Package 'DoE.MIParray'

September 28, 2017

Type Package

Title Creation of Arrays by Mixed Integer Programming

Date 2017-09-26

Version 0.10

Author Ulrike Groemping

Maintainer Ulrike Groemping <groemping@beuth-hochschule.de>

Description 'CRAN' package 'DoE.base' and non-'CRAN' packages 'gurobi' and 'Rmosek' (newer version than that on 'CRAN') are enhanced with functionality for the creation of optimized arrays for experimentation, where optimization is in terms of generalized minimum aberration. It is also possible to optimally extend existing arrays to larger run size. Optimization requires the availability of at least one of the commercial products 'Gurobi' or 'Mosek' (free academic licenses available for both). For installing 'Gurobi' and its R package 'gurobi', follow instructions at <http://www.gurobi.com/downloads/gurobi-optimizer> and <http://www.gurobi.com/documentation/7.5/refman/r_api_overview.html>. For installing 'Mosek' and its R package 'Rmosek', follow instructions at <https://www.mosek.com/downloads/> and <http://docs.mosek.com/8.1/rmosek/install-interface.html>.

Imports stats, methods, DoE.base

Enhances gurobi, Rmosek (>= 8.0)

Suggests slam (>= 0.1-9), Matrix (>= 1.1.0)

License GPL (>= 2)

Encoding UTF-8

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2017-09-28 15:26:18 UTC

R topics documented:

DoE.MIParray-package   *Package to Create a MIP Based Array*

Description

The package creates an array with specified minimum resolution and optimized word length pattern, using mixed integer programming (MIP). It requires the presence of at least one of the commercial softwares Gurobi (free academic license available, R package gurobi included with the software) or Mosek (free academic license available, outdated CRAN package Rmosek must be updated with current version provided by vendor). Arrays can be created from scratch, or using a user-specified starting array, or extending an existing array.

Details

The package implements generalized minimum aberration (GMA) according to Xu and Wu (2001), which is a way to minimize the confounding potential in a factorial design, as measured by the generalized word length pattern (GWLP). Reduction of short words has priority over reduction of long words, because it pertains to confounding of lower order interactions, which is assumed to be more severe than confounding of higher order interactions.

There is always one word of length zero. An array is said to have strength t (resolution R=t+1), if it has no words of lengths 1,...,t, but has words of length R=t+1; in that case, t-factor interactions are not confounded with the overall mean, (t-1)-factor interactions are not confounded with main effects, and so forth. Groemping and Xu (2014) provided an interpretation of the number of shortest words (i.e. words of length R) in terms of coefficients of determination of linear models with main effects model matrix columns on the LHS and full factorial t-factor models on the right-hand side; for example, in an array of strength 2 with three 3-level factors, the number of words of length 3 is the sum of the two R^2 values obtained from regressing the two main effects model matrix columns of one of the factors on a full model in the other two factors, provided main effect model matrix columns are coded orthogonally (to each other and the overall mean). GMA considers a design better than another one, if it has larger strength or resolution. In case of ties, a design is better if it has fewer shortest words; further ties are resolved by comparing words of lengths t+2, t+3, ...

Any array found by a function of this package will have the requested resolution (if not possible, an error will be thrown), and optimization of the number of shortest words will be attempted. If kmax is chosen larger than the resolution, optimization of longer words will also be attempted. Mixed integer optimization is very resource-intensive and often fails to provide a confirmed optimum (see also below). Choosing kmax larger than the resolution is therefore advisable for very small problems only; in most cases, one should attempt an optimization of the number of shortest words only. Only after this has been achieved, possibly with several sequential attempts, a subsequent attempt to improve the number of longer words should be undertaken.

Functions [gurobi_MIParray](#) and [mosek_MIParray](#) create an array from scratch (start=NULL and forced=NULL), by trying to improve a starting array specified with the argument start, or by

restricting a portion of the array to correspond to a pre-existing array with the argument `forced`. If no starting array is given, the functions initially obtain one by solving a linear optimization problem for obtaining a design with the requested resolution. Subsequently, the number of shortest words of the starting array is optimized, followed by the numbers of words of length up to `kmax`.

