Package 'BatchExperiments'

March 21, 2022

Title Statistical Experiments on Batch Computing Clusters

Description Extends the BatchJobs package to run statistical experiments on batch computing clusters. For further details see the project web page.

Author Bernd Bischl <bernd_bischl@gmx.net>,
Michel Lang <michellang@gmail.com>,
Olaf Mersmann <olafm@p-value.net>

Maintainer Michel Lang <michellang@gmail.com>

URL https://github.com/tudo-r/BatchExperiments

BugReports https://github.com/tudo-r/BatchExperiments/issues

License BSD_3_clause + file LICENSE

Depends R (>= 3.0.0), BatchJobs (>= 1.7)

Imports backports, utils, stats, checkmate (>= 1.8.5), BBmisc (>= 1.11), DBI, RSQLite (>= 2.0), data.table (>= 1.9.6)

Suggests plyr, randomForest, rpart, testthat

Version 1.4.3

RoxygenNote 7.1.2

NeedsCompilation no

Repository CRAN

Date/Publication 2022-03-21 11:00:02 UTC

R topics documented:

`addAlgorithm` *Add an algorithm to registry.*

Description

Add an algorithm to registry and stores it on disk.

Usage

```
addAlgorithm(reg, id, fun, overwrite = FALSE)
```

Arguments

`reg`	[ExperimentRegistry] Registry.
`id`	[`character(1)`] Name of algorithm.
`fun`	[`function(job,static,dynamic,...)`] Function which applies the algorithm to a problem instance. Takes a Job object, the `static` problem part and the evaluated `dynamic` problem part as arguments. You may omit any of `job`, `static` or `dynamic`. In this case, the respective arguments will not get passed to `fun`. Further parameters from Design are passed to ... argument. If you are using multiple result files this function must return a named list. To retrieve job informations from the `job` object see the documentation on ExperimentJob.
`overwrite`	[`logical(1)`] Overwrite the algorithm file if it already exists? Default is `FALSE`.

Value

character(1) . Invisibly returns the id.

See Also

Other add: addExperiments(), addProblem()

---

addExperiments *Add experiemts to the registry.*

Description

Add experiments for running algorithms on problems to the registry, so they can be executed later.

Usage

```
addExperiments(
  reg,
  prob.designs,
  algo.designs,
  repls = 1L,
  skip.defined = FALSE
)
```

Arguments

reg	[ExperimentRegistry] Registry.
prob.designs	[character | Design | list of Design] Either problem ids, a single problem design or a list of problem designs, the latter two created by makeDesign. If missing, all problems are selected (without associating a design), and this is the default.
algo.designs	[character | Design | list of Design] Either algorithm ids, a single algorithm design or a list of algorithm designs, the latter two created by makeDesign. If missing, all algorithms are selected (without associating a design), and this is the default.
repls	[integer(1)] Number of replications. Default is 1.
skip.defined	[logical] If set to TRUE, already defined experiments get skipped. Otherwise an error is thrown. Default is FALSE.

Value

Invisibly returns vector of ids of added experiments.

See Also

Other add: addAlgorithm(), addProblem()

