

# Package ‘mlrMBO’

May 13, 2017

**Title** A Toolbox for Model-Based Optimization of Expensive Black-Box Functions

**Description** Flexible and comprehensive R toolbox for model-based optimization ('MBO'), also known as Bayesian optimization. It is designed for both single- and multi-objective optimization with mixed continuous, categorical and conditional parameters. The machine learning toolbox 'mlr' provide dozens of regression learners to model the performance of the target algorithm with respect to the parameter settings. It provides many different infill criteria to guide the search process. Additional features include multipoint batch proposal, parallel execution as well as visualization and sophisticated logging mechanisms, which is especially useful for teaching and understanding of algorithm behavior. 'mlrMBO' is implemented in a modular fashion, such that single components can be easily replaced or adapted by the user for specific use cases.

**License** BSD\_2\_clause + file LICENSE

**URL** <https://github.com/mlr-org/mlrMBO>

**BugReports** <https://github.com/mlr-org/mlrMBO/issues>

**Depends** mlr (>= 2.10), ParamHelpers (>= 1.10), smoof (>= 1.4)

**Imports** backports, BBmisc (>= 1.11), checkmate (>= 1.8.2), data.table, lhs, parallelMap (>= 1.3)

**Suggests** akima, cmaesr (>= 1.0.3), ggplot2, RColorBrewer, DiceKriging, DiceOptim, earth, emoa, GGally, gridExtra, kernlab, kkn, knitr, mco, nnet, party, randomForest, rmarkdown, rpart, testthat, eaf, covr

**LazyData** yes

**Encoding** UTF-8

**ByteCompile** yes

**Version** 1.1.0

**RoxygenNote** 6.0.1

**VignetteBuilder** knitr

**NeedsCompilation** yes

**Author** Bernd Bischl [aut],  
 Jakob Bossek [aut],  
 Jakob Richter [aut, cre],  
 Daniel Horn [aut],  
 Michel Lang [aut],  
 Janek Thomas [aut]

**Maintainer** Jakob Richter <code@jakob-r.de>

**Repository** CRAN

**Date/Publication** 2017-05-12 22:53:13 UTC

## R topics documented:

error_handling . . . . .	3
exampleRun . . . . .	4
exampleRunMultiObj . . . . .	5
getGlobalOpt . . . . .	7
getMBOInfillCrit . . . . .	7
getSupportedInfillOptFunctions . . . . .	8
getSupportedMultipointInfillOptFunctions . . . . .	8
infillcrits . . . . .	9
initCrit . . . . .	10
makeMBOControl . . . . .	11
makeMBOInfillCrit . . . . .	13
makeMBOLearner . . . . .	14
makeMBOTrafoFunction . . . . .	16
mbo . . . . .	16
mboContinue . . . . .	18
mboFinalize . . . . .	18
MBOMultiObjResult . . . . .	19
MBOSingleObjResult . . . . .	19
mbo_OptPath . . . . .	20
mbo_parallel . . . . .	21
mlrMBO_examples . . . . .	21
OptProblem . . . . .	28
OptResult . . . . .	28
OptState . . . . .	28
plotExampleRun . . . . .	29
plotMBOResult . . . . .	30
print.MBOControl . . . . .	31
renderExampleRunPlot . . . . .	31
setMBOControlInfill . . . . .	33
setMBOControlMultiObj . . . . .	36
setMBOControlMultiPoint . . . . .	38
setMBOControlTermination . . . . .	39
trafos . . . . .	41

<b>Index</b>	<b>43</b>
--------------	-----------

**Description**

There are multiple types of errors that can occur during one optimization process. mlrMBO tries to handle most of them as smart as possible.

The target function could

- 1The target function returns NA(s) or NaN(s) (plural for the multi-objective case).
- 2The target function stops with an error.
- 3The target function does not return at all (infinite or very long execution time).
- 4The target function crashes the whole R process.
- 5The surrogate machine learning model might crash. Kriging quite often can run into numerical problems.
- 6The proposal mechanism - in multipoint or single point mode - produces a point which is either close to another candidate point in the same iteration or an already visited point in a previous iteration.
- 7The mbo process exits / stops / crashes itself. Maybe because it hit a walltime.

**Mechanism I - Objective value imputation** Issues 1-4 all have in common that the optimizer does not obtain a useful objective value. 3-4 are problematic, because we completely lose control of the R process. We are currently only able to handle them, if you are parallelizing your optimization via [parallelMap](#) and use the BatchJobs mode. In this case, you can specify a walltime (handles 3) and the function evaluation is performed in a separate R process (handles 4). A later path might be to allow function evaluation in a separate process in general, with a capping time. If you really need this now, you can always do this yourself.

Now back to the problem of invalid objective values. By default, the mbo function stops with an error (if it still has control of the process). But in many cases you still want the algorithm to continue. Hence, mbo allows imputation of bad values via the control option `impute.y.fun`.

**Logging:** All error messages are logged into the optimization path `opt.path` if problems occur.

**Mechanism II - The mlr's `on.learner.error`** This produces a FailureModel for the surrogate, if it crashed (issue 6). A random point (or multiple ones) are proposed now for the current iteration. And we pray that we can fit the model again in the next iteration. **Logging:** The entry “model.error” is set in the `opt.path`.

**Mechanism III - Filtering of proposed point which are too close**

Issue 6 is solved by filtering points that are too close to other proposed points or points already proposed in preceding iterations. Filtering in this context means replacing the proposed points by a randomly generated new point. The heuristics mechanism is (de)activated via the logical `filter.proposed.points.tol` parameter of the [setMBOControlInfill](#) function, which defaults to TRUE. (closeness of two points is determined via the `filter.proposed.points.tol` parameter).

**Logging:** The logical entry “filtered.point” is set in the `opt.path` indicating whether the corresponding point was filtered.

### Mechanism IV - Continue optimization process

The mechanism is a save-state-then-continue-mechanism, that allows you to continue your optimization after your system or the optimization process crashed for some reason (issue 7). The `mbo` function has the option to save the current state after certain iterations of the main loop on disk via the control option `save.on.disk.at` of `makeMBOControl`. Note that this saving mechanism is *disabled* by default. Here you can specify, after which iteration you want the current state to be saved (option `save.on.disk.at`). Notice that 0 denotes saving the initial design and `iters + 1` denotes saving the final results. With `mboContinue` you can continue the optimization from the last saved state. This function only requires the path of the saved state.

You will get a warning if you turn on saving in general, but not for the the final result, as this seems a bit stupid. `save.file.path` defines the path of the RData file where the state is stored. It is overwritten (= extended) in each saving iteration.

---

exampleRun	<i>Perform an mbo run on a test function and and visualize what happens.</i>
------------	--

---

### Description

Usually used for 1D or 2D examples, useful for figuring out how stuff works and for teaching purposes. Currently only parameter spaces with numerical parameters are supported. For visualization, run `plotExampleRun` on the resulting object. What is displayed is documented here: `plotExampleRun`. Rendering the plots without displaying them is possible via the function `renderExampleRunPlot`.

Please note the following things: - The true objective function (and later everything which is predicted from our surrogate model) is evaluated on a regular spaced grid. These evaluations are stored in the result object. You can control the resolution of this grid via `points.per.dim`. Parallelization of these evaluations is possible with the R package `parallelMap` on the level `mlrMBO.feval`. - In every iteration the fitted, approximating surrogate model is stored in the result object (via `store.model.at` in control) so we can later visualize it quickly. - The global optimum of the function (if defined) is extracted from the passed smooof function. - If the passed objective function fun does not provide the true, unnoisy objective function some features will not be displayed (for example the gap between the best point so far and the global optimum).

### Usage

```
exampleRun(fun, design = NULL, learner = NULL, control,
  points.per.dim = 50, noisy.eval = 10,
  show.info = getOption("mlrMBO.show.info", TRUE))
```

### Arguments

fun	[smooof_function] Fitness function to optimize. For one dimensional target functions you can obtain a smooof_function by using <code>makeSingleObjectiveFunction</code> . For multi dimensional functions use <code>makeMultiObjectiveFunction</code> . It is possible to return even more information which will be stored in the optimization path. To
-----	---

achieve this, simply append the attribute “extras” to the return value of the target function. This has to be a named list of scalar values. Each of these values will be stored additionally in the optimization path.

design	[data.frame] Initial design as data frame. If the y-values are not already present in design, mbo will evaluate the points. If the parameters have corresponding trafo functions, the design must not be transformed before it is passed! Functions to generate designs are available in ParamHelpers: <a href="#">generateDesign</a> , <a href="#">generateGridDesign</a> , <a href="#">generateRandomDesign</a> . Default is NULL, which means <a href="#">generateDesign</a> is called and a design of size 4 times number of all parameters is created. The points are drawn via <a href="#">maximinLHS</a> to maximize the minimal distance between design points.
learner	[ <a href="#">Learner</a> ] Regression learner from mlr, which is used as a surrogate to model our fitness function. If NULL (default), the default learner is determined as described here: <a href="#">mbo_default_learner</a> .
control	[ <a href="#">MBOControl</a> ] Control object for mbo.
points.per.dim	[integer] Number of (regular spaced) locations at which to sample the fun function per dimension. Default is 50.
noisy.eval	[integer(1)] Number of function evaluations per point if fun is noisy. Default is 10.
show.info	[logical(1)] Verbose output on console? Default is TRUE.

