that will be tuned.

irace option: Configurable option of **irace**.

Elite configurations: Best configurations found so far by **irace**. New configurations for the next iteration of **irace** are sampled from the probabilistic models associated to the elite configurations. All elite configurations are also included in the next iteration.

`$IRACE_HOME`: The filesystem path where **irace** is installed. You can find this information by opening an R console and executing:

```r
system.file(package = "irace")
```

Appendix E NEWS

NEWS

```
**If you are viewing this file on CRAN, please check [latest news on GitHub](https://github.com/MLopez-Ibanez/irace/blob

# irace 4.0

## Major breaking changes

 * Requires R version >= 4.0

 * Logfiles `*.Rdata` use format version 3, which can only be read by R version >= 3.5.

 * The scenario options `forbiddenFile` and `forbiddenExps` have been removed
   and will give an error if present.  Forbidden configurations are now
   specified in the parameter space description. See the example in
   `readParameters()`.

 * The scenario option `digits` has been removed and will give an error if
   present. The number of `digits` for real-valued parameters is now specified
   in the parameter space description. See the example in `readParameters()`.

 * The default value of the scenario option `trainInstancesDir` is now `""`.
   The previous default value of `"./Instances"` often caused confusion to
   users not using files as training instances.

 * The `scenario` object now includes the `parameters` object. Thus
   functions such as `irace()`, which previously took as arguments both
   `scenario` and `parameters`, now only take `scenario`. This also means that
   the log file `irace.Rdata` does not contain a separate `parameters` element
   since this element can now be found within `scenario`.

 * The default value of the scenario option `softRestartThreshold` is now
   0.0001 and does not depend on `digits`.

 * The command-line executables `irace` and `ablation` (`irace.exe` and
  `ablation.exe` in Windows) will load the version of the `irace` package that
  is found in the same path where the executables are. In earlier versions,
  the executables will always load the version found via `base::.libPaths()`.
  This change allows installing multiple versions of the irace package in
  different locations and each executable will use its corresponding version.
  The correct location can be verified by looking at the line `"installed at:"`
  printed in the output.

* Adaptive capping is now enabled by default if `maxTime > 0` and `maxBound > 0`.
  It can be disabled with `--capping 0` in the command-line options or `capping=0` in the scenario options. See [Pérez-
                                  (Leslie Pérez Cáceres, Manuel López-Ibáñez)

* The scenario option `targetRunnerLauncherArgs`, introduced in version 3.5,
  was removed and replaced by `targetCmdline`, which is more flexible (fixes #38).
  Please see the user-guide for details.

* Command-line options in joined form, given as `"--log-file= --check"`, without
  any argument after the `'='` will be interpreted as an empty argument,
  equivalent to using `logFile=""` in `scenario.txt`.

* irace will now give an error if you try to recover from a file generated by
  a different version of irace, since such attempts typically end up in errors
  that are difficult to understand.

* irace warns about using `'&&'` and `'||'` instead of `'&'` and `'|'` in
  parameter conditions and forbidden expressions. A future version of irace
  will reject those uses as errors.

* The internal function `irace.reload.debug()` has been removed.
  Use `devtools::reload()` instead.

* The column `"instance"` of the `instancesList` data frame stored in the
  logFile has been renamed to `"instanceID"`.  This data frame should not be
  accessed directly. Instead use the new function `get_instanceID_seed_pairs()`.

* Using `maxTime > 0` with `elitist=0` now gives a clear error rather than fail later.
                                                 (fix #65, reported by @DE0CH)

* `path_rel2abs()` will not expand symlinks to avoid problems with Python's venv.
                                                 (fix #64, reported by @DE0CH)

* Expansion of `'~'` in Windows now follows the definition of `fs::path_expand()` rather than `base::path.expand()`.

* irace is now more strict in enforcing runtime bounds given with `scenario$boundMax`
  and will stop with an error if the `target-runner` reports a runtime larger than the given bound.

* All functions that contained a period (`'.'`) in the name have been renamed to use `'_'` instead.

* The periods (`'.'`) in the arguments of `scenario$targetRunnerParallel`
  and `scenario$targetEvaluator` have also been replaced by `'_'`.

* The environment `.irace` that was available in the log file under
  `iraceResults$state$.irace` is replaced directly by `iraceResults$state`.
  It contains similar information but some entries have been renamed. For
  example, the `experimentLog` data frame is now called `experiment_log`
  and it is a [`data.table`](https://r-datatable.com).

* The interface of `psRace()` has been simplified.

* `irace` will automatically execute a post-selection race (`psRace()`) using
  any remaining budget (currently only when `maxTime == 0`). To disable
  this behavior, set the scenario option `postselection` (`--postselection`)
```

```
   to `0`.


## New features and improvements

 * `sampleUniform()` and `sampleModel()` are significantly faster thanks to using [`data.table`](https://r-datatable.com

 * Initial configurations are sampled using Sobol low-discrepancy sequences using `spacefillr::generate_sobol_set()`. Th

 * Parameter spaces can be constructed programmatically using `parametersNew()`. See the documentation for details.