Examples

```
### EXAMPLE 1 ###
reg = makeExperimentRegistry(id = "example1", file.dir = tempfile())

# Define a problem:
# Subsampling from the iris dataset.
data(iris)
subsample = function(static, ratio) {
  n = nrow(static)
  train = sample(n, floor(n * ratio))
  test = setdiff(seq(n), train)
  list(test = test, train = train)
}
addProblem(reg, id = "iris", static = iris,
           dynamic = subsample, seed = 123)

# Define algorithm "tree":
# Decision tree on the iris dataset, modeling Species.
tree.wrapper = function(static, dynamic, ...) {
  library(rpart)
  mod = rpart(Species ~ ., data = static[dynamic$train, ], ...)
  pred = predict(mod, newdata = static[dynamic$test, ], type = "class")
  table(static$Species[dynamic$test], pred)
}
addAlgorithm(reg, id = "tree", fun = tree.wrapper)

# Define algorithm "forest":
# Random forest on the iris dataset, modeling Species.
forest.wrapper = function(static, dynamic, ...) {
  library(randomForest)
  mod = randomForest(Species ~ ., data = static, subset = dynamic$train, ...)
  pred = predict(mod, newdata = static[dynamic$test, ])
  table(static$Species[dynamic$test], pred)
}
addAlgorithm(reg, id = "forest", fun = forest.wrapper)

# Define problem parameters:
pars = list(ratio = c(0.67, 0.9))
iris.design = makeDesign("iris", exhaustive = pars)

# Define decision tree parameters:
pars = list(minsplit = c(10, 20), cp = c(0.01, 0.1))
tree.design = makeDesign("tree", exhaustive = pars)

# Define random forest parameters:
pars = list(ntree = c(100, 500))
forest.design = makeDesign("forest", exhaustive = pars)
```

```
# Add experiments to the registry:
# Use  previously defined experimental designs.
addExperiments(reg, prob.designs = iris.design,
               algo.designs = list(tree.design, forest.design),
               repls = 2) # usually you would set repls to 100 or more.

# Optional: Short summary over problems and algorithms.
summarizeExperiments(reg)

# Optional: Test one decision tree job and one expensive (ntree = 1000)
# random forest job. Use findExperiments to get the right job ids.
do.tests = FALSE
if (do.tests) {
  id1 = findExperiments(reg, algo.pattern = "tree")[1]
  id2 = findExperiments(reg, algo.pattern = "forest",
                        algo.pars = (ntree == 1000))[1]
  testJob(reg, id1)
  testJob(reg, id2)
}

# Submit the jobs to the batch system
submitJobs(reg)

# Calculate the misclassification rate for all (already done) jobs.
reduce = function(job, res) {
  n = sum(res)
  list(mcr = (n-sum(diag(res)))/n)
}
res = reduceResultsExperiments(reg, fun = reduce)
print(res)

# Aggregate results using 'ddply' from package 'plyr':
# Calculate the mean over all replications of identical experiments
# (same problem, same algorithm and same parameters)
library(plyr)
vars = setdiff(names(res), c("repl", "mcr"))
aggr = ddply(res, vars, summarise, mean.mcr = mean(mcr))
print(aggr)

## Not run:
### EXAMPLE 2 ###
# define two simple test functions
testfun1 = function(x) sum(x^2)
testfun2 = function(x) -exp(-sum(abs(x)))

# Define ExperimentRegistry:
reg = makeExperimentRegistry("example02", seed = 123, file.dir = tempfile())

# Add the testfunctions to the registry:
addProblem(reg, "testfun1", static = testfun1)
addProblem(reg, "testfun2", static = testfun2)

# Use SimulatedAnnealing on the test functions:
```

```
addAlgorithm(reg, "sann", fun = function(static, dynamic) {
  upp = rep(10, 2)
  low = -upp
  start = sample(c(-10, 10), 2)
  res = optim(start, fn = static, lower = low, upper = upp, method = "SANN")
  res = res[c("par", "value", "counts", "convergence")]
  res$start = start
  return(res)
})

# add experiments and submit
addExperiments(reg, repls = 10)
submitJobs(reg)

# Gather informations from the experiments, in this case function value
# and whether the algorithm convergenced:
reduceResultsExperiments(reg, fun = function(job, res) res[c("value", "convergence")])

## End(Not run)
```

`addProblem` *Add a problem to registry.*

Description

Add a algorithm to problem and stores it on disk.

Usage

```
addProblem(
  reg,
  id,
  static = NULL,
  dynamic = NULL,
  seed = NULL,
  overwrite = FALSE
)
```

Arguments

`reg`	[ExperimentRegistry] Registry.
`id`	[character(1)] Name of problem.
`static`	[any] Static part of problem that never changes and is not dependent on parameters. Default is `NULL`.

dynamic [function(job,static,...)]
R generator function that creates dynamic / stochastic part of problem instance, which might be dependent on parameters. First parameter job is a Job object, second is static problem part static. Further parameters from design are passed to ... argument on instance creation time. The arguments job and static may be omitted. To retrieve job informations from the job object see the documentation on ExperimentJob. Default is NULL.

seed [integer(1)]
Start seed for this problem. This allows the "synchronization" of a stochastic problem across algorithms, so that different algorithms are evaluated on the same stochastic instance. The seeding mechanism works as follows, if a problem seed is defined: (1) Before the dynamic part of a problem is instantiated, the seed of the problem + replication - 1 is set, so for the first replication the exact problem seed is used. (2) The stochastic part of the problem is instantiated (3) From now on the usual experiment seed of the registry is used, see ExperimentRegistry. If seed is set to NULL this extra problem seeding is switched off, meaning different algorithms see different stochastic versions of the same problem. Default is NULL.

overwrite [logical(1)]
Overwrite the problem file if it already exists? Default is FALSE.