### Value

MBOExampleRun

---

exampleRunMultiObj	<i>Perform an MBO run on a multi-objective test function and and visualize what happens.</i>
--------------------	--

---

### Description

Only available for 2D -> 2D examples, useful for figuring out how stuff works and for teaching purposes. Currently only parameter spaces with numerical parameters are supported. For visualization, run `plotExampleRun` on the resulting object. What is displayed is documented here: [plotExampleRun](#).

### Usage

```
exampleRunMultiObj(fun, design = NULL, learner, control,
  points.per.dim = 50, show.info = getOption("mlrMBO.show.info", TRUE),
  nsga2.args = list(), ...)
```

**Arguments**

fun	[smoof_function] Fitness function to optimize. For one dimensional target functions you can obtain a smoof_function by using <a href="#">makeSingleObjectiveFunction</a> . For multi dimensional functions use <a href="#">makeMultiObjectiveFunction</a> . It is possible to return even more information which will be stored in the optimization path. To achieve this, simply append the attribute “extras” to the return value of the target function. This has to be a named list of scalar values. Each of these values will be stored additionally in the optimization path.
design	[data.frame] Initial design as data frame. If the y-values are not already present in design, mbo will evaluate the points. If the parameters have corresponding trafo functions, the design must not be transformed before it is passed! Functions to generate designs are available in ParamHelpers: <a href="#">generateDesign</a> , <a href="#">generateGridDesign</a> , <a href="#">generateRandomDesign</a> . Default is NULL, which means <a href="#">generateDesign</a> is called and a design of size 4 times number of all parameters is created. The points are drawn via <a href="#">maximinLHS</a> to maximize the minimal distance between design points.
learner	[Learner] Regression learner from mlr, which is used as a surrogate to model our fitness function. If NULL (default), the default learner is determined as described here: <a href="#">mbo_default_learner</a> .
control	[MBOControl] Control object for mbo.
points.per.dim	[integer] Number of (regular spaced) locations at which to sample the fun function per dimension. Default is 50.
show.info	[logical(1)] Verbose output on console? Default is TRUE.
nsga2.args	[list] Further arguments passed to the nsga2 call. Default is list().
...	[any] Further arguments passed to the learner.

**Value**

MBOExampleRunMultiObj

**Note**

If the passed objective function has no associated reference point  $\max(y_i) + 1$  of the nsga2 front is used.

---

getGlobalOpt	<i>Helper function which returns the (estimated) global optimum.</i>
--------------	--

---

**Description**

Helper function which returns the (estimated) global optimum.

**Usage**

```
getGlobalOpt(run)
```

**Arguments**

run	[MBOExampleRun] Object of type MBOExampleRun.
-----	--

**Value**

numeric(1) . (Estimated) global optimum.

---

getMBOInfillCrit	<i>Get properties of MBO infill criterion.</i>
------------------	--

---

**Description**

Returns properties of an infill criterion, e.g., name or id.

**Usage**

```
getMBOInfillCritParams(x)

getMBOInfillCritParam(x, par.name)

getMBOInfillCritName(x)

getMBOInfillCritId(x)

hasRequiresInfillCritStandardError(x)

getMBOInfillCritComponents(x)
```

**Arguments**

x	[MBOInfillCrit] Infill criterion.
par.name	[character(1)] Parameter name.

---

`getSupportedInfillOptFunctions`*Get names of supported infill-criteria optimizers.*

---

**Description**

None.

**Usage**

```
getSupportedInfillOptFunctions()
```

**Value**

character

---

`getSupportedMultipointInfillOptFunctions`*Get names of supported multipoint infill-criteria optimizers.*

---

**Description**

Returns all names of supported multipoint infill-criteria optimizers.

**Usage**

```
getSupportedMultipointInfillOptFunctions()
```

**Value**

character



---

infillcrits	<i>Infill criteria.</i>
-------------	-------------------------

---

### Description

**mlrMBO** contains most of the most popular infill criteria, e.g., expected improvement, (lower) confidence bound etc. Moreover, custom infill criteria may be generated with the [makeMBOInfillCrit](#) function.

### Usage

```
makeMBOInfillCritMeanResponse()

makeMBOInfillCritStandardError()

makeMBOInfillCritEI(se.threshold = 1e-06)

makeMBOInfillCritCB(cb.lambda = NULL)

makeMBOInfillCritAEI(aei.use.nugget = FALSE, se.threshold = 1e-06)

makeMBOInfillCritEQI(eqi.beta = 0.75, se.threshold = 1e-06)

makeMBOInfillCritDIB(cb.lambda = 1, sms.eps = NULL)
```

### Arguments

se.threshold	[numeric(1)] In order to avoid numerical problems the standard error estimation is assumed to be exactly zero, if it is below se.threshold. Default is 1e-6.
cb.lambda	[numeric(1)   NULL] Lambda parameter for confidence bound infill criterion. Default is NULL, which means 1 in case of a fully numeric parameter set and 2 otherwise.
aei.use.nugget	[logical(1)] Should the nugget effect be used for the pure variance estimation for augmented expected improvement? Default is FALSE.
eqi.beta	[numeric(1)] Beta parameter for expected quantile improvement criterion. Default is 0.75.
sms.eps	[numeric(1)   NULL] Epsilon for epsilon-dominance for dib.indicator = "sms". Default is NULL, in this case it is adaptively set.

### Details

In the multi-objective case we recommend to set `cb.lambda` to  $q(0.5 \cdot \pi_{CB}^{(1/n)})$  where  $q$  is the quantile function of the standard normal distribution,  $\pi_{CB}$  is the probability of improvement value and  $n$  is the number of objectives of the considered problem.

**See Also**[makeMBOInfillCrit](#)

---

`initCrit`*Initialize an MBO infill criterion.*

---

**Description**

Some infill criteria have parameters that are dependent on values in the parameter set, design, used learner or other control settings. To actually set these default values, this function is called, which returns a fully initialized [MBOInfillCrit](#). This function is mainly for internal use. If a custom infill criterion is created, it may be required to create a separate method `initCrit.InfillCritID` where ID is the id of the custom [MBOInfillCrit](#).

**Usage**

```
initCrit(crit, fun, design, learner, control)
```

**Arguments**

<code>crit</code>	<a href="#">[MBOInfillCrit]</a> Uninitialized infill criterion.
<code>fun</code>	<a href="#">[smoof_function]</a> Fitness function to optimize.
<code>design</code>	Sampling plan.
<code>learner</code>	<a href="#">[Learner]</a> Regression learner from mlr, which is used as a surrogate to model our fitness function.
<code>control</code>	<a href="#">[MBOControl]</a> MBO control object.

**Value**[MBOInfillCrit](#)

---

makeMBOControl	<i>Set MBO options.</i>
----------------	-------------------------

---

## Description

Creates a control object for MBO optimization.

## Usage

```
makeMBOControl(n.objectives = 1L, propose.points = 1L,
  final.method = "best.true.y", final.eval.s = 0L, y.name = "y",
  impute.y.fun = NULL, trafo.y.fun = NULL, suppress.eval.errors = TRUE,
  save.on.disk.at = integer(0L), save.on.disk.at.time = Inf,
  save.file.path = file.path(getwd(), "mlr_run.RData"),
  store.model.at = NULL, resample.at = integer(0),
  resample.desc = makeResampleDesc("CV", iter = 10),
  resample.measures = list(mse), output.num.format = "%.3g")
```

## Arguments

n.objectives	[integer(1)] How many objectives are to be optimized? n.objectives = 1 implies normal single criteria optimization, n.objectives > 1 implies multi-objective optimization. Default is 1.
propose.points	[integer(1)] Number of proposed / really evaluated points each iteration. Default is 1.
final.method	[character(1)] How should the final point be proposed. Possible values are: "best.true.y": Return best point ever visited according to true value of target function. Can be bad if target function is noisy. "last.proposed": Return the last point proposed by the model. "best.predicted": Use the final model to predict all points ever visited and use the best one. This might average-out noisy function values. Default is: "best.true.y".
final.eval.s	[integer(1)] How many target function eval.s should be done at final point to reduce noise? Default is 0.
y.name	[character] Vector for names of y-columns for target values in optimization path. Default is "y_i", i = 1, ..., n.objectives.
impute.y.fun	[function(x, y, opt.path, ...)*] Functions that gets triggered if your objective evaluation produced a) an exception b) a return object of invalid type c) a numeric vector that contains NA, NaN, Inf. You now have a chance to handle this. You are expected to return a numeric vector of the correct length with concrete values. The optimization path will show some information whether y-values were imputed and what the original, faulty object was. x is the current x-value, y the current (invalid) y-object

	(or an error object) and <code>opt.path</code> the current optimization path. Default is NULL which means to stop if the objective function did not produce the desired result.
<code>trafo.y.fun</code>	[MBOTrafoFunction] Sometimes it is favourable to transform the target function values before modelling. Provide a MBO transformation function to do so.
<code>suppress.eval.errors</code>	[logical(1)] Should reporting of error messages during target function evaluations be suppressed? Only used if <code>impute.errors</code> is TRUE. Default is TRUE.
<code>save.on.disk.at</code>	[integer] Sequential optimization iteration when the actual state should be saved on disk. Iteration 0 denotes the initial design. If the optimization stops with an crucial error, it can be restarted with this file via the function <a href="#">mboContinue</a> . Default is <code>integer(0L)</code> , i. e., not to save.
<code>save.on.disk.at.time</code>	[integer] Same as above. But here you define the time which have to be passed until the last save in seconds. Any finite value will lead to save at end. Default is <code>Inf</code> , i. e., not to save ever.
<code>save.file.path</code>	[character(1)] If <code>save.on.disk.at</code> is used, this is the name of the file where the data will be saved. Default “mbo_run.RData” in your current working directory.
<code>store.model.at</code>	[integer] Sequential optimization iterations when the model should be saved. Iteration 1 is the model fit for the initial design, <code>iters + 1</code> is a final save containing the final results of the optimization. . Default is <code>iters + 1</code> .
<code>resample.at</code>	[integer] At which iterations should the model be resampled and assessed? Iteration 0 does some resampling on the initial design. Default is none.
<code>resample.desc</code>	[ResampleDesc] How should the model be resampled? Default is 10-fold CV.
<code>resample.measures</code>	[list of Measure] Performance measures to assess model with during resampling. Default is <a href="#">mse</a> .
<code>output.num.format</code>	[logical(1)] Format string for the precision of the numeric output of mbo.