 * Ablation will report configurations that produced the same results, which
   indicates parameter values that have the same effect on the target algorithm,
   possibly indicating a bug in the target algorithm.

 * New option `instancesFile` of `ablation()` for using either the training
   instances, the test instances or reading instances from a given file.

 * New option `nrep` of `ablation()` specifies the number of replications per
   instance used in `"full"` ablation. It replaces the previous parameter
   `n_instances`, whose definition was more difficult to use correctly.

 * Matrix operations are faster thanks to `matrixStats`.

 * New scenario option `blockSize` for defining blocks of instances.
   Configurations are only eliminated after evaluating a complete block and
   never in the middle of a block. This is useful for scenarios when there are
   clearly defined instance classes and the best configuration should be
   balanced among them. In that case, `trainInstancesFile` should be written
   so that each block contains one instance of each class and
   `blockSize` is set to the number of classes.

 * New scenario option `targetRunnerTimeout`: Timeout in seconds of any
   `targetRunner` call (only applies to `target-runner` executables not to R
   functions).

 * `plotAblation()` has several new options:
    - `type='rank'` to plot ranks per instance instead of raw cost values.
    - `n` to limit the number of parameters shown in the plot.
    - `width` replaces `pdf.width`.
    - `height` sets the height of the plot in the PDF file.

 * The previously internal function `check.output.target.runner` is renamed to
   `check_output_target_runner` and exported to allow users who write their own
   `targetRunnerParallel` to check the output and repair it if possible.
   (Deyao Chen)

 * New functions `read_ablogfile()`, `has_testing_data()`, `irace_summarise()`.

 * New functions `get_random_seed()`, `set_random_seed()`,
   `restore_random_seed()` useful for writing `targetRunner` functions in R.

 * New function `get_instanceID_seed_pairs()` to get the pairs of instanceID
   and random seed used during the races (and optionally the actual instances).

 * The `parameters` object now stores the number of `digits` (decimal places
   after the point) for each parameter of type `r`. As a result, the
   `repairConfiguration` function (see `defaultScenario()`) only needs two
   arguments: `configuration` and `parameters`. See examples in the user-guide.

 * `readScenario()` (and command-line irace) do not require a `scenario.txt` file.
                                                        (Contributed by @DEOCH)
 * `read_pcs_file()` now supports forbidden configurations.

 * When testing, `irace` now prints the random seed used for each instance as an additional column.

 * The package provides a new executable `target-runner-dummy` (or
   `target-runner-dummy.exe` in Windows) for the purposes of testing.  It may
   also be useful for understanding the typical setup of `irace`.

 * New scenario option `minExperiments` to set a minimum budget of runs.
                                             (proposed by @Saethox, fixes #58)

 * New function `multi_irace()` for executing multiple runs of irace with the
   same or different scenarios and parameters, possibly in parallel.
                                                    (Contributed by @Saethox)

## Fixes

 * `ablation_cmdline()` and `plotAblation()` no longer create an empty `Rplots.pdf` file when specifying an output PDF f

 * Fix #66: when using `maxTime > 0`, irace estimates the time per run by
   executing 2 configurations on `firstTest` instances and adjusts `boundMax`
   to not go over `budgetEstimation`. This may result in a smaller `boundMax`
   than before. To reduce this impact, the default value of `budgetEstimation`
   is now `0.05` instead of `0.02`.
                                    (Manuel López-Ibáñez, reported by @DE0CH)

 * Fix #55: Configurations provided may use `<NA>` in addition to `NA` to denote
   the missing value of a disabled parameter.
                                   (Manuel López-Ibáñez, reported by @TheIronBorn)

 * Fix #44: irace now will give an error if the domain of real-valued (r)
   parameters would change depending on the value of `'digits'`. The solution
   is to increase the value of `'digits'` or adjust the domain.
                                    (Manuel López-Ibáñez, reported by @mb706)

 * If scenario option `targetRunnerParallel` is set, irace no longer tries to
   initialize a parallel environment or setup MPI. It is now the responsibility
   of the user to do that before calling irace or within the function assigned
   to `targetRunnerParallel`.

 * irace no longer sets `option(error=utils::recover())` in debug mode to avoid issues
   when calling irace from Python. The user can set this if desired.

 * Fix bug failing to restart with parameters that have dependent domains.

 * Fix bug with `sampleInstances=FALSE` that could re-evaluate the same
   (instance, seed) pair more than once.

 * Fix bug when using `targetRunnerLauncher` and `targetRunner` contain whitespace.

 * Fix bug in `ablation_cmdline()` about missing `scenario` object.

 * `ablation()` will now save and restore the previous random seed.

 * `ablation()` will detect if the logfile (e.g., `irace.Rdata`) is incomplete.

 * `readConfigurationsFile()` now handles parameters with dependent domains.

 * Fix #71: Ensure `".ID."` is the first column in `checkTargetFiles()` (Manuel López-Ibáñez, reported by @ivan1arriola)
```

```
# irace 3.5

## New features and improvements

 * Handling of dependent parameter domains: These should be specified in the
   parameter domain definition and, for now, only numerical parameter can