Value

character(1) . Invisibly returns the id.

See Also

Other add: addAlgorithm(), addExperiments()

BatchExperiments *The BatchExperiments package*

Description

Extends the BatchJobs package to run statistical experiments on batch computing clusters.

Additional information

Homepage: https://github.com/tudo-r/BatchExperiments

Wiki: https://github.com/tudo-r/BatchExperiments/wiki

ExperimentJob *ExperimentJob*

Description

You can access job properties using the `job` object which is optionally passed to dynamic problem functions and algorithms. The object is a named list with the following elements:

`id` [`integer(1)` :] Job ID.

`prob.id` [`character(1)` :] Problem ID.

`prob.pars` [`list` :] Problem parameters as named list.

`algo.id` [`character(1)` :] algo.idAlgorithm ID.

`algo.pars` [`list` :] Algorithm parameters as named list.

`repl` [`integer(1)` :] Replication number of this experiment.

`seed` [`integer(1)` :] Seed set right before algorithm execution.

`prob.seed` [`integer(1)` :] Seed set right before generation of problem instance.

findExperiments *Find ids of experiments that match a query.*

Description

Find job ids by querying problem/algorithm ids, problem/algorithm parameters or replication number.

Usage

```
findExperiments(
  reg,
  ids,
  prob.pattern,
  prob.pars,
  algo.pattern,
  algo.pars,
  repls,
  match.substring = TRUE,
  regexp = FALSE
)
```

Arguments

reg	[ExperimentRegistry] Registry.
ids	[integer] Ids of selected experiments to restrict to. Default is all experiments.
prob.pattern	[character(1)] If not missing, all problem ids that match this string are selected.
prob.pars	[R expression] If not missing, all problems whose parameters match the given expression are selected.
algo.pattern	[character(1)] If not missing, all algorithm ids that match this string are selected.
algo.pars	[R expression] If not missing, all algorithms whose parameters match the given expression are selected.
repls	[integer] If not missing, restrict to jobs with given replication numbers.
match.substring	[logical(1)] Is a match in prob.pattern and algo.pattern if the id contains the pattern as substring or must the id exactly match? Default is TRUE.
regexp	[logical(1)] Are prob.pattern and algo.pattern regular expressions? Note that this is significantly slower than substring matching. If set to TRUE the argument match.substring has no effect. Default is FALSE.

Value

integer . Ids for experiments which match the query.

Examples

```
reg = makeExperimentRegistry(id = "example1", file.dir = tempfile())
p1 = addProblem(reg, "one", 1)
p2 = addProblem(reg, "two", 2)
a = addAlgorithm(reg, "A", fun = function(static, n) static + n)
addExperiments(reg, algo.design = makeDesign(a, exhaustive = list(n = 1:4)))
findExperiments(reg, prob.pattern = "one")
findExperiments(reg, prob.pattern = "o")
findExperiments(reg, algo.pars = (n > 2))
```

`generateProblemInstance` *Generate dynamic part of problem.*

Description

Calls the dynamic problem function on the static problem part and thereby creates the problem instance. The seeding mechanism is identical to execution on the slave.

Usage

```
generateProblemInstance(reg, id)
```

Arguments

`reg`	[ExperimentRegistry] Registry.
`id`	[character(1)] Id of job.

Value

Dynamic part of problem.

`getAlgorithm` *Get algorithm from registry by id.*

Description

The requested object is loaded from disk.

Usage

```
getAlgorithm(reg, id)
```

Arguments

`reg`	[ExperimentRegistry] Registry.
`id`	[character(1)] Id of algorithm.

Value

Algorithm .

See Also

Other get: getAlgorithmIds(), getExperimentParts(), getJobs.ExperimentRegistry(), getProblemIds(), getProblem()

`getAlgorithmIds` *Get ids of algorithms in registry.*

Description

Get algorithm ids for jobs.