**Value**

[MBOControl](#) .

**See Also**

Other MBOControl: [setMBOControlInfill](#), [setMBOControlMultiObj](#), [setMBOControlMultiPoint](#), [setMBOControlTermination](#)

---

makeMBOInfillCrit	Create an infill criterion.
-------------------	-----------------------------

---

## Description

The infill criterion guides the model based search process. The most prominent infill criteria, e.g., expected improvement, lower confidence bound and others, are already implemented in `mlrMBO`. Moreover, the package allows for the creation of custom infill criteria.

## Usage

```
makeMBOInfillCrit(fun, name, id, opt.direction = "minimize",
  components = character(0L), params = list(), requires.se = FALSE)
```

## Arguments

<code>fun</code>	<p>[function(points, models, control, par.set, design, iter)]          A function which expects the following parameters in exactly this order and return a numeric vector of criteria values at the points:</p> <p><b>points</b> [data.frame ] n points where to evaluate.  <b>models</b> [WrappedModel   list ] Model(s) fitted on design.  <b>control</b> [MBOControl ] Control object.  <b>par.set</b> [ParamSet ] Parameter set.  <b>design</b> [data.frame ] Design of already visited points.  <b>iter</b> [integer(1) ] Current iteration.  <b>attributes</b> [logical{1} ] Are there attributes appended to the return value that should be added to the OptPath?</p> <p>Important: Internally, this function will be minimized. So the proposals will be where this function is low.</p>
<code>name</code>	<p>[character(1)]          Full name of the criterion.</p>
<code>id</code>	<p>[character(1)]          Short name of the criterion. Used internally and in plots.</p>
<code>opt.direction</code>	<p>[character(1)]          Only for visualization: Shall this criterion be plotted as if it were to be minimized (minimize), maximized (maximize) or is the direction the same as for the objective function (objective)? Default is minimize.</p>
<code>components</code>	<p>[character]          Infill criteria may not return proposed point(s) only. Additional information can be returned by appending a named list "crit.components" to the returned value as an attribute. The components argument takes a character vector of the names of the meta information, i.e., the names of the named "crit.components" list. Default is the empty character vector.</p>

params	[list] Named list of parameters for the infill criterion. There values may be used by <b>mlrMBO</b> internally. Default is the empty list.
requires.se	[logical(1)] Does the infill criterion require the regression learner to provide a standard error estimation? Default is FALSE.

**Value**

MBOInfillCrit

**Predefined standard infill criteria**

- crit.ei** Expected Improvement
- crit.mr** Mean response
- crit.se** Standard error
- crit.cb** Confidence bound with lambda automatically chosen, see [infillcrits](#)
- crit.cb1** Confidence bound with lambda=1
- crit.cb2** Confidence bound with lambda=2
- crit.aei** Augmented expected improvement
- crit.eq** Expected quantile improvement
- crit.dib1** Direct indicator-based with lambda=1

makeMBOLearner

*Generate default learner.***Description**

This is a helper function that generates a default surrogate, based on properties of the objective function and the selected infill criterion.

For numeric-only (including integers) parameter spaces without any dependencies:

- A Kriging model “regr.km” with kernel “matern3\_2” is created.
- If the objective function is deterministic we add a small nugget effect ( $10^{-8} \cdot \text{Var}(y)$ ,  $y$  is vector of observed outcomes in current design) to increase numerical stability to hopefully prevent crashes of DiceKriging.
- If the objective function is noisy the nugget effect will be estimated with `nugget.estim = TRUE` (but you can override this in `...`. Also `jitter` is set to `TRUE` to circumvent a problem with DiceKriging where already trained input values produce the exact trained output. For further information check the `$note` slot of the created learner.
- Instead of the default “BFGS” optimization method we use `rgenoud` (“gen”), which is a hybrid algorithm, to combine global search based on genetic algorithms and local search based on gradients. This may improve the model fit and will less frequently produce a constant surrogate model. You can also override this setting in `...`

For mixed numeric-categorical parameter spaces, or spaces with conditional parameters:

- A random regression forest “`regr.randomForest`” with 500 trees is created.
- The standard error of a prediction (if required by the infill criterion) is estimated by computing the jackknife-after-bootstrap. This is the `se.method = "jackknife"` option of the [`regr.randomForest`](#), see this page for further info and alternatives.

If additionally dependencies are in present in the parameter space, inactive conditional parameters are represented by missing NA values in the training design `data.frame`. We simply handle those with an imputation method, added to the random forest:

- If a numeric value is inactive, i.e., missing, it will be imputed by 2 times the maximum of observed values
- If a categorical value is inactive, i.e., missing, it will be imputed by the special class label “`__miss__`”

Both of these techniques make sense for tree-based methods and are usually hard to beat, see Ding et al. (2010).

## Usage

```
makeMBOLearner(control, fun, config = list(), ...)
```

## Arguments

<code>control</code>	<a href="#">[MBOControl]</a> Control object for mbo.
<code>fun</code>	<a href="#">[smoof_function]</a> The same objective function which is also passed to <a href="#">mbo</a> .
<code>config</code>	<a href="#">[named list]</a> Named list of config option to overwrite global settings set via <a href="#">configureMlr</a> for this specific learner.
<code>...</code>	<a href="#">[any]</a> Further parameters passed to the constructed learner. Will overwrite <code>mlrMBO</code> ’s defaults.

## Value

Learner

## References

Ding, Yufeng, and Jeffrey S. Simonoff. An investigation of missing data methods for classification trees applied to binary response data. *Journal of Machine Learning Research* 11.Jan (2010): 131-170.

---

makeMBOTrafoFunction	<i>Create a transformation function for MBOExampleRun.</i>
----------------------	--

---

### Description

Creates a transformation function for MBOExampleRun.

### Usage

```
makeMBOTrafoFunction(name, fun)
```

### Arguments

name	[character(1)] Name of the transformation.
fun	[function] R function which expects a numeric vector.

### Value

Object of type MBOTrafoFunction.

### See Also

[trafos](#)

---

mbo	<i>Optimizes a function with sequential model based optimization.</i>
-----	---

---

### Description

See [mbo\\_parallel](#) for all parallelization options.

### Usage

```
mbo(fun, design = NULL, learner = NULL, control,  
     show.info = getOption("mlrMBO.show.info", TRUE), more.args = list())
```



## Arguments

fun	[smoof_function] Fitness function to optimize. For one dimensional target functions you can obtain a smoof_function by using <a href="#">makeSingleObjectiveFunction</a> . For multi dimensional functions use <a href="#">makeMultiObjectiveFunction</a> . It is possible to return even more information which will be stored in the optimization path. To achieve this, simply append the attribute “extras” to the return value of the target function. This has to be a named list of scalar values. Each of these values will be stored additionally in the optimization path.
design	[data.frame] Initial design as data frame. If the y-values are not already present in design, mbo will evaluate the points. If the parameters have corresponding trafo functions, the design must not be transformed before it is passed! Functions to generate designs are available in ParamHelpers: <a href="#">generateDesign</a> , <a href="#">generateGridDesign</a> , <a href="#">generateRandomDesign</a> . Default is NULL, which means <a href="#">generateDesign</a> is called and a design of size 4 times number of all parameters is created. The points are drawn via <a href="#">maximinLHS</a> to maximize the minimal distance between design points.
learner	[Learner] Regression learner from mlr, which is used as a surrogate to model our fitness function. If NULL (default), the default learner is determined as described here: <a href="#">mbo_default_learner</a> .
control	[MBOControl] Control object for mbo.
show.info	[logical(1)] Verbose output on console? Default is TRUE.
more.args	[list] Further arguments passed to fitness function.

## Value

[MBOSingleObjResult](#) | [MBOMultiObjResult](#)

## Examples

```
# simple 2d objective function
obj.fun = makeSingleObjectiveFunction(
  fn = function(x) x[1]^2 + sin(x[2]),
  par.set = makeNumericParamSet(id = "x", lower = -1, upper = 1, len = 2)
)

# create base control object
ctrl = makeMBOControl()

# do three MBO iterations
ctrl = setMBOControlTermination(ctrl, iters = 3L)

# use 500 points in the focussearch (should be sufficient for 2d)
```

```

ctrl = setMBOControlInfill(ctrl, opt.focussearch.points = 500)
# create initial design
des = generateDesign(n = 5L, getParamSet(obj.fun), fun = lhs::maximinLHS)

# start mbo
res = mbo(obj.fun, design = des, control = ctrl)

print(res)
## Not run:
plot(res)

## End(Not run)

```

---

mboContinue

*Continues an mbo run from a save-file.*


---

### Description

Useful if your optimization is likely to crash, so you can continue from a save point and will not lose too much information and runtime.