   define dependent domains. A numerical domain can be dependent on one bound,
   e.g. `(1, "param1*2")`, where the dependent bound can include basic
   arithmetic operators.          (Leslie Pérez Cáceres, Manuel López-Ibáñez)

 * The package now provides an `ablation` executable (`ablation.exe` in
   Windows) that makes easier to perform ablation analysis without having any R
   knowledge.

 * The interface to functions `ablation()` and `plotAblation()` has been
   simplified. The `ablation()` function now allows overriding scenario
   settings. The `plotAblation()` function will not create the plot if the
   ablation log does not contain a complete ablation.
                                                    (Manuel López-Ibáñez)

 * The argument `n.instances` of `ablation()` has been renamed to `n_instances`
   and it is now a factor that multiplies `scenario$firstTest`.
                                                         (Manuel López-Ibáñez)

 * New command-line option `--quiet` to run without producing any output
   except errors (also available as a scenario option).
                                                     (Manuel López-Ibáñez)

 * New command-line option `--init` to initialize a scenario. (Deyao Chen)

 * Added support for HTCondor cluster framework to `--batchmode`.
                                                     (Filippo Bistaffa)

 * `--check` now also check the contents of `configurationsFile` and runs
   configurations provided via `initConfigurations`.
                         (Manuel López-Ibáñez, reported by Andreea Avramescu)

 * New scenario options `targetRunnerLauncher` and `targetRunnerLauncherArgs`
   to help in cases where the target-runner must be invoked via another
   software with particular options (such as `python.exe` in Windows).
                                                           (Manuel López-Ibáñez)

 * New scenario option `minMeasurableTime`.
                                                     (Manuel López-Ibáñez)

 * An error is produced if a variable set in the scenario file is not known to
   irace.  If your scenario file contains R code, then use variable names
   beginning with a dot `'.'`, which will be ignored by irace.
                                                    (Manuel López-Ibáñez)

 * Plotting functions have been moved to the new package
   [iraceplot](https://auto-optimization.github.io/iraceplot/).  In particular,
   `configurationsBoxplot()` is replaced by `iraceplot::boxplot_training()` and
   `iraceplot::boxplot_test()`; `parallelCoordinatesPlot()` is replaced by
   `iraceplot::parallel_cat()` and `iraceplot::parallel_coord()`; and
   `parameterFrequency()` is replaced by `iraceplot::sampling_frequency()`.
                                  (Leslie Pérez Cáceres, Manuel López-Ibáñez)

 * The user-guide now contains a detailed section on "Hyper-parameter
   optimization of machine learning methods".
                                                     (Manuel López-Ibáñez)
 * When `testType="F-test"` and only two configurations remain, the elimination
   test now uses the pseudo-median estimated by the Wilcoxon signed-rank test
   to decide which configuration is the best one instead of comparing the
   median difference.
                                                        (Manuel López-Ibáñez)

 * New functions `testing_fromlog()` and `testing_fromfile()` for independently
   executing the testing phase. The function `testing.main()` was removed as it
   is superseded by the new ones.
                                                       (Manuel López-Ibáñez)

 * New function `read_logfile()` to easily read the log file produced by irace.
                                                        (Manuel López-Ibáñez)

 * New function `printParameters()` that prints a parameters R object as a valid input text.
                                                        (Manuel López-Ibáñez)

 * `irace2pyimp` moved to its own R package.
                                                      (Manuel López-Ibáñez)

 * Generating the file `irace.Rdata` may be disabled by setting `logFile=""`.
                                (Manuel López-Ibáñez, reported by Johann Dreo)

 * `path_rel2abs()` and `checkParameters()` are now exported so that other
   packages may use them.
                                                        (Manuel López-Ibáñez)

 * `path_rel2abs()` also searches in system paths.     (Manuel López-Ibáñez)

 * `readConfigurationsFile()` will now detect duplicated configurations and
   error.                                          (Manuel López-Ibáñez)

 * The interface to functions `getFinalElites()`, `getConfigurationById()` and
   `getConfigurationByIteration()` has been simplified.

 * The package provides a `irace.sindef` file that may be used for building a
   standalone container of irace using Singularity. See the `README.md` file
   for instructions.                                (Contributed by Johann Dreo)

 * New example `examples/target-runner-python/target-runner-python-win.bat`
   contributed by Levi Ribeiro.

 * New helper script in `bin/parallel-irace-slurm` to launch `irace` in [SLURM](https://slurm.schedmd.com/) computer clu
                                                        (Manuel López-Ibáñez)

 * Rename `scenario.update.paths()` to `scenario_update_paths()`. The old name is deprecated. (Manuel López-Ibáñez)

## Fixes

 * Correctly handle clear out-performance cases despite strong bi-modality.
                                             (Reported by Nguyen Dang,
                                             fixed by Manuel López-Ibáñez)

 * Fix error when recovering from a parallel run on Windows.
                             (Manuel López-Ibáñez, reported by Tarek Gamal)