Usage

```
getAlgorithmIds(reg, ids)
```

Arguments

reg	[ExperimentRegistry] Registry.
ids	[codeinteger] Job ids to restrict returned algorithm ids to.

Value

character .

See Also

Other get: getAlgorithm(), getExperimentParts(), getJobs.ExperimentRegistry(), getProblemIds(), getProblem()

`getExperimentParts` *Get all parts required to run a single job.*

Description

Get all parts which define an Experiment.

Usage

```
getExperimentParts(reg, id)
```

Arguments

reg	[ExperimentRegistry] Registry.
id	[integer(1)] Id of a job.

Value

named list . Returns the Job, Problem, Instance and Algorithm.

See Also

Other get: getAlgorithmIds(), getAlgorithm(), getJobs.ExperimentRegistry(), getProblemIds(), getProblem()

---

getIndex *Group experiments.*

Description

Creates a list of factor to use in functions like tapply, by or aggregate.

Usage

```
getIndex(
  reg,
  ids,
  by.prob = FALSE,
  by.algo = FALSE,
  by.repl = FALSE,
  by.prob.pars,
  by.algo.pars,
  enclos = parent.frame()
)
```

Arguments

reg	[ExperimentRegistry] Registry.
ids	[integer] If not missing, restict grouping to this subset of experiment ids.
by.prob	[logical] Group experiments by problem. Default is FALSE.
by.algo	[logical] Group experiments by algorithm. Default is FALSE.

by.repl	[logical] Group experiments by replication. Default is FALSE.
by.prob.pars	[R expression] If not missing, group experiments by this R expression. The expression is evaluated in the environment of problem parameters and converted to a factor using as.factor.
by.algo.pars	[R expression] If not missing, group experiments by this R expression. The expression is evaluated in the environment of algorithm parameters and converted to a factor using as.factor.
enclos	[environment] Enclosing frame for evaluation of parameters used by by.prob.pars and by.algo.pars, see eval. Defaults to the parent frame.

Value

list . List of factors.

Examples

```
# create a registry and add problems and algorithms
reg = makeExperimentRegistry("getIndex", file.dir = tempfile(""))
addProblem(reg, "prob", static = 1)
addAlgorithm(reg, "f0", function(static, dynamic) static)
addAlgorithm(reg, "f1", function(static, dynamic, i, k) static * i^k)
ad = list(makeDesign("f0"), makeDesign("f1", exhaustive = list(i = 1:5, k = 1:3)))
addExperiments(reg, algo.designs = ad)
submitJobs(reg)

# get grouped job ids
ids = getJobIds(reg)
by(ids, getIndex(reg, by.prob = TRUE, by.algo = TRUE), identity)
ids = findExperiments(reg, algo.pattern = "f1")
by(ids, getIndex(reg, ids, by.algo.pars = (k == 1)), identity)

# groupwise reduction
ids = findExperiments(reg, algo.pattern = "f1")
showStatus(reg, ids)
f = function(aggr, job, res) aggr + res
by(ids, getIndex(reg, ids, by.algo.pars = k), reduceResults, reg = reg, fun = f)
by(ids, getIndex(reg, ids, by.algo.pars = i), reduceResults, reg = reg, fun = f)
```

---

getJobs.ExperimentRegistry

Get jobs (here: experiments) from registry by id.

Description

Constructs an Experiment for each job id provided.

Usage

```
## S3 method for class 'ExperimentRegistry'
getJobs(reg, ids, check.ids = TRUE)
```

Arguments

reg	[ExperimentRegistry] Registry.
ids	[integer] Ids of job. Default is all jobs.
check.ids	[logical(1)] Check the job ids? Default is TRUE.

Value

list of Experiment .

See Also

Other get: getAlgorithmIds(), getAlgorithm(), getExperimentParts(), getProblemIds(), getProblem()

---

getProblem *Get problem from registry by id.*

Description

The requested object is loaded from disk.

Usage

```
getProblem(reg, id)
```

Arguments

reg	[ExperimentRegistry] Registry.
id	[character(1)] Id of problem.

Value

Problem .

See Also

Other get: getAlgorithmIds(), getAlgorithm(), getExperimentParts(), getJobs.ExperimentRegistry(), getProblemIds()

getProblemIds *Get ids of problems in registry.*

Description

Get problem ids for jobs.