### Usage

```
mboContinue(file)
```

### Arguments

file	[character(1)] File path of saved MBO state. See <code>save.on.disk.at</code> argument of <a href="#">MBOControl</a> object.
------	---

### Value

See [mbo](#).

---

mboFinalize

*Finalizes an mbo run from a save-file.*


---

### Description

Useful if your optimization didn't terminate but you want a results nonetheless.

### Usage

```
mboFinalize(file)
```

**Arguments**

`file` [character(1)]  
File path of saved MBO state. See `save.on.disk.at` argument of [MBOControl](#) object.

**Value**

See [mbo](#).

---

MBOMultiObjResult      *Multi-Objective result object.*

---

**Description**

- `pareto.front` [matrix] Pareto front of all evaluated points.
- `pareto.set` [list of lists] Pareto set of all evaluated points.
- `pareto.inds` [numeric] Indices of the Pareto-optimal points in the `opt.path`
- `opt.path` [[OptPath](#)] Optimization path. Includes all evaluated points and additional information as documented in [mbo\\_OptPath](#). You can convert it via `as.data.frame`.
- `final.state` [character] The final termination state. Gives information why the optimization ended
- `models` [List of [WrappedModel](#)] List of saved regression models.
- `control` [[MBOControl](#)] Control object used in optimization

---

MBOSingleObjResult      *Single-Objective result object.*

---

**Description**

- `x` [list] Named list of proposed optimal parameters.
- `y` [numeric(1)] Value of objective function at `x`, either from evals during optimization or from requested final evaluations, if those were greater than 0.
- `best.ind` [numeric(1)] Index of `x` in the `opt.path`.
- `opt.path` [[OptPath](#)] Optimization path. Includes all evaluated points and additional information as documented in [mbo\\_OptPath](#). You can convert it via `as.data.frame`.
- `resample.results` [List of [ResampleResult](#)] List of the desired `resample.results` if `resample.at` is set in `makeMBOControl`.
- `final.state` [character] The final termination state. Gives information why the optimization ended. Possible values are
  - term.iter** Maximal number of iterations reached.
  - term.time** Maximal running time exceeded.

**term.exectime** Maximal execution time of function evaluations reached.

**term.yval** Target function value reached.

**term.fevals** maximal number of function evaluations reached.

**term.custom** Terminated due to custom, user-defined termination condition.

- **models** [List of [WrappedModel](#)] List of saved regression models if `store.model.at` is set in `makeMBOControl`. The default is that it contains the model generated after the last iteration.
- **control** [MBOControl] Control object used in optimization

---

mbo\_OptPath

*OptPath in mlrMBO*


---

## Description

In `mlrMBO` the [OptPath](#) contains extra information next to the information documented in [OptPath](#). The extras are:

**train.time** Time to train the model(s) that produced the points. Only the first slot of the vector is used (if we have multiple points), rest are NA.

**propose.time** Time needed to propose the point. If we have individual timings from the proposal mechanism, we have one different value per point here. If all were generated in one go, we only have one timing, we store it in the slot for the first point, rest are NA.

**errors.model** Possible Error Messages. If point-producing model(s) crashed they are replicated for all n points, if only one error message was passed we store it for the first point, rest are NA.

**prop.type** Type of point proposal. Possible values are

**initdesign** Points actually not proposed, but in the initial design.

**infill\\_x** Here x is a placeholder for the selected infill criterion, e.g., `infill\_ei` for expected improvement.

**random\\_interleave** Uniformly sampled points added additionally to the proposed points.

**random\\_filtered** If filtering of proposed points located too close to each other is active, these are replaced by random points.

**final\\_eval** If `final.evals` is set in [makeMBOControl](#): Final evaluations of the proposed solution to reduce noise in y.

**parego.weight** Weight vector sampled for multipoint ParEGO

... Depending on the chosen infill criterion there will be additional columns, e.g. `se` and `mean` for the Expected Improvement)

Moreover, the user may pass additional “user extras” by appending a named list of scalar values to the return value of the objective function.

mbo\_parallel

Parallelization in mlrMBO

## Description

In mlrMBO you can parallelize the tuning on two different levels to speed up computation:

- `mlrMBO.feval` Multiple evaluations of the target function.
- `mlrMBO.propose.points` Optimization of the infill criteria if multiple are used (e.g. ParEGO and ParallelLCB)

Internally the evaluation of the target function is realized with the R package `parallelMap`. See the mlrMBO tutorial and the Github project pages of `parallelMap` for instructions on how to set up parallelization. The different levels of parallelization can be specified in `parallelStart*`. Details for the levels mentioned above are given below:

- Evaluation of the objective function can be parallelized in cases multiple points are to be evaluated at once. These are: evaluation of the initial design, multiple proposed points per iteration and evaluation of the target function in [exampleRun](#). (Level: `mlrMBO.feval`)
- Model fitting / point proposal - in some cases where independent, expensive operations are performed. (Level: `mlrMBO.propose.points`)

Details regarding the latter:

**single-objective MBO with LCB multipoint** Parallel optimization of LCBs for the lambda-values.

**Multi-objective MBO with ParEGO** Parallel optimization of scalarization functions.

mlrMBO\_examples

mlrMBO examples

## Description

Different scenarios of the usage of mlrMBO and visualizations.

## Examples

```
#####
###
### optimizing a simple sin(x) with mbo / EI
###
#####
## Not run:
library(ggplot2)
library(mlrMBO)
configureMlr(show.learner.output = FALSE)
set.seed(1)
```

```

obj.fun = makeSingleObjectiveFunction(
  name = "Sine",
  fn = function(x) sin(x),
  par.set = makeNumericParamSet(lower = 3, upper = 13, len = 1),
  global.opt.value = -1
)

ctrl = makeMBOControl(propose.points = 1)
ctrl = setMBOControlTermination(ctrl, iters = 10L)
ctrl = setMBOControlInfill(ctrl, crit = makeMBOInfillCritEI(),
  opt = "focussearch", opt.focussearch.points = 500L)

lrn = makeMBOLearner(ctrl, obj.fun)

design = generateDesign(6L, getParamSet(obj.fun), fun = lhs::maximinLHS)

run = exampleRun(obj.fun, design = design, learner = lrn,
  control = ctrl, points.per.dim = 100, show.info = TRUE)

plotExampleRun(run, densregion = TRUE, gg.objects = list(theme_bw()))

## End(Not run)
#####
###
### optimizing branin in 2D with mbo / EI #####
###
#####
## Not run:
library(mlrMBO)
library(ggplot2)
set.seed(1)
configureMlr(show.learner.output = FALSE)

obj.fun = makeBraninFunction()

ctrl = makeMBOControl(propose.points = 1L)
ctrl = setMBOControlTermination(ctrl, iters = 10L)
ctrl = setMBOControlInfill(ctrl, crit = makeMBOInfillCritEI(),
  opt = "focussearch", opt.focussearch.points = 2000L)

lrn = makeMBOLearner(ctrl, obj.fun)
design = generateDesign(10L, getParamSet(obj.fun), fun = lhs::maximinLHS)

run = exampleRun(obj.fun, design = design, learner = lrn, control = ctrl,
  points.per.dim = 50L, show.info = TRUE)

print(run)

plotExampleRun(run, gg.objects = list(theme_bw()))

## End(Not run)
#####
###

```

```

### optimizing a simple sin(x) with multipoint proposal
###
#####
## Not run:
library(mlrMBO)
library(ggplot2)
set.seed(1)
configureMlr(show.learner.output = FALSE)

obj.fun = makeSingleObjectiveFunction(
  name = "Sine",
  fn = function(x) sin(x),
  par.set = makeNumericParamSet(lower = 3, upper = 13, len = 1L),
  global.opt.value = -1
)

ctrl = makeMBOControl(propose.points = 2L)
ctrl = setMBOControlTermination(ctrl, iters = 10L)
ctrl = setMBOControlInfill(ctrl, crit = makeMBOInfillCritMeanResponse())
ctrl = setMBOControlMultiPoint(
  ctrl,
  method = "moimbo",
  moimbo.objective = "ei.dist",
  moimbo.dist = "nearest.neighbor",
  moimbo.maxit = 200L
)

lrn = makeMBOLEARNER(ctrl, obj.fun)

design = generateDesign(4L, getParamSet(obj.fun), fun = lhs::maximinLHS)

run = exampleRun(obj.fun, design = design, learner = lrn,
  control = ctrl, points.per.dim = 100, show.info = TRUE)

print(run)

plotExampleRun(run, densregion = TRUE, gg.objects = list(theme_bw()))

## End(Not run)
#####
###
### optimizing branin in 2D with multipoint proposal #####
###
#####
## Not run:
library(mlrMBO)
library(ggplot2)
set.seed(2)
configureMlr(show.learner.output = FALSE)

obj.fun = makeBraninFunction()

ctrl = makeMBOControl(propose.points = 5L)