 * `testNbElites` now controls how many iteration elites are tested when
   `testIterationElites=1`. This is the documented behavior in the user guide.
                            (Manuel López-Ibáñez, reported by Marcelo de Souza)

 * Fixes to the Matlab example. (Manuel López-Ibáñez)
```

~~~~
 * The default of `testType` is now set to `t-test` when capping is enabled.
                       (Manuel López-Ibáñez, reported by Jovana Radjenovic)

 * Fix various issues in the user guide.
                       (Manuel López-Ibáñez, reported by Jovana Radjenovic)

 * Remove duplicated elites.
                       (Manuel López-Ibáñez, reported by Federico Naldini)

 * Fix (#7): warnings with partial matched parameters.
                               (Manuel López-Ibáñez, reported by Marc Becker)

 * Fix (#10): wrong assert with `elitist=0`.            (Manuel López-Ibáñez)

 * Fix (#12): irace can be run with [FastR](https://www.graalvm.org/22.1/docs/getting-started/#run-r).

 * Fix (#13): Maximum number configurations immediately rejected reached.
                                                         (Manuel López-Ibáñez)

 * Fix: when setting the scenario file in the command-line, `scenarioFile` was
   not set correctly. The correct scenario was used, however, the debug output
   and the value stored in the log / recovery file was wrong.
                      (Manuel López-Ibáñez, reported by Richard Schoonhoven)

 * With `sampleInstances = FALSE`, elitist irace does not change the order of
   instances already seen.  However, if you want to make sure that the order of
   the instances is enforced, you also need to set `elitistNewInstances=0`.

 * The function `irace.usage()` was removed. It was not really useful for R
   users as the same result can be obtained by calling
   `irace.cmdline("--help")`.
                                                        (Manuel López-Ibáñez)


# irace 3.4.1  (31/03/2020)

 * `NEWS` converted to markdown.

 * Fix CRAN error on Solaris.


# irace 3.4  (30/03/2020)

 * `irace2pyimp` function and executable (`irace2pyimp.exe` on Windows) to
   convert .Rdata files generated by irace to the input files required by the
   parameter importance analysis tool PyImp
   (https://github.com/automl/ParameterImportance).
                                          (Nguyen Dang, Manuel López-Ibáñez)

 * Initial configurations may also be provided directly in R using
   `scenario$initConfigurations`
                                                  (Manuel López-Ibáñez)

 * Rdata files are saved in version 2 to keep compatibility with older R
   versions.
                                                  (Manuel López-Ibáñez)
 * Fix invalid assert with ordered parameters:       (Leslie Pérez Cáceres)

    ```
    value >= 1L && value <= length(possibleValues) is not TRUE
    ```
~~~~

~~~
 * The `irace` executable (`irace.exe` on Windows) is a compiled binary instead
   of a script. On Windows, `irace.exe` replaces `irace.bat`
                                                    (Manuel López-Ibáñez)

 * `inst/examples/Spear` contains the Spear (SAT solver) configuration scenario.
                                                    (Manuel López-Ibáñez)

 * Fixed bug when reporting minimum `maxTime` required.
                                            (Reported by Luciana Salete Buriol,
                                             fixed by Manuel López-Ibáñez)

 * Fixed bug detected by assert:

    ```R
    all(apply(!is.na(elite.data$experiments), 1, any)) is not TRUE
    ```

     (Reported by Maxim Buzdalov, fixed by Manuel López-Ibáñez)


# irace 3.3 (26/04/2019)

 * Fix buggy test that breaks CRAN.                   (Manuel López-Ibáñez)

 * Do not print "23:59:59" when wall-clock time is actually close to zero.
                                                        (Manuel López-Ibáñez)

# irace 3.2 (24/04/2019)

 * Fix `irace --check --parallel 2` on Windows.         (Manuel López-Ibáñez)

 * Values of real-valued parameter are now printed with sufficient precision to
   satisfy `digits` (up to `digits=15`).
                                                        (Manuel López-Ibáñez)

 * It is possible to specify `boundMax` without capping.
                                  (Leslie Pérez Cáceres, Manuel López-Ibáñez)

 * `irace --check` will exit with code 1 if the check is unsuccessful
                                                        (Manuel López-Ibáñez)

 * Print where irace is installed with `--help`.          (Manuel López-Ibáñez)

 * irace will now complain if the output of `target-runner` or `target-evaluator`
   contains extra lines even if the first line of output is correct. This is to
   avoid parsing the wrong output. Unfortunately, this may break setups that
   relied on this behavior. The solution is to only print the output that irace
   expects.
                                                        (Manuel López-Ibáñez)

 * Completely re-implement `log` parameters to fix several bugs. Domains that
   contain zero or negative values are now rejected.
                                  (Leslie Pérez Cáceres, Manuel López-Ibáñez)

 * New option `aclib=` (`--aclib 1`) enables compatibility with the
   GenericWrapper4AC (https://github.com/automl/GenericWrapper4AC/) used by
   AClib (http://aclib.net/). This is EXPERIMENTAL. `--aclib 1` also sets
   digits to 15 for compatibility with AClib defaults.
                                                         (Manuel López-Ibáñez)

 * Fix printing of output when capping is enabled.
                                                         (Manuel López-Ibáñez)
~~~

```
 * `checkTargetFiles()` (`--check`) samples an instance unless
   `sampleInstances` is FALSE.                        (Manuel López-Ibáñez)