The strategy used in the package is a modification of Fontana (2017). Modifications include enforcing the requested resolution via a linear optimization step (much faster than sequentially optimizing all those word lenths until they are zeroes), and using a result on coding invariance from Groemping (in press) for a more parsimonious formulation of constraints. Mixed integer programming can use a lot of time and resources; in particular, the confirmation of the optimality of a solution that has been found can take prohibitively long, even if the optimal solution itself has been found fast (which is also not necessarily so). Functions `gurobi_MIPcontinue` and `mosek_MIPcontinue` can be used to continue optimization for larger problems, where optimization was previously aborted, e.g. due to a time limit. Note that it is possible to continue an optimization effort started with Gurobi using Mosek and vice versa, because the functions `gurobi_MIPcontinue` and `mosek_MIPcontinue` can convert problems using the internal functions `mosek2gurobi` and `gurobi2mosek`.

Warning

Escaping from a Gurobi run will most likely be unsuccessful, might leave R in an unstable state, and usually fails to release the entire CPU usage; thus, one should think carefully about the affordable run times.

Mosek can usually be escaped using the <ESC> key, and one can even hope to get a valid output. However, after such escapes, it is also advisable to use **RMosek**'s internal clean function (`Rmosek:::mosek_clean()`), since computer instabilities after repeated escapes from Mosek have been observed (the package functions execute the `mosek_clean` command after conducting Mosek runs).

Note

The package is not meant for situations, for which a full factorial design would be huge; the mixed integer problem to be solved has at least `prod(nlevels)` binary or general integer variables and will likely be untractable, if this number is too large. (For extending an existing designs, since some variables are fixed, the limit moves out a bit.)

Author(s)

Ulrike Groemping

References

Fontana, R. (2017). Generalized Minimum Aberration mixed-level orthogonal arrays: a general approach based on sequential integer quadratically constrained quadratic programming. *Communications in Statistics – Theory Methods* **46**, 4275-4284.

Groemping, U. and Xu, H. (2014). Generalized resolution for orthogonal arrays. *The Annals of Statistics* **42**, 918-939.

Groemping, U. (in press). Coding Invariance in Factorial Linear Models and a New Tool for Assessing Combinatorial Equivalence of Factorial Designs. *Journal of Statistical Planning and Inference*.

Gurobi Optimization Inc. (2017). Gurobi Optimizer Reference Manual. http://www.gurobi.com/documentation/.

Mosek ApS (2017a). MOSEK version w.x.y.z documentation. Accessible at: https://www.mosek.com/documentation/. This package has been developed using version 8.1.0.23 (accessed August 29 2017).

Mosek ApS (2017b). MOSEK Rmosek Package 8.1.y.z. http://docs.mosek.com/8.1/rmosek/index.html. *!!! In normal R speak, this is the documentation of the Rmosek package version 8.0.69 (or whatever comes next), when applied on top of the Mosek version 8.1.y.z (this package has been devoloped with Mosek version 8.1.0.23 and will likely not work for Mosek versions before 8.1). !!! (accessed August 29 2017)*

Xu, H. and Wu, C.F.J. (2001). Generalized minimum aberration for asymmetrical fractional factorial designs. *Annals of Statistics* **29**, 549-560.

See Also

See also as gurobi_MIParray, mosek_MIParray, gurobi_MIPcontinue and mosek_MIPcontinue. Furthermore, the following links from package **DoE.base** are of interest: show.oas queries catalogued orthogonal arrays and oa_feasible carries out feasibility checks for runs, levels and strength requirements.