Usage

```
getProblemIds(reg, ids)
```

Arguments

reg	[ExperimentRegistry] Registry.
ids	[codeinteger] Job ids to restrict returned problem ids to.

Value

character .

See Also

Other get: getAlgorithmIds(), getAlgorithm(), getExperimentParts(), getJobs.ExperimentRegistry(), getProblem()

getResultVars *Get variable groups of reduced results.*

Description

Useful helper for e.g. package plyr and such.

Usage

```
getResultVars(data, type = "group")
```

Arguments

data	[ReducedResultsExperiments] Result data.frame from reduceResultsExperiments.
type	[character(1)] Can be "prob" (prob + pars), "prob.pars" (only problem pars), "algo" (algo + pars), "algo.pars" (only algo pars), "group" (prob + problem pars + algo + algo pars), "result" (result column names). Default is "group".

Value

character . Names of of columns.

Examples

```
reg = makeExperimentRegistry("BatchExample", seed = 123, file.dir = tempfile())
addProblem(reg, "p1", static = 1)
addProblem(reg, "p2", static = 2)
addAlgorithm(reg, id = "a1",
  fun = function(static, dynamic, alpha) c(y = static*alpha))
addAlgorithm(reg, id = "a2",
  fun = function(static, dynamic, alpha, beta) c(y = static*alpha+beta))
ad1 = makeDesign("a1", exhaustive = list(alpha = 1:2))
ad2 = makeDesign("a2", exhaustive = list(alpha = 1:2, beta = 5:6))
addExperiments(reg, algo.designs = list(ad1, ad2), repls = 2)
submitJobs(reg)
data = reduceResultsExperiments(reg)
library(plyr)
ddply(data, getResultVars(data, "group"), summarise, mean_y = mean(y))
```

---

makeDesign *Create parameter designs for problems and algorithms.*

Description

Create a parameter design for either a problem or an algorithm that you can use in addExperiments. All parameters in design and exhaustive be "primitive" in the sense that either is.atomic is TRUE or is.factor is TRUE.

Be aware of R's default behaviour of converting strings into factors if you use the design parameter. See option stringsAsFactors in data.frame to turn this off.

Usage

```
makeDesign(id, design = data.frame(), exhaustive = list())
```

Arguments

id	[character(1)] Id of algorithm or problem.
design	[data.frame] The design. Must have named columns corresponding to parameters. Default is an empty data.frame().
exhaustive	[list] Named list of parameters settings which should be exhaustively tried. Names must correspond to parameters. Default is empty list.

Value

Design .

Examples

```
## Not run:
# simple design for algorithm "a1" with no parameters:
design = makeDesign("a1")

# design for problem "p1" using predefined parameter combinations
design = makeDesign("p1", design = data.frame(alpha = 0:1, beta = c(0.1, 0.2)))

# creating a list of designs for several algorithms at once, all using the same
# exhaustive grid of parameters
designs = lapply(c("a1", "a2", "a3"), makeDesign,
                 exhaustive = list(alpha = 0:1, gamma = 1:10/10))

## End(Not run)
```

---

makeExperimentRegistry

Construct a registry object for experiments.

Description

Note that if you don't want links in your paths (`file.dir`, `work.dir`) to get resolved and have complete control over the way the path is used internally, pass an absolute path which begins with "/".

Every object is a list that contains the passed arguments of the constructor.

Usage

```
makeExperimentRegistry(
  id = "BatchExperimentRegistry",
  file.dir,
  sharding = TRUE,
  work.dir,
  multiple.result.files = FALSE,
  seed,
  packages = character(0L),
  src.dirs = character(0L),
  src.files = character(0L),
  skip = TRUE
)
```