 * Fix symbol printed in elimination test.            (Manuel López-Ibáñez)

 * Use `dynGet()` to find `targetRunner` and `targetEvaluator`.
   As a result, we now require R >= 3.2.
                                                      (Manuel López-Ibáñez)

 * All tests now use `testthat`.                      (Manuel López-Ibáñez)

 * New function `scenario.update.paths()`             (Manuel López-Ibáñez)

 * Fix assert failure that may happen when `elitistNewInstances` is larger than
   `firstTest`. Reported by Jose Riveaux.             (Manuel López-Ibáñez)

 * Fix bug in `checkTargetFiles()` (`--check`) with capping.
                                                     (Leslie Pérez Cáceres)

 * Clarify a few errors/warnings when `maxTime > 0`.
                   (Manuel López-Ibáñez, suggested by Haroldo Gambini Santos)


# irace 3.1  (12/07/2018)

 * Use testthat for unit testing.                     (Manuel López-Ibáñez)

 * Allow instances to be a list of arbitrary R objects (`mlr` bugfix).
                                                      (Manuel López-Ibáñez)

# irace 3.0  (05/07/2018)

 * irace now supports adaptive capping for computation time minimization.
   The default value of the `testType` option is t-test when adaptive capping
   is enabled. Please see the user-guide for details.
                                (Leslie Pérez Cáceres, Manuel López-Ibáñez)

 * The package contains an `ablation()` function implementing the ablation
   method for parameter importance analysis by Fawcett and Hoos (2016).
                                 (Leslie Pérez Cáceres, Manuel López-Ibáñez)

 * New option `postselection` executes a post-selection race.
                                                     (Leslie Pérez Cáceres)

 * At the end of each race, if the race stops before evaluating all instances
   seen in previous races, then the best overall may be different than the best
   of the race. We now print the best overall (best-so-far). Elites evaluated
   on more instances are considered better than those evaluated on fewer.
                                (Manuel López-Ibáñez, Leslie Pérez Cáceres)

 * Last active parameter values of numerical parameters (`i` and `r`) are carried
   by the sampling model. When a value must be assigned and the parameter was
   previously not active, the sampling is performed around the last value.
                               (Leslie Pérez Cáceres, Manuel López-Ibáñez)

 * R help pages are now generated with Roxygen2.
                               (Leslie Pérez Cáceres, Manuel López-Ibáñez)

 * The user guide documents `--version`, `--help`, and `--check`.
                                                     (Manuel López-Ibáñez)
 * A return value of `Inf` from `targetRunner`/`targetEvaluation` results in
   the immediate rejection of the configuration without any further evaluation.
   This is useful for handling unreliable or broken configurations that should
   not stop irace.                                          (Manuel López-Ibáñez)

 * Numerical parameters may be sampled on a logarithmic scale using `i,log`
   or `r,log`.                                                 (Alberto Franzin)

 * New `target-runner.bat` for Windows contributed by André de Souza Andrade.

 * Fixed all shell scripts calling functions before defining them, which is not
   portable.
                                                           (Manuel López-Ibáñez)

 * Fixed `--parallel` bug in Windows that resulted in
   `Error in checkForRemoteErrors(val)`.
                                                            (Manuel López-Ibáñez)

 * Improve error message when no training instances are given.
                                                            (Manuel López-Ibáñez)


# irace 2.4 (03/08/2017)

 * The output of irace now specifies in which order, if any, configurations are
   printed.
                             (Manuel López-Ibáñez, suggested by Markus Wagner)

 * Several fixes for handling paths in Windows.
                                                       (Manuel López-Ibáñez)

 * `readConfigurationsFile()` now has a `text=` argument, which allows reading
   configurations from a string.
                                                       (Manuel López-Ibáñez)

 * User-provided functions (targetRunner, targetEvaluator and
   repairConfiguration) and user-provided conditions for forbidden
   configurations are now byte-compiled when read, which should make their
   evaluation noticeably faster.
                                                        (Manuel López-Ibáñez)

 * The argument `'experiment'` passed to the R function `targetRunner` does not
   contain anymore an element `'extra.params'`. Similarly, the `'scenario'`
   structure does not contain anymore the elements `'instances.extra.params'` and
   `'testInstances.extra.params'`. Any instance-specific parameters values now
   form part of the character string that defines an instance and it is up to
   the user-defined `targetRunner` to parse them appropriately. These changes
   make no difference when targetRunner is an external script, or when
   instances and instance-specific parameter values are read from a file.
                                                          (Manuel López-Ibáñez)

# irace 2.3

 * Fix bug that will cause `iraceResults$experimentLog` to count calls to
   `targetEvaluator` as experiments, even if no call to `targetRunner` was
   performed. This does not affect the computation of the budget consumed and,
   thus, it does not affect the termination criteria of irace. The bug triggers
   an assertion that terminates irace, thus no run that was successful with
   version 2.2 is affected.
                                                       (Manuel López-Ibáñez)

# irace 2.2
```