Examples

```
## Not run:
## ideal sequence of optimization problems
## shown here for Mosek,
## for Gurobi analogous, if necessary increasing maxtime to e.g. 600 or 3600 or ...

## very small problem
plan <- mosek_MIParray(16, rep(2,6), resolution=4, kmax=6)

## an example approach for a larger problem
## optimize shortest word length
plan3 <- mosek_MIParray(24, c(2,4,3,2,2,2,2), resolution=3, maxtime=20)
## feasible solution was found, no confirmed optimum, 7/3 words of length 3
## try to optimize further or confirm optimality (improve=TRUE does this),
##                      give it 10 minutes
plan3b <- mosek_MIPcontinue(plan3, improve=TRUE, maxtime=600)
##     no improvement has been found, and the gap is still very large
##     (the time limit makes the result non-deterministic, of course,
##      because it depends on the computer's power and availability of its resources)

## For large problems, it cannot be expected that a *confirmed* optimum is found.
## Of course, one can put more effort into the optimization, e.g. by running overnight.
## It is also advisable to compare the outcome to other ways for obtaining a good array,
##     e.g. function oa.design from package DoE.base with optimized column allocation.
require(DoE.base)
show.oas(nruns=24, nlevels=c(2,4,3,2,2,2,2), show=Inf)
GWLP(plan_oad <- oa.design(nruns=24, nlevels=c(2,4,3,2,2,2,2), col="min34"))
## here, plan3b has a better A3 than plan_oad
```

```
## one might also try to confirm optimality by switching to the other optimizer
plan3c <- gurobi_MIPcontinue(plan3b, improve=TRUE, maxtime=600, MIPFocus=3)
   ## focus on improved bound with option MIPFocus
   ## still same value with very large gap after running this
   ## thus, now assume this as best practically feasible value

## one might now try to improve words of length 4 (improve=FALSE turns to the next word length)
plan4 <- mosek_MIPcontinue(plan3b, improve=FALSE, maxtime=600)
   ## this does not yield any improvement
   ## working on longer words is not considered worthwhile
   ## thus, plan3 or plan3b are used for pragmatic reasons,
   ## without confirmed optimality

## End(Not run)
```

---

lowerbound_AR	*Function to Calculate a Lower Bound for A_R and Internal Auxiliary Functions*

Description

The functions serve the calculation of lower bounds for the worst case confounding. lowerbound_AR is intended for direct use, lowerbounds and lowerbound_chi2 are internal functions.

Usage

```
lowerbound_AR(nruns, nlevels, R, crit = "total")
lowerbounds(nruns, nlevels, R)
lowerbound_chi2(nruns, nlevels)
```

Arguments

nruns	positive integer, the number of runs
nlevels	vector of positive integers, the numbers of levels for the factors
R	positive integer, the resolution of the design
crit	"total" or "worst"; if "total", a bound for the overall A_R (sum of the results from lowerbounds) is calculated; otherwise, a bound for the largest individual contribution from an R factor set is calculated

Details

Function lowerbounds provides (integral) bounds on $n^2 A_R$ (with n=nruns) according to Groemping and Xu (2014) Theorem 5 for all R factor sets. If the number of runs permits a design with resolution larger than R, the value(s) will be 0. For resolution at least III, the result of function lowerbound_AR is the sum (crit="total") or maximum (crit="worst") of these individual bounds, divided by the square of the number of runs.

For resolution II and crit="total", function lowerbound_chi2 implements the lower bound *B* on chi^2 which was provided in Lemma 2 of Liu and Lin (2009). For supersaturated resolution

II designs, this bound is is usually sharper than the one obtained on the basis of Grömping and Xu (2014). Due to the relation between A_2 and χ^2 that is stated in Groemping (2017) (summands of A_2 are an *n*th of a χ^2, with *n*=nruns), this bound can be easily transformed into a bound for A_2; this relation is also used to slightly sharpen the bound *B* itself: $n^2 \cdot A_2$ must be integral, which implies that *B* can be replaced by `ceiling(nruns*B)/nruns`, which is applied in function `lowerbound_chi2`. Function `lowerbound_AR` increases the lower bound on A_2 accordingly, if `lowerbound_chi2` provides a sharper bound than the sum of the elements returned by functioni `lowerbounds`.

Value

`lowerbound_AR` returns a lower bound for the number of words of length R (either total or worst case),
`lowerbounds` returns a vector of lower bounds for individual R factor sets on a different scale (division by `nruns^2` needed for transforming this into the contributions to words of length R),
and function `lowerbound_chi2` returns a lower bound on the χ^2 value which can be used as a quality criterion for supersaturated designs.

Author(s)

Ulrike Groemping

References

Groemping, U. and Xu, H. (2014). Generalized resolution for orthogonal arrays. *The Annals of Statistics* **42**, 918-939.

Groemping, U. (2017). Frequency tables for the coding-invariant quality assessment of factorial designs. *IISE Transactions* **49**, 505-517.