Arguments

id	[character(1)] Name of registry. Displayed e.g. in mails or in cluster queue. Default is "BatchExperimentRegistry".
file.dir	[character(1)] Path where files regarding the registry / jobs should be saved. Default is dQuote<name of registry>_files in current working directory.
sharding	[logical(1)] Enable sharding to distribute result files into different subdirectories? Important if you have many experiments. Default is TRUE.
work.dir	[character(1)] Working directory for R process when experiment is executed. Default is the current working directory when registry is created.
multiple.result.files	[logical(1)] Should a result file be generated for every list element of the returned list of the algorithm function? Note that your algorithm functions in addAlgorithm must return named lists if this is set to TRUE. The result file will be named "<id>-result-<element name>.RData" instead of "<id>-result.RData". Default is FALSE.
seed	[integer(1)] Start seed for experiments. The first experiment in the registry will use this seed, for the subsequent ones the seed is incremented by 1. Default is a random number from 1 to .Machine$integer.max/2.
packages	[character] Packages that will always be loaded on each node. Default is character(0).
src.dirs	[character] Directories relative to your work.dir containing R scripts to be sourced on registry load (both on slave and master). Files not matching the pattern "\.[Rr]$" are ignored. Useful if you have many helper functions that are needed during the execution of your jobs. These files should only contain function definitions and no executable code. Default is character(0).
src.files	[character] R scripts files relative to your work.dir to be sourced on registry load (both on slave and master). Useful if you have many helper functions that are needed during the execution of your jobs. These files should only contain function definitions and no executable code. Default is character(0).
skip	[logical(1)] Skip creation of a new registry if a registry is found in file.dir. Defaults to TRUE.

Value

ExperimentRegistry

`reduceResultsExperiments` *Reduce results into a data.frame with all relevant information.*

Description

Generates a `data.frame` with one row per job id. The columns are: ids of problem and algorithm (named "prob" and "algo"), one column per parameter of problem or algorithm (named by the parameter name), the replication number (named "repl") and all columns defined in the function to collect the values. Note that you cannot rely on the order of the columns. If a parameter does not have a setting for a certain job / experiment it is set to `NA`. Have a look at `getResultVars` if you want to use something like `ddply` on the results.

The rows are ordered as `ids` and named with `ids`, so one can easily index them.

Usage

```
reduceResultsExperiments(
  reg,
  ids,
  part = NA_character_,
  fun,
  ...,
  strings.as.factors = FALSE,
  block.size,
  impute.val,
  apply.on.missing = FALSE,
  progressbar = TRUE
)
```

Arguments

`reg`	[`ExperimentRegistry`] Registry.
`ids`	[`integer`] Ids of selected experiments. Default is all jobs for which results are available.
`part`	[`character`] Only useful for multiple result files, then defines which result file part(s) should be loaded. `NA` means all parts are loaded, which is the default.
`fun`	[`function(job,res,...)`] Function to collect values from `job` and result `res` object, the latter from stored result file. Must return a named object which can be coerced to a `data.frame` (e.g. a `list`). Default is a function that simply returns `res` which may or may not work, depending on the type of `res`. We recommend to always return a named list.
`...`	[any] Additional arguments to `fun`.

strings.as.factors
: [logical(1)] Should all character columns in result be converted to factors? Default is FALSE.

block.size
: [integer(1)] Results will be fetched in blocks of this size. Default is max(100, 5 percent of ids).

impute.val
: [named list]
If not missing, the value of impute.val is used as a replacement for the return value of function fun on missing results. An empty list is allowed.

apply.on.missing
: [logical(1)]
Apply the function on jobs with missing results? The argument "res" will be NULL and must be handled in the function. This argument has no effect if impute.val is set. Default ist FALSE.

progressbar
: [logical(1)]
Set to FALSE to disable the progress bar. To disable all progress bars, see makeProgressBar.

Value

data.frame . Aggregated results, containing problem and algorithm paramaters and collected values.

---

reduceResultsExperimentsParallel

Reduce very many results in parallel.

Description

Basically the same as reduceResultsExperiments but creates a few (hopefully short) jobs to reduce the results in parallel. The function internally calls batchMapQuick, does "busy-waiting" till all jobs are done and cleans all temporary files up.

The rows are ordered as ids and named with ids, so one can easily index them.