* Command-line parameters are printed to stdout (useful for future replications). (Manuel López-Ibáñez, suggested by Markus Wagner)

* Users may provide a function to repair configurations before being evaluated. See the scenario variable repairConfiguration. (Manuel López-Ibáñez)

* The option `--sge-cluster` (`sgeCluster`) was removed and replaced by `--batchmode` (`batchmode`). It is now the responsibility of the target-runner to parse the output of the batch job submission command (e.g., `qsub` or `squeue`), and return just the job ID. Values supported are: "sge", "torque", "pbs" and "slurm". (Manuel López-Ibáñez)

* The option `--parallel` can now be combined with `--batchmode` to limit the number of jobs submitted by irace at once. This may be useful in batch clusters that have a small queue of jobs. (Manuel López-Ibáñez)

* New examples under `inst/examples/batchmode-cluster/`. (Manuel López-Ibáñez)

* It is now possible to include scenario definition files from other scenario files by using:

  ```R
  eval.parent(source("scenario-common.txt", chdir = TRUE, local = TRUE))
  ```

  This feature is VERY experimental and the syntax is likely to change in the future. (Manuel López-Ibáñez)

* Fix a bug that re-executed elite results under some circumstances. (Leslie Pérez Cáceres)

* Restrict the number of maximum configurations per race to 1024. (Leslie Pérez Cáceres)

* Do not warn if the last line in the instance file does not terminate with a newline. (Manuel López-Ibáñez)

* Fix bug when `deterministic == 1`. (Manuel López-Ibáñez, Leslie Pérez Cáceres)

* Update manual and vignette with details about the expected arguments and return value of `targetRunner` and `targetEvaluator`. (Manuel López-Ibáñez)

* Many updates to the User Guide vignette. (Manuel López-Ibáñez)

* Fix `\dontrun` example in `irace-package.Rd` (Manuel López-Ibáñez)