```

```

ctrl = setMBOControlInfill(ctrl, crit = makeMBOInfillCritMeanResponse())
ctrl = setMBOControlTermination(ctrl, iters = 10L)
ctrl = setMBOControlMultiPoint(ctrl,
  method = "moimbo",
  moimbo.objective = "ei.dist",
  moimbo.dist = "nearest.neighbor",
  moimbo.maxit = 200L
)

lrn = makeLearner("regr.km", predict.type = "se")
design = generateDesign(10L, getParamSet(obj.fun), fun = lhs::maximinLHS)

run = exampleRun(obj.fun, design = design, learner = lrn, control = ctrl,
  points.per.dim = 50L, show.info = TRUE)

print(run)

plotExampleRun(run, gg.objects = list(theme_bw()))

## End(Not run)
#####
###
### optimizing a simple noisy sin(x) with mbo / EI
###
#####
## Not run:
library(mlrMBO)
library(ggplot2)
set.seed(1)
configureMlr(show.learner.output = FALSE)

# function with noise
obj.fun = makeSingleObjectiveFunction(
  name = "Some noisy function",
  fn = function(x) sin(x) + rnorm(1, 0, 0.1),
  par.set = makeNumericParamSet(lower = 3, upper = 13, len = 1L),
  noisy = TRUE,
  global.opt.value = -1,
  fn.mean = function(x) sin(x)
)

ctrl = makeMBOControl(
  propose.points = 1L,
  final.method = "best.predicted",
  final.evals = 10L
)
ctrl = setMBOControlTermination(ctrl, iters = 5L)

ctrl = setMBOControlInfill(ctrl, crit = makeMBOInfillCritEI(),
  opt = "focussearch", opt.focussearch.points = 500L)

lrn = makeMBOLEarner(ctrl, obj.fun)

```



```

design = generateDesign(6L, getParamSet(obj.fun), fun = lhs::maximinLHS)

run = exampleRun(obj.fun, design = design, learner = lrn,
  control = ctrl, points.per.dim = 200L, noisy.eval = 50L,
  show.info = TRUE)

print(run)

plotExampleRun(run, densregion = TRUE, gg.objects = list(theme_bw()))

## End(Not run)
#####
###
### optimizing 1D fun with 3 categorical level and
### noisy outout with random forest
###
#####
## Not run:
library(mlrMBO)
library(ggplot2)
set.seed(1)
configureMlr(show.learner.output = FALSE)

obj.fun = makeSingleObjectiveFunction(
  name = "Mixed decision space function",
  fn = function(x) {
    if (x$foo == "a") {
      return(5 + x$bar^2 + rnorm(1))
    } else if (x$foo == "b") {
      return(4 + x$bar^2 + rnorm(1, sd = 0.5))
    } else {
      return(3 + x$bar^2 + rnorm(1, sd = 1))
    }
  },
  par.set = makeParamSet(
    makeDiscreteParam("foo", values = letters[1:3]),
    makeNumericParam("bar", lower = -5, upper = 5)
  ),
  has.simple.signature = FALSE, # function expects a named list of parameter values
  noisy = TRUE
)

ctrl = makeMBOControl()
ctrl = setMBOControlTermination(ctrl, iters = 10L)

# we can basically do an exhaustive search in 3 values
ctrl = setMBOControlInfill(ctrl, crit = makeMBOInfillCritEI(),
  opt.restarts = 1L, opt.focussearch.points = 3L, opt.focussearch.maxit = 1L)

design = generateDesign(20L, getParamSet(obj.fun), fun = lhs::maximinLHS)

lrn = makeMBOLE learner(ctrl, obj.fun)

```

```

run = exampleRun(obj.fun, design = design, learner = lrn, control = ctrl,
points.per.dim = 50L, show.info = TRUE)

print(run)
plotExampleRun(run, densregion = TRUE, gg.objects = list(theme_bw()))

## End(Not run)
#####
###
### optimizing mixed space function
###
#####
## Not run:
library(mlrMBO)
library(ggplot2)
set.seed(1)
configureMlr(show.learner.output = FALSE)

obj.fun = makeSingleObjectiveFunction(
  name = "Mixed functions",
  fn = function(x) {
    if (x$cat == "a")
      x$num^2
    else
      x$num^2 + 3
  },
  par.set = makeParamSet(
    makeDiscreteParam("cat", values = c("a", "b")),
    makeNumericParam("num", lower = -5, upper = 5)
  ),
  has.simple.signature = FALSE,
  global.opt.value = -1
)

ctrl = makeMBOControl(propose.points = 1L)
ctrl = setMBOControlTermination(ctrl, iters = 10L)
ctrl = setMBOControlInfill(ctrl, crit = makeMBOInfillCritEI(),
  opt = "focussearch", opt.focussearch.points = 500L)

lrn = makeMBOLearner(ctrl, obj.fun)

design = generateDesign(4L, getParamSet(obj.fun), fun = lhs::maximinLHS)

run = exampleRun(obj.fun, design = design, learner = lrn,
  control = ctrl, points.per.dim = 100L, show.info = TRUE)

print(run)

plotExampleRun(run, densregion = TRUE, gg.objects = list(theme_bw()))

## End(Not run)
#####
###

```

```

### optimizing multi-objective function
###
#####
## Not run:
library(mlrMBO)
library(ggplot2)
set.seed(1)
configureMlr(show.learner.output = FALSE)

obj.fun = makeZDT1Function(dimensions = 2L)

ctrl = makeMBOControl(n.objectives = 2L, propose.points = 2L, save.on.disk.at = integer(0L))
ctrl = setMBOControlTermination(ctrl, iters = 5L)
ctrl = setMBOControlInfill(ctrl, crit = makeMBOInfillCritDIB(),
  opt.focussearch.points = 10000L)
ctrl = setMBOControlMultiObj(ctrl, parego.s = 100)
learner = makeMBOLearner(ctrl, obj.fun)

design = generateDesign(5L, getParamSet(obj.fun), fun = lhs::maximinLHS)

run = exampleRunMultiObj(obj.fun, design = design, learner = learner, ctrl, points.per.dim = 50L,
  show.info = TRUE, nsga2.args = list())

plotExampleRun(run, gg.objects = list(theme_bw()))

## End(Not run)
#####
###
### optimizing multi objective function and plots
###
#####
## Not run:
library(mlrMBO)
library(ggplot2)
set.seed(1)
configureMlr(show.learner.output = FALSE)

obj.fun = makeDTLZ1Function(dimensions = 5L, n.objectives = 2L)

ctrl = makeMBOControl(n.objectives = 2L,
  propose.points = 2L)
ctrl = setMBOControlTermination(ctrl, iters = 10L)
ctrl = setMBOControlInfill(ctrl, crit = makeMBOInfillCritEI(), opt.focussearch.points = 1000L,
  opt.focussearch.maxit = 3L)
ctrl = setMBOControlMultiObj(ctrl, method = "parego")
lrn = makeMBOLearner(ctrl, obj.fun)

design = generateDesign(8L, getParamSet(obj.fun), fun = lhs::maximinLHS)

res = mbo(obj.fun, design = design, learner = lrn, control = ctrl, show.info = TRUE)

plot(res)

```

```
## End(Not run)
```

---

OptProblem	<i>OptProblem object.</i>
------------	---------------------------

---

### Description

The OptProblem contains all the constants values which define a OptProblem within our MBO Steps. It is an environment and is always pointed at by the OptState.

---

OptResult	<i>OptResult object.</i>
-----------	--------------------------

---

### Description

The OptResult stores all entities which are not needed while optimizing but are needed to build the final result. It contains fitted surrogate models at certain times as well as resample objects. When the optimization ended it will contain the [MBOResult].

---

OptState	<i>OptState object.</i>
----------	-------------------------

---

### Description

The OptState is the central component of the mbo iterations. This environment contains every necessary information needed during optimization in MBO. It also links to the [OptProblem](#) and to the [OptResult](#).

plotExampleRun

*Renders plots for exampleRun objects and displays them.***Description**

The graphical output depends on the target function at hand. - For 1D numeric functions the upper plot shows the true function (if known), the model and the (infill) points. The lower plot shows the infill criterion. - For 2D mixed target functions only one plot is displayed. - For 2D numeric only target functions up to four plots are presented to the viewer: - levelplot of the true function landscape (with [infill] points), - levelplot of the model landscape (with [infill] points), - levelplot of the infill criterion - levelplot of the standard error (only if learner supports standard error estimation). - For bi-criteria target functions the upper plot shows the target space and the lower plot displays the x-space.

**Usage**

```
plotExampleRun(object, iters, pause = interactive(), densregion = TRUE,
  se.factor = 1, single.prop.point.plots = FALSE, xlim = NULL,
  ylim = NULL, point.size = 3, line.size = 1, trafo = NULL,
  colors = c("red", "blue", "green"), gg.objects = list(), ...)
```

**Arguments**

object	[function] MBOExampleRun object from exampleRun or MBOExampleRunMultiObj object from exampleRunMultiObj.
iters	[integer] Selected iterations of object to produce plots. Default is all iterations.
pause	[logical(1)] Should the process be paused after each iteration? Default is interactive().
densregion	[logical(1)] Should the background be shaded? Default is TRUE. Only used if learner supports computation of standard error.
se.factor	[numeric(1)] If the model provides local standard error estimation, in addition to the mean response $\hat{y}(x) \pm \text{se.factor} * \text{se}(x)$ is plotted above and below. Default is 1.
single.prop.point.plots	[logical(1)] Parameter for MOI-MBO Multipoint proposal: Should every proposed point be displayed in a single plot - or one plot per Iteration? Default is FALSE indicating single plots per proposed points.
xlim	[numeric(2)] For 1D: xlim parameter for first and second plot. Default is range of x-values evaluated in run object x.

ylim	[numeric(2)] For 1D: ylim parameter for first plot, for the second plot ylim is always set automatically, depending on the range of the evaluated infill criterion. Default for the first plot is a heuristic to have the true function and yhat(x) +- se.factor2 * se(x) both in the plot. Note that this heuristic might change the ylim setting between plot iterations.
point.size	[numeric(1)] Point size for plotted points. Default ist 3.
line.size	[numeric(1)] Line width of the graphs of plotted functions.
trafo	[list] List of transformation functions of type MBOTrafoFunction for the different plots. For 1D: The list elements should be named with “y” (applied to objective function and model) or “crit” (applied to the criterion). Only applied to plots with numeric parameters. For 2D: The list should contain at least one element “y”, “yhat”, “crit” or “se”. This way one can specify different transformations for different plots. If a single function is provided, this function is used for all plots.
colors	[character(3)] Specify colors for point in the plots. Must be a vector of length 3, each element a color for the type design, prop and seq respectively. Default is red for the initial design, blue for allready proposed points and green for the actual iteration.
gg.objects	[list]] List of gg objects that should be added to all ggplots.
...	[any] Currently not used.