Liu, M.Q. and Lin, D.K.J. (2009). Construction of Optimal Mixed-Level Supersaturated Designs. *Statistica Sinica* **19**, 197-211.

See Also

See also oa_feasible.

Examples

```
lowerbound_AR(24, c(2,3,4,6),2)
```

| mosek2gurobi | *Functions to recast quadratically constrained MIP in different format* |

Description

The functions recast a Mosek model into Gurobi format and vice versa, for use with objects of class qco from package DoE.MIParray.

Usage

```
mosek2gurobi(qco, ...)
gurobi2mosek(qco, ...)
```

Arguments

qco	a mixed integer optimization problem of class qco, as generated from package **DoE.MIParray**
...	not used so far

Details

The functions only treat the special cases treated in package DoE.MIParray, (minimization problem, all linear constraints are equality constraints, cones are created in the form done in that package).

Value

an object of S3 class qco

Author(s)

Ulrike Groemping

See Also

See also as `mosek_MIParray`, `gurobi_MIParray`.

---

mosek_MIParray   *Functions to Create a MIP Based Array Using Gurobi or Mosek*

Description

The functions create an array with specified minimum resolution and optimized word length pattern based on mixed integer programming with the commercial software Gurobi (free academic license available) or Mosek (free academic license available). Creation is done from scratch, or using a user-specified starting value, or extending an existing array. Important: Installation of Gurobi and/of Mosek as well as the corresponding R packages is necessary. The R package gurobi comes with the software, the current version of the R package Rmosek has to be obtained from vendor's website (CRAN version is outdated!).

Usage

```
mosek_MIParray(nruns, nlevels, resolution = 3, kmax = max(resolution, 2),
   distinct = TRUE, detailed = 0, start=NULL, forced=NULL,
   maxtime = Inf, nthread=2, mosek.opts = list(verbose = 10, soldetail = 1),
  mosek.params = list(dparam = list(LOWER_OBJ_CUT = 0.5, MIO_TOL_ABS_GAP = 0.2,
      INTPNT_CO_TOL_PFEAS = 1e-05, INTPNT_CO_TOL_INFEAS = 1e-07),
      iparam = list(PRESOLVE_LINDEP_USE="OFF", LOG_MIO_FREQ=100)))
gurobi_MIParray(nruns, nlevels, resolution = 3, kmax = max(resolution, 2),
  distinct = TRUE, detailed = 0, start=NULL, forced=NULL,
  maxtime = 60, nthread = 2, heurist=0.5, MIQCPMethod=0, MIPFocus=1,
  gurobi.params = list(BestObjStop = 0.5, LogFile=""))
```

Arguments

nruns	positive integer; number of runs
nlevels	vector of integers (>=2); numbers of factor levels
resolution	positive integer; the minimum resolution requested
kmax	integer, kmax >= resolution and kmax >= 2 are required; the largest number of words to be optimized (default: kmax = resolution)
distinct	logical; if TRUE (default), restricts counting vector to 0/1 entries, which means that the resulting array is requested to have distinct rows; otherwise, duplicate rows are permitted, i.e. the counting vector can have arbitrary non-negative integers. Designs with distinct runs are usually better; in addition, binary variables are easier to handle by the optimization algorithm. Nevertheless, there are occasions where a better array is found faster with option distinct=FALSE, even if it has distinct rows.
detailed	integer (default 0); determines the output detail: positive values imply inclusion of a problem and solution history (attribute history), values of at least 3 add the lists of optimization matrices (Us and Hs, attribute matrices).
start	for resolution > 1 only; a starting value for the algorithm: can be a array matrix with entries 1 to number of levels for each column, or a counting vector for the full factorial in lexicographic order; if specified, start must specify an array with the appropriate number of rows and columns, the requested resolution and, if distinct = TRUE, also contain distinct rows (matrix) or 0/1 elements only.
forced	for resolution > 1 only; runs to force into the solution design; can be given as an array matrix with the appropriate number of columns and less than nruns rows or a counting vector for the full factorial in lexicographic order with sum smaller than nruns; if distinct=TRUE, forced must have distinct rows (matrix) or 0/1 elements only.
maxtime	the maximum run time in seconds per Gurobi or Mosek optimization request (the overall run time may become (much) larger); in case of conflict between maxtime and an explicit timing request in gurobi.params$TimeLimit or mosek.params$dparam$$MIO_I the stricter request prevails; the default values differ between Gurobi (60) and Mosek (Inf), because Mosek runs can be easily escaped, while Gurobi runs cannot.