* Fix bug: If testInstances contains duplicates, results of testing are not correctly saved in `iraceResults$testing$experiments` nor reported correctly at the end of a run. Now unique IDs of the form `1t, 2t, ...` are used for each testing instance. These IDs are used for the rownames of `iraceResults$testing$experiments` and the names of the `scenario$testInstances` and `iraceResults$testing$seeds` vectors. (Manuel López-Ibáñez)

* Fix bug where irace keeps retrying the `target-runner` call even if it succeeds. (Manuel López-Ibáñez)

 * New command-line parameter
```
      --only-test FILE
```
   which just evaluates the configurations given in FILE on the testing
   instances defined by the scenario. Useful if you decide on the testing
   instances only after running irace.   (Manuel López-Ibáñez)

 * Bugfix: When using `maxTime != 0`, the number of experiments performed may be
   miscounted in some cases.   (Manuel López-Ibáñez)

# irace 2.1

 * Fix CRAN errors in tests. (Manuel López-Ibáñez)

 * Avoid generating too many configurations at once if the initial time
   estimation is too small. (Manuel López-Ibáñez)

# irace 2.0

 * Minimum R version is 2.15.

 * Elitist irace by default, it can be disabled with parameter `--elitist 0`.
   (Leslie Pérez Cáceres, Manuel López-Ibáñez)

 * The parameter `--test-type` gains two additional values: (Manuel López-Ibáñez)

   - `t-test-bonferroni` (t-test with Bonferroni's correction for multiple
                         comparisons),
   - `t-test-holm` (t-test with Holm's correction for multiple comparisons)

 * MPI does not create log files with `--debug-level 0`.
   (Manuel López-Ibáñez)

 * For simplicity, the `parallel-irace-*` scripts do not use an auxiliary
   `tune-main` script. For customizing them, make a copy and edit them
   directly.
   (Manuel López-Ibáñez)

 * New parameters: (Manuel López-Ibáñez)
```
--target-runner-retries : Retry target-runner this many times in case of error.
```

 * We print diversity measures after evaluating on each instance:
   (Leslie Pérez Cáceres)

   - Kendall's W (also known as Kendall's coefficient of concordance) If 1,
     all candidates have ranked in the same order in all instances. If 0, the
     ranking of each candidate on each instance is essentially random.

             W = Friedman / (m * (k-1))

   - Spearman's rho: average (Spearman) correlation coefficient computed on the
     ranks of all pairs of raters. If there are no repeated data values, a
     perfect Spearman correlation of +1 or -1 occurs when each of the variables
     is a perfect monotone function of the other.