Value

Nothing.

---

plotMBOResult	<i>MBO Result Plotting</i>
---------------	----------------------------

---

Description

Plots any MBO result objects. Plots for X-Space, Y-Space and any coloumn in the optimization path are available. This function uses [plotOptPath](#) from package ParamHelpers.

Usage

```
## S3 method for class 'MBOSingleObjResult'
plot(x, iters = NULL, pause = interactive(),
     ...)

## S3 method for class 'MBOMultiObjResult'
plot(x, iters = NULL, pause = interactive(),
     ...)
```

**Arguments**

x	[MBOResult] MBOSingleObjResult or MBOMultiObjResult object.
iters	[integer] Iterations to be plotted, 0 indicates the initial design. Default is all iterations.
pause	[logical(1)] Should the process be paused after each iteration? Default is interactive().
...	Additional parameters for the <a href="#">plotOptPath</a> function in package ParamHelpers.

---

print.MBOControl	<i>Print mbo control object.</i>
------------------	----------------------------------

---

**Description**

Print mbo control object.

**Usage**

```
## S3 method for class 'MBOControl'
print(x, ...)
```

**Arguments**

x	[ <a href="#">MBOControl</a> ] Control object.
...	[any] Not used.

---

renderExampleRunPlot	<i>Renders plots for exampleRun objects, either in 1D or 2D, or exampleRunMultiObj objects.</i>
----------------------	---

---

**Description**

The graphical output depends on the target function at hand. - For 1D numeric functions the upper plot shows the true function (if known), the model and the (infill) points. The lower plot shows the infill criterion. - For 2D mixed target functions only one plot is displayed. - For 2D numeric only target functions up to four plots are presented to the viewer: - levelplot of the true function landscape (with [infill] points), - levelplot of the model landscape (with [infill] points), - levelplot of the infill criterion - levelplot of the standard error (only if learner supports standard error estimation). - For bi-criteria target functions the upper plot shows the target space and the lower plot displays the x-space.

**Usage**

```
renderExampleRunPlot(object, iter, densregion = TRUE, se.factor = 1,
  single.prop.point.plots = FALSE, xlim = NULL, ylim = NULL,
  point.size = 3, line.size = 1, trafo = NULL, colors = c("red", "blue",
  "green"), ...)
```

**Arguments**

object	[function] MBOExampleRun or MBOExampleRunMultiObj object.
iter	[integer] Selected iteration of object to render plots for.
densregion	[logical(1)] Should the background be shaded? Default is TRUE. Only used if learner supports computation of standard error.
se.factor	[numeric(1)] If the model provides local standard error estimation, in addition to the mean response $\hat{y}(x) \pm \text{se.factor} * \text{se}(x)$ is plotted above and below. Default is 1.
single.prop.point.plots	[logical(1)] Parameter for MOI-MBO Multipoint proposal: Should every proposed point be displayed in a single plot - or one plot per Iteration? Default is FALSE indicating single plots per proposed points.
xlim	[numeric(2)] For 1D: xlim parameter for first and second plot. Default is range of x-values evaluated in run object object.
ylim	[numeric(2)] For 1D: ylim parameter for first plot, for the second plot ylim is always set automatically, depending on the range of the evaluated infill criterion. Default for the first plot is a heuristic to have the true function and $\hat{y}(x) \pm \text{se.factor}^2 * \text{se}(x)$ both in the plot. Note that this heuristic might change the ylim setting between plot iterations.
point.size	[numeric(1)] Point size for plotted points. Default is 3.
line.size	[numeric(1)] Line width of the graphs of plotted functions.
trafo	[list] List of transformation functions of type MBOTrafoFunction for the different plots. For 1D: The list elements should be named with "y" (applied to objective function and model) or "crit" (applied to the criterion). Only applied to plots with numeric parameters. For 2D: The list should contain at least one element "y", "yhat", "crit" or "se". This way one can specify different transformations for different plots. If a single function is provided, this function is used for all plots.



colors	[character(3)] Specify colors for point in the plots. Must be a vector of length 3, each element a color for the type design, prop and seq respectively. Default is red for the initial design, blue for already proposed points and green for the actual iteration.
...	[any] Currently not used.

### Value

list . List containing separate ggplot object. The number of plots depends on the type of MBO problem. See the description for details.

---

setMBOControlInfill	<i>Extends mbo control object with infill criteria and infill optimizer options.</i>
---------------------	--

---

### Description

Please note that internally all infill criteria are minimized. So for some of them, we internally compute their negated version, e.g., for EI or also for CB when the objective is to be maximized. In the latter case mlrMBO actually computes the negative upper confidence bound and minimizes that.

### Usage

```
setMBOControlInfill(control, crit = NULL, interleave.random.points = 0L,
  filter.proposed.points = NULL, filter.proposed.points.tol = NULL,
  opt = "focussearch", opt.restarts = NULL, opt.focussearch.maxit = NULL,
  opt.focussearch.points = NULL, opt.cmaes.control = NULL,
  opt.ea.maxit = NULL, opt.ea.mu = NULL, opt.ea.sbx.eta = NULL,
  opt.ea.sbx.p = NULL, opt.ea.pm.eta = NULL, opt.ea.pm.p = NULL,
  opt.ea.lambda = NULL, opt.nsga2.popsizes = NULL,
  opt.nsga2.generations = NULL, opt.nsga2.cprob = NULL,
  opt.nsga2.cdists = NULL, opt.nsga2.mprobs = NULL, opt.nsga2.mdists = NULL)
```

### Arguments

control	[MBOControl] Control object for mbo.
crit	[MBOInfillCrit] How should infill points be rated. See <a href="#">infillcrits</a> for an overview of available infill criteria or implement a custom one via <a href="#">makeMBOInfillCrit</a> .# Default is "(lower) confidence bound" (see <a href="#">makeMBOInfillCritCB</a> ).
interleave.random.points	[integer(1)] Add interleave.random.points uniformly sampled points additionally to the regular proposed points in each step. If crit="random" this value will be neglected. Default is 0.

<code>filter.proposed.points</code>	<p>[logical(1)]</p> <p>Design points located too close to each other can lead to numerical problems when using e.g. kriging as a surrogate model. This may solve the 'leading minor of order ...' error during model fit. This parameter activates or deactivates a heuristic to handle this issue. If TRUE, proposed points whose distance to design points or other current candidate points is smaller than <code>filter.proposed.points.tol</code>, are replaced by random points. If enabled, the column entry for <code>prop.type</code> is set to "random_filter" in the resulting <code>opt.path</code>, so you can see whether such a replacement happened. This does only work for numeric parameter sets without any discrete parameters. Default is FALSE.</p>
<code>filter.proposed.points.tol</code>	<p>[numeric(1)]</p> <p>Tolerance value filtering of proposed points. We currently use a maximum metric to calculate the distance between points. Default is 0.0001.</p>
<code>opt</code>	<p>[character(1)]</p> <p>How should SINGLE points be proposed by using the surrogate model. Possible values are: "focussearch": In several iteration steps the parameter space is focused on an especial promising region according to infill criterion. "cmaes": Use CMA-ES (function <a href="#">cmaes</a> from package <b>cmaesr</b> to optimize infill criterion. If all CMA-ES runs fail, a random point is generated instead and a warning informs about it. "ea": Use an (mu+1) EA to optimize infill criterion. "nsga2": NSGA2 for multi obj. optimization. Needed for mspot. Default is "focussearch". Alternatively, you may pass a function name as string.</p>
<code>opt.restarts</code>	<p>[integer(1)]</p> <p>Number of independent restarts for optimizer of infill criterion. If <code>opt == "cmaes"</code> the first start point for the optimizer is always the currently best point in the design of already visited points. Subsequent starting points are chosen according to the CMA-ES restart strategy introduced by Auger and Hansen. For details see the corresponding paper in the references and the help page of the underlying optimizer <a href="#">cmaes</a>. Default is 3.</p>
<code>opt.focussearch.maxit</code>	<p>[integer(1)]</p> <p>For <code>opt = "focussearch"</code>: Number of iteration to shrink local focus. Default is 5.</p>
<code>opt.focussearch.points</code>	<p>[integer(1)]</p> <p>For <code>opt = "focussearch"</code>: Number of points in each iteration of the focus search optimizer. Default is 1000.</p>
<code>opt.cmaes.control</code>	<p>[list]</p> <p>For <code>opt = "cmaes"</code>: Control argument for cmaes optimizer. For the default see the help page of the underlying optimizer <a href="#">cmaes</a>.</p>
<code>opt.ea.maxit</code>	<p>[integer(1)]</p> <p>For <code>opt = "ea"</code>: Number of iterations / generations of EA. Default is 500.</p>
<code>opt.ea.mu</code>	<p>[integer(1)]</p> <p>For <code>opt = "ea"</code>: Population size of EA. The default is 10 times the number of parameters of the function to optimize.</p>