Usage

```
reduceResultsExperimentsParallel(
  reg,
  ids,
  part = NA_character_,
  fun,
  ...,
  timeout = 604800L,
  njobs = 20L,
  strings.as.factors = FALSE,
  impute.val,
  apply.on.missing = FALSE,
  progressbar = TRUE
)
```

Arguments

reg	[ExperimentRegistry] Registry.
ids	[integer] Ids of selected experiments. Default is all jobs for which results are available.
part	[character] Only useful for multiple result files, then defines which result file part(s) should be loaded. NA means all parts are loaded, which is the default.
fun	[function(job,res,...)] Function to collect values from job and result res object, the latter from stored result file. Must return a named object which can be coerced to a data.frame (e.g. a list). Default is a function that simply returns res which may or may not work, depending on the type of res. We recommend to always return a named list.
...	[any] Additional arguments to fun.
timeout	[integer(1)] Seconds to wait for completion. Passed to waitForJobs. Default is 648400 (one week).
njobs	[integer(1)] Number of parallel jobs to create. Default is 20.
strings.as.factors	[logical(1)] Should all character columns in result be converted to factors? Default is FALSE.
impute.val	[named list] If not missing, the value of impute.val is used as a replacement for the return value of function fun on missing results. An empty list is allowed.
apply.on.missing	[logical(1)] Apply the function on jobs with missing results? The argument "res" will be NULL and must be handled in the function. This argument has no effect if impute.val is set. Default ist FALSE.
progressbar	[logical(1)] Set to FALSE to disable the progress bar. To disable all progress bars, see makeProgressBar.

Value

data.frame . Aggregated results, containing problem and algorithm paramaters and collected values.

---

removeAlgorithm *Remove algorithm from registry.*

Description

THIS DELETES ALL FILES REGARDING THIS ALGORITHM, INCLUDING ALL JOBS AND RESULTS!

Usage

```
removeAlgorithm(reg, id, force = FALSE)
```

Arguments

reg	[ExperimentRegistry] Registry.
id	[character(1)] Id of algorithm.
force	[logical(1)] Also remove jobs which seem to be still running. Default is FALSE.

Value

Nothing.

See Also

Other remove: removeExperiments(), removeProblem()

---

removeExperiments *Remove jobs from registry.*

Description

THIS DELETES ALL FILES REGARDING THE JOBS, INCLUDING RESULTS! If you really know what you are doing, you may set force to TRUE to omit sanity checks on running jobs.

Usage

```
removeExperiments(reg, ids, force = FALSE)
```

Arguments

reg	[ExperimentRegistry] Registry.
ids	[integer] Ids of jobs you want to remove. Default is none.
force	[logical(1)] Also remove jobs which seem to be still running. Default is FALSE.

Value

Vector of type integer of removed job ids.

See Also

Other remove: removeAlgorithm(), removeProblem()

removeProblem *Remove problem from registry.*

Description

THIS DELETES ALL FILES REGARDING THIS PROBLEM, INCLUDING ALL JOBS AND RESULTS!

Usage

```
removeProblem(reg, id, force = FALSE)
```

Arguments

reg	[ExperimentRegistry] Registry.
id	[character(1)] Id of problem.
force	[logical(1)] Also remove jobs which seem to be still running. Default is FALSE.

Value

Nothing.

See Also

Other remove: removeAlgorithm(), removeExperiments()

summarizeExperiments *Summarize selected experiments.*

Description

A data.frame is returned that contains summary information about the selected experiments. The data.frame is constructed by building the columns "prob, <prob.pars>, algo, <algo.pars>, repl". Now only the columns in show will be selected, how many of such experiments exist will be counted in a new column ".count".

Usage

```
summarizeExperiments(reg, ids, show = c("prob", "algo"))
```

Arguments

reg	[ExperimentRegistry] Registry.
ids	[integer] Selected experiments. Default is all experiments.
show	[character] Should detailed information for each single experiment be printed? Default is c("prob","algo").

Value

data.frame .

Examples

```
reg = makeExperimentRegistry("summarizeExperiments", seed = 123, file.dir = tempfile())
p1 = addProblem(reg, "p1", static = 1)
a1 = addAlgorithm(reg, id = "a1", fun = function(static, dynamic, alpha, beta) 1)
a2 = addAlgorithm(reg, id = "a2", fun = function(static, dynamic, alpha, gamma) 2)
ad1 = makeDesign(a1, exhaustive = list(alpha = 1:2, beta = 1:2))
ad2 = makeDesign(a2, exhaustive = list(alpha = 1:2, gamma = 7:8))
addExperiments(reg, algo.designs = list(ad1, ad2), repls = 2)
print(summarizeExperiments(reg))
print(summarizeExperiments(reg, show = c("prob", "algo", "alpha", "gamma")))
```

Index