 * Many internal and external interfaces have changed. For example, now we
   consistently use 'scenario' to denote the settings passed to irace and

~~~
   'configuration' instead of 'candidate' to denote the parameter settings
   passed to the target algorithm. Other changes are:
```R
   parameters$boundary -> parameters$domain
   hookRun             -> targetRunner
   hookEvaluate        -> targetEvaluator
   tune-conf           -> scenario.txt
   instanceDir         -> trainInstancesDir
   instanceFile        -> trainInstancesFile
   testInstanceDir     -> testInstancesDir
   testInstanceFile    -> testInstancesFile
```

 * Minimal example of configuring a MATLAB program
   (thanks to Esteban Diaz Leiva)

 * Paths to files or directories given in the scenario file are relative to the
   scenario file (except for `--log-file`, which is an output file and it is
   relative to `--exec-dir`). Paths given in the command-line are relative to the
   current working directory. Given
```bash
      $ cat scenario/scenario.txt
      targetRunner <- "./target-runner"
      $ irace -s scenario/scenario.txt
```
   irace will search for `"./scenario/target-runner"`, but given
```bash
      $ irace -s scenario/scenario.txt --target-runner ./target-runner
```
   irace will search for `"./target-runner"`.      (Manuel López-Ibáñez)

 * New command-line wrapper for Windows installed at
   `system.file("bin/irace.bat", package="irace")`
   (thanks to Anthony Antoun)

 * Budget can be specified as maximum time (`maxTime`, `--max-time`) consumed by
   the target algorithm. See the documentation for the details about how this
   is handled.
   (Leslie Pérez Cáceres, Manuel López-Ibáñez)


# irace 1.07

 * The best configurations found, either at the end or at each iteration of an
   irace run, can now be applied to a set of test instances different from the
   training instances. See options `testInstanceDir`, `testInstanceFile`,
   `testNbElites`, and `testIterationElites`. (Leslie Pérez Cáceres, Manuel López-Ibáñez)

 * The R interfaces of `hookRun`, `hookEvaluate` and `hookRunParallel` have changed.
   See `help(hook.run.default)` and `help(hook.evaluate.default)` for examples of
   the new interfaces.

 * Printing of race progress now reports the actual configuration and instance
   IDs, and numbers are printed in a more human-readable format.
   (Leslie Pérez Cáceres, Manuel López-Ibáñez)

 * Reduce memory use for very large values of `maxExperiments`.
   (Manuel López-Ibáñez, thanks to Federico Caselli for identifying the issue)

 * New option `--load-balancing` (`loadBalancing`) for disabling load-balancing
   when executing jobs 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.
   (Manuel López-Ibáñez, thanks to Federico Caselli for identifying the issue)

 * The option `--parallel` in Windows now uses load-balancing by default.
   (Manuel López-Ibáñez)

 * The wall-clock time after finishing each task is printed in the output.
   (Manuel López-Ibáñez, thanks to Federico Caselli for providing an initial
   patch)


# irace 1.06

 * Fix bug that could introduce spurious whitespace when printing the
   final configurations. (Manuel López-Ibáñez)

 * Fix bug if there are more initial candidates than needed for the
   first race. (Leslie Pérez Cáceres, Manuel López-Ibáñez)

 * New configuration options, mainly for R users:

    - `hookRunParallel`: Optional R function to provide custom
      parallelization of `hook.run`.

    - `hookRunData`: Optional data passed to `hookRun`. This is ignored by the
      default `hookRun` function, but it may be used by custom `hookRun` R
      functions to pass persistent data around.  (Manuel López-Ibáñez)

# irace 1.05

 * New option `--version`. (Manuel López-Ibáñez)

 * Terminate early if there is no sufficient budget to run irace with
   the given settings. (Manuel López-Ibáñez)

 * The option `--parallel` (without `--mpi`) now works under Windows.
   (Manuel López-Ibáñez, thanks to Pablo Valledor Pellicer for testing
   it)

 * Improved error handling when running under Rmpi. Now irace will
   terminate as soon as the master node detects at least one failed
   slave node. This avoids irace reporting two times the same error.
   Also, irace will print all the unique errors returned by all slaves
   and not just the first one.
   (Manuel López-Ibáñez)

 * Forbidden configurations may be specified in terms of constraints
   on their values. Forbidden configurations will never be evaluated by irace.
   See `--forbidden-file` and `inst/templates/forbidden.tmpl`.
   (Manuel López-Ibáñez)

 * New option `--recovery-file` (`recoveryFile`) allows resuming a
   previous irace run. (Leslie Pérez Cáceres)

 * The confidence level for the elimination test is now
   configurable with parameter `--confidence`. (Leslie Pérez Cáceres)

 * Much more robust handling of relative/absolute paths. Improved support
   for Windows. (Leslie Pérez Cáceres, Manuel López-Ibáñez)

 * Provide better error messages for incorrect parameter
   descriptions. (Manuel López-Ibáñez)
```

~~~~
    Examples:
```
    x "" i (0, 0)       # lower and upper bounds are the same
    x "" r (1e-4, 5e-4) # given digits=2, ditto
    x "" i (-1, -2)     # lower bound must be smaller than upper bound
    x "" c ("a", "a")   # duplicated values
```
 * Print elapsed time for calls to hook-run if `debugLevel >=1`.
   (Manuel López-Ibáñez)

 * `examples/hook-run-python/hook-run`: A multi-purpose `hook-run` written
   in Python. (Franco Mascia)

 * Parallel mode in an SGE cluster (`--sge-cluster`) is more
   robust. (Manuel López-Ibáñez)

# irace 1.04

 * Replace obsolete package multicore by package parallel
   (requires R >= 2.14.0)

 * Use load-balancing (`mc.preschedule = FALSE`) in `mclapply`.

# irace 1.03

 * Use `reg.finalizer` to finish Rmpi properly without clobbering
   `.Last()`.

 * Remove uses of deprecated `as.real()`.

 * Nicer error handling in `readParameters()`.

 * Add hypervolume (multi-objective) example.

 * Fix several bugs in the computation of similar candidates.

# irace 1.02

 * More concise output.

 * The parameters `expName` and `expDescription` are now useless and they
   were removed.

 * Faster computation of similar candidates (Jeremie Dubois-Lacoste
   and Leslie Pérez Cáceres).

 * Fix bug when saving instances in `tunerResults$experiments`.

 * `irace.cmdline ("--help")` does not try to quit R anymore.

# irace 1.01

 * Fix bug caused by file.exists (and possibly other functions)
   not handling directory names with a trailing backslash or slash on
   Windows.

 * Fix bug using per-instance parameters (Leslie Pérez Cáceres).

 * Fix bug when reading initial candidates from a file.
~~~~