opt.ea.sbx.eta	[numeric(1)] For opt = "ea": Distance parameter of crossover distribution , see <a href="#">sbx_operator</a> . Default is 15.
opt.ea.sbx.p	[numeric(1)] For opt = "ea": Probability of 1-point crossover, see <a href="#">sbx_operator</a> . Default is 0.5.
opt.ea.pm.eta	[numeric(1)] For opt = "ea": Distance parameter of mutation distribution, see <a href="#">pm_operator</a> . Default is 15.
opt.ea.pm.p	[numeric(1)] For opt = "ea": Probability of 1-point mutation, see <a href="#">pm_operator</a> . Default is 0.5.
opt.ea.lambda	[numeric{1}] For opt.ea = "ea". Number of children generated in each generation. Default is 1.
opt.nsga2.popsiz	[numeric{1}] For opt.multiobj.method = "nsga2". Population size of nsga2. Default is 100.
opt.nsga2.generations	[numeric{1}] For opt.multiobj.method = "nsga2". Number of populations for of nsga2. Default is 50.
opt.nsga2.cprob	[numeric{1}] For opt.multiobj.method = "nsga2". nsga2 param. Default is 0.7.
opt.nsga2.cdists	[numeric{1}] For opt.multiobj.method = "nsga2". nsga2 param. Default is 5.
opt.nsga2.mprob	[numeric{1}] For opt.multiobj.method = "nsga2". nsga2 param. Default is 0.2.
opt.nsga2.mdists	[numeric{1}] For opt.multiobj.method = "nsga2". nsga2 param. Default is 10.

**Value**

[MBOControl](#) .

**See Also**

Other MBOControl: [makeMBOControl](#), [setMBOControlMultiObj](#), [setMBOControlMultiPoint](#), [setMBOControlTermination](#)

---

setMBOControlMultiObj *Set multi-objective options.*

---

### Description

Extends MBO control object with multi-objective specific options.

### Usage

```
setMBOControlMultiObj(control, method = NULL, ref.point.method = NULL,
  ref.point.offset = NULL, ref.point.val = NULL, parego.s = NULL,
  parego.rho = NULL, parego.use.margin.points = NULL,
  parego.sample.more.weights = NULL, parego.normalize = NULL,
  dib.indicator = NULL, mspot.select.crit = NULL)
```

### Arguments

control	[MBOControl] Control object for mbo.
method	[character(1)] Which multi-objective method should be used? “parego”: The ParEGO algorithm. “dib”: Direct indicator-based method. Subsumes SMS-EGO and epsilon-EGO. “mspot”: Directly optimizes multicrit problem where we substitute the true objectives with model-based infill crits via an EMOA. All methods can also propose multiple points in parallel. Default is “dib”.
ref.point.method	[character(1)] Method for the determination of the reference point used for S-metric. Currently used for “mspot” and “dib” with indicator “sms”. Possible Values are: “all”: In each dimension: maximum of all points + ref.point.offset. “front”: In each dimension: maximum of all non-dominated points + ref.point.offset. “const”: Constant value, see ref.point.val. Default is “all”.
ref.point.offset	[numeric(1)] See ref.point.method, default is 1.
ref.point.val	[numeric] Constant value of reference point for hypervolume calculation. Used if ref.point.method = “const”. Has to be specified in this case.
parego.s	[integer(1)] Parameter of parego - controls the number of weighting vectors. The default depends on n.objectives and leads to ca. 100000 different possible weight vectors. The defaults for (2, 3, 4, 5, 6) dimensions are (100000, 450, 75, 37, 23) and 10 for higher dimensions.
parego.rho	[numeric(1)] Parameter of parego - factor for Tchebycheff function. Default 0.05 as suggested in parego paper.

parego.use.margin.points  
[logical]  
For each target function: Should the weight vector (0, ..., 0, 1, 0, ..., 0), i.e. the weight vector with only 0 and a single 1 at the i.th position for the i.th target function, be drawn with probability 1? Number of TRUE entries must be less or equal to propose.points Default is not to do this.

parego.sample.more.weights  
[integer(1)]  
In each iteration parego.sample.more.weights \* propose.points are sampled and the weights with maximum distance to each other are chosen. Default is 1, if only 1 point is proposed each iteration, otherwise 5.

parego.normalize  
[character]  
Normalization to use. Either map the whole image space to [0, 1] (standard, the default) or just the paretofront (front).

dib.indicator [character(1)]  
Either “sms” (SMS-EGO like algorithm) or “eps” (epsilon-EGO like algorithm). Default is “sms”.

mspot.select.crit  
[MBOInfillCrit]  
Which infill.crit to use in the candidate selection. After the NSGA2 proposed a set of candidates, “propose.points” are selected via the hypervolume contribution of this infill.crit. Possible values are “crit.mr” and “crit.cb” (or any other InfillCrit generated with [makeMBOInfillCritCB](#)), default is “crit.mr”.

## Value

[MBOControl](#) .

## References

For more information on the implemented multi-objective procedures the following sources might be helpful: Knowles, J.: ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. IEEE Transactions on Evolutionary Computation, 10 (2006) 1, pp. 50-66

Wagner, T.; Emmerich, M.; Deutz, A.; Ponweiser, W.: On Expected- Improvement Criteria for Model-Based Multi-Objective Optimization. In: Proc. 11th Int. Conf. Parallel Problem Solving From Nature (PPSN XI) - Part I, Krakow, Poland, Schaefer, R.; Cotta, C.; Kolodziej, J.; Rudolph, G. (eds.), no. 6238 in Lecture Notes in Computer Science, Springer, Berlin, 2010, ISBN 978-3-642-15843-8, pp. 718-727, doi:10. 1007/978-3-642-15844-5 72

Wagner, T.: Planning and Multi-Objective Optimization of Manufacturing Processes by Means of Empirical Surrogate Models. No. 71 in Schriftenreihe des ISF, Vulkan Verlag, Essen, 2013, ISBN 978-3-8027-8775-1

Zaefferer, M.; Bartz-Beielstein, T.; Naujoks, B.; Wagner, T.; Emmerich, M.: A Case Study on Multi-Criteria Optimization of an Event Detection Software under Limited Budgets. In: Proc. 7th International. Conf. Evolutionary Multi-Criterion Optimization (EMO 2013), March 19-22, Sheffield, UK, R. Purshouse; P. J. Fleming; C. M. Fonseca; S. Greco; J. Shaw, eds., 2013, vol. 7811

of Lecture Notes in Computer Science, ISBN 978-3-642-37139-4, pp. 756770, doi:10.1007/978-3-642-37140-0 56

Jeong, S.; Obayashi, S.: Efficient global optimization (EGO) for Multi-Objective Problem and Data Mining. In: Proc. IEEE Congress on Evolutionary Computation (CEC 2005), Edinburgh, UK, Corne, D.; et al. (eds.), IEEE, 2005, ISBN 0-7803-9363-5, pp. 2138-2145

### See Also

Other MBOControl: [makeMBOControl](#), [setMBOControlInfill](#), [setMBOControlMultiPoint](#), [setMBOControlTermination](#)

---

setMBOControlMultiPoint

*Set multipoint proposal options.*

---

### Description

Extends an MBO control object with options for multipoint proposal.

### Usage

```
setMBOControlMultiPoint(control, method = NULL, cl.lie = NULL,
  moimbo.objective = NULL, moimbo.dist = NULL, moimbo.selection = NULL,
  moimbo.maxit = NULL, moimbo.sbx.eta = NULL, moimbo.sbx.p = NULL,
  moimbo.pm.eta = NULL, moimbo.pm.p = NULL)
```

### Arguments

control	<a href="#">[MBOControl]</a> Control object for mbo.
method	<a href="#">[character(1)]</a> Method used for proposal of multiple infill points, for parallel batch evaluation. Possible values are: "cb": Proposes points by optimizing the confidence bound "cb" criterion, propose.points times. Each lambda value for "cb" is drawn randomly from an exp(1)-distribution, so do not define infill.opt.cb.lambda. The optimizer for each proposal is configured in the same way as for the single point case, i. e., by specifying infill.opt and related stuff. "moimbo": Proposes points by multi-objective infill criteria via evolutionary multi-objective optimization. The EA is a (mu+1) type of algorithm and runs for moimbo.maxit generations. The population size is set to propose.points. The selection criterion is moimbo.selection. If this method is selected the infill criterion in setMBOInfillCrit is ignored. "cl": Proposes points by constant liar strategy. Only meaningful if infill.crit == "cb" In the first step the kriging model is fitted based on the real data and the best point is calculated according to the regular EI-criterion. Then, the function value of the best point is simply guessed by the worst seen function evaluation. This lie is used to update the model in order to propose the subsequent point. The procedure is applied until the number of best points achieves propose.points. Default is cb.

cl.lie	[function] Function used by constant liar method for lying. Default is min.
moimbo.objective	[character(1)] Objectives which are optimized in multi-objective approach. Possible values are: “mean.dist”, “ei.dist”, “mean.se”, “mean.se.dist”. Default is “ei.dist”.
moimbo.dist	[character(1)] Distance function used in multi-objective EA. Possible values are: “nearest.neighbor”, “nearest.better”. Default is “nearest.better”.
moimbo.selection	[character(1)] Method used for selecting 1 element for removal from the population in each iteration of the multi-objective EA. Possible values are: “hypervolume”: Non-dominated sorting + hypervolume contribution. “crowdingdist”: Non-dominated sorting + crowding distance based ranking. “first”: Non-dominated sorting + first objective of moimbo.objective as criterion. “last”: Non-dominated sorting + last objective of moimbo.objective as criterion. Default is hypervolume.
moimbo.maxit	[character(1)] Number of generations for multi-objective EA. Default is 100.
moimbo.sbx.eta	[numeric(1)] Distance parameter of crossover distribution, see <a href="#">sbx_operator</a> . Default is 15.
moimbo.sbx.p	[numeric(1)] Probability of 1-point crossover, see <a href="#">sbx_operator</a> . Default is 1.
moimbo.pm.eta	[numeric(1)] Distance parameter of mutation distribution, see <a href="#">pm_operator</a> . Default is 15.
moimbo.pm.p	[numeric(1)] Probability of 1-point mutation, see <a href="#">pm_operator</a> . Default is 1.

**Value**

[MBOControl](#) .

**See Also**

Other MBOControl: [makeMBOControl](#), [setMBOControlInfill](#), [setMBOControlMultiObj](#), [setMBOControlTermination](#)

---

setMBOControlTermination

*Set termination options.*

---

**Description**

Extends an MBO control object with infill criteria and infill optimizer options.

**Usage**

```
setMBOControlTermination(control, iters = NULL, time.budget = NULL,
  exec.time.budget = NULL, target.fun.value = NULL, max.evals = NULL,
  more.termination.conds = list())
```

**Arguments**

control	[ <a href="#">MBOControl</a> ] Control object for mbo.
iters	[integer(1)] Number of sequential optimization steps.
time.budget	[integer(1)   NULL] Running time budget in seconds. Note that the actual mbo run can take more time since the condition is checked after each iteration. The default NULL means: There is no time budget.
exec.time.budget	[integer(1)   NULL] Execution time (time spent executing the function passed to mbo) budget in seconds. Note that the actual mbo run can take more time since the condition is checked after each iteration. The default NULL means: There is no execution time budget.
target.fun.value	[numeric(1)]   NULL] Termination criterion for single-objective optimization: Stop if a function evaluation is better than this given target.value. The default NULL means: The function value won't be taken into account for termination.
max.evals	[integer(1)   NULL] Maximal number of function evaluations. The default NULL means: The total number of evaluations won't be taken into account for termination.
more.termination.conds	[list] Optional list of termination conditions. Each condition needs to be a function of a single argument opt.state of type <a href="#">OptState</a> and should return a list with the following elements:  <b>term</b> [logical(1) ] Logical value indicating whether the termination condition is met. <b>message</b> [character(1) ] Termination message. At the moment we just allow term.custom.

**Value**

[MBOControl](#) .

**See Also**

Other MBOControl: [makeMBOControl](#), [setMBOControlInfill](#), [setMBOControlMultiObj](#), [setMBOControlMultiPoint](#)



## Examples

```
fn = smooof::makeSphereFunction(1L)
ctrl = makeMBOControl()

# custom termination condition (stop if target function value reached)
# We neglect the optimization direction (min/max) in this example.
yTargetValueTerminator = function(y.val) {
  force(y.val)
  function(opt.state) {
    opt.path = opt.state$opt.path
    current.best = getOptPathEl(opt.path, getOptPathBestIndex((opt.path)))$y
    term = (current.best <= y.val)
    message = if (!term) NA_character_ else sprintf("Target function value %f reached.", y.val)
    return(list(term = term, message = message))
  }
}

# assign custom termination condition
ctrl = setMBOControlTermination(ctrl, more.termination.conds = list(yTargetValueTerminator(0.05)))
res = mbo(fn, control = ctrl)
print(res)
```

trafos

*Transformation methods.*

## Description

- **logTrafo**  
Natural logarithm.
- **sqrtTrafo**  
Square root.

If negative values occur and the trafo function can handle only positive values, a shift of the form  $x - \min(x) + 1$  is performed prior to the transformation if the argument `handle.violations` is set to “warn” which is the default value.

## Usage

```
trafoLog(base = 10, handle.violations = "warn")
```

```
trafoSqrt(handle.violations = "warn")
```

## Arguments

base	[numeric(1)] The base with respect to which logarithms are computed. Default is 10.
handle.violations	[character(1)] What should be done, if negative values occur? Setting this option to “warn”, which is the default, shifts the function. “error” stops the process immediately.

**Format**

None

# Index

cmaes, [34](#)  
configureMlr, [15](#)  
crit.aei (makeMBOInfillCrit), [13](#)  
crit.cb (makeMBOInfillCrit), [13](#)  
crit.cb1 (makeMBOInfillCrit), [13](#)  
crit.cb2 (makeMBOInfillCrit), [13](#)  
crit.dib1 (makeMBOInfillCrit), [13](#)  
crit.ei (makeMBOInfillCrit), [13](#)  
crit.eq (makeMBOInfillCrit), [13](#)  
crit.mr (makeMBOInfillCrit), [13](#)  
crit.se (makeMBOInfillCrit), [13](#)  
  
error\_handling, [3](#)  
exampleRun, [4](#), [21](#)  
exampleRunMultiObj, [5](#)  
  
generateDesign, [5](#), [6](#), [17](#)  
generateGridDesign, [5](#), [6](#), [17](#)  
generateRandomDesign, [5](#), [6](#), [17](#)  
getGlobalOpt, [7](#)  
getMBOInfillCrit, [7](#)  
getMBOInfillCritComponents  
    (getMBOInfillCrit), [7](#)  
getMBOInfillCritId (getMBOInfillCrit), [7](#)  
getMBOInfillCritName  
    (getMBOInfillCrit), [7](#)  
getMBOInfillCritParam  
    (getMBOInfillCrit), [7](#)  
getMBOInfillCritParams  
    (getMBOInfillCrit), [7](#)  
getSupportedInfillOptFunctions, [8](#)  
getSupportedMultipointInfillOptFunctions,  
    [8](#)  
  
hasRequiresInfillCritStandardError  
    (getMBOInfillCrit), [7](#)  
  
infillcrits, [9](#), [14](#), [33](#)  
initCrit, [10](#)  
  
Learner, [5](#), [6](#), [10](#), [17](#)  
  
makeMBOControl, [4](#), [11](#), [20](#), [35](#), [38–40](#)  
makeMBOInfillCrit, [9](#), [10](#), [13](#), [33](#)  
makeMBOInfillCritAEI (infillcrits), [9](#)  
makeMBOInfillCritCB, [33](#), [37](#)  
makeMBOInfillCritCB (infillcrits), [9](#)  
makeMBOInfillCritDIB (infillcrits), [9](#)  
makeMBOInfillCritEI (infillcrits), [9](#)  
makeMBOInfillCritEQI (infillcrits), [9](#)  
makeMBOInfillCritMeanResponse  
    (infillcrits), [9](#)  
makeMBOInfillCritStandardError  
    (infillcrits), [9](#)  
makeMBOLearner, [14](#)  
makeMBOTrafoFunction, [16](#)  
makeMultiObjectiveFunction, [4](#), [6](#), [17](#)  
makeSingleObjectiveFunction, [4](#), [6](#), [17](#)  
maximinLHS, [5](#), [6](#), [17](#)  
mbo, [4](#), [15](#), [16](#), [18](#), [19](#)  
mbo\_default\_learner, [5](#), [6](#), [17](#)  
mbo\_default\_learner (makeMBOLearner), [14](#)  
mbo\_OptPath, [19](#), [20](#)  
mbo\_parallel, [16](#), [21](#)  
mboContinue, [4](#), [12](#), [18](#)  
MBOControl, [5](#), [6](#), [10](#), [12](#), [15](#), [17–19](#), [31](#), [33](#),  
    [35–40](#)  
MBOControl (makeMBOControl), [11](#)  
mboFinalize, [18](#)  
MBOInfillCrit, [7](#), [10](#), [14](#), [33](#), [37](#)  
MBOInfillCrit (makeMBOInfillCrit), [13](#)  
MBOMultiObjResult, [17](#), [19](#)  
MBOSingleObjResult, [17](#), [19](#)  
Measure, [12](#)  
mlrMBO\_examples, [21](#)  
mse, [12](#)  
  
OptPath, [19](#), [20](#)  
OptProblem, [28](#), [28](#)  
OptResult, [28](#), [28](#)  
OptState, [28](#), [40](#)

`parallelMap`, [3](#)  
`plot.MBOMultiObjResult (plotMBOResult),  
    30  
plot.MBOSingleObjResult  
    (plotMBOResult), 30  
plotExampleRun, 4, 5, 29  
plotMBOResult, 30  
plotOptPath, 30, 31  
pm_operator, 35, 39  
print.MBOControl, 31  
  
regr.randomForest, 15  
renderExampleRunPlot, 4, 31  
ResampleDesc, 12  
ResampleResult, 19  
  
sbx_operator, 35, 39  
setMBOControlInfill, 3, 12, 33, 38–40  
setMBOControlMultiObj, 12, 35, 36, 39, 40  
setMBOControlMultiPoint, 12, 35, 38, 38,  
    40  
setMBOControlTermination, 12, 35, 38, 39,  
    39  
  
trafoLog (trafos), 41  
trafos, 16, 41  
trafoSqrt (trafos), 41  
  
WrappedModel, 13, 19, 20`