nthread	the number of threads (=cores) to use; there are also the Mosek parameter NUM_THREADS and the Gurobi parameter Threads; in case of conflict, the smaller request prevails. For using Gurobi's or Mosek's default (which is in most cases the use of all available cores), choose nthread=0.
	CAUTION: nthread should not be chosen larger than the available number of cores. Gurobi warns that performance will deteriorate, but was observed to perform OK. For Mosek, performance will strongly deteriorate, and for extreme choices the R session might even crash (even for small problems)!
mosek.opts	list of Mosek options; these have to be looked up in Mosek documentation
mosek.params	list of mosek parameters, which can have the list-valued elements dparam, iparam and/or sparam; their use has to be looked up in the RMosek documentation. The arguments maxtime and nthread correspond to the dparam$MIO_MAX_TIME and iparam$NUM_THREADS specifications. Conflicts are resolved as stated in their documentation.
	The element dparam$LOWER_OBJ_CUT can be used to incorporate a best bound found in an earlier successless optimization attempt; per default, it is set to 0.5, since the target function can take on integer values only and cannot be negative.
	If a valid starting value is not accepted by Mosek, it may be worthwhile to increase dparam$INTPNT_CO_TOL_PFEAS.
	Note that a user specified mosek.params should always contain the specifications shown under Usage. Exceptions: LOWER_OBJ_CUT is always specified to be at least 0.5, i.e. this option can be safely omitted without loosing anything, and intentional changes can of course be made.
heurist	the proportion heuristics time used by Gurobi in quadratic objective optimization (default 0.5; Gurobi default is 0.05); there is also the Gurobi parameter Heuristics; in case of conflict, the larger request prevails; the setting for heurist is deactivated for the initial linear problem which is always run with the Gurobi default. It can be worthwhile playing with this option for improving the run time for certain settings; for example, with nruns=48 and nlevels=c(2,2,3,4,4), heurist=0.05 performs better than the default 0.5.
MIQCPMethod	the method used by Gurobi for quadratically constrained optimization (default 0; other possibilities -1 (Gurobi decides) or 1); there is also the Gurobi parameter MIQCPMethod; in case of conflict, the method is set to "0"; this choice is made because it proved beneficial in many cases explored (although there also were a few cases which fared better with Gurobi's default).
MIPFocus	the strategy used by Gurobi for quadratically constrained optimization (default 1: focus on finding good feasible solutions fast; other possibilities: 0 (Gurobi decides/compromise), 2 or 3 (focus on increasing the lower bound fast)); there is also the Gurobi parameter MIPFocus; in case of conflict, MIPFocus is set to "0"; the setting for MIPFocus is deactivated for the initial linear problem which is always run with the Gurobi default.
gurobi.params	list of gurobi parameters; these have to be looked up in Gurobi documentation; the arguments maxtime, heurist, MIQCPMethod and MIPFocus refer to the Gurobi parameters "TimeLimit", "Heuristics", "MIQCPMethod" and "MIPFocus", respectively. See their documentation for what happens in case of conflict.
	The Gurobi parameter BestObjStop can be used to incorporate a best bound found in an earlier successless optimization attempt; per default, it is set to 0.5,

since the objective function can take on integer values only and cannot be nega-
tive.

Details

The functions initially solve a linear optimization problem for obtaining a design with the requested
resolution (if a start value is provided, this step is skipped). Subsequently, the number of shortest
words is optimized, followed by the numbers of words of length up to kmax. The argument forced
allows to specify an existing array that is to be extended (e.g. to double or triple size; extension by
a small number of runs will usually not be possible) in an optimized way.

For the optimization of the number of shortest words, the functions automatically enforce the bound
reported in Groemping and Xu (2014, Theorem 5), whenever it is larger than the prespecified bound.
If an array of weak strength R exists (with R the resolution), this will eliminate the need to prove
optimality the hard way.

For all but very small problems, it is likely advisable to choose kmax equal to the requested resolu-
tion (the default for resolution >= 2), and to proceed to longer words only after it has been made sure
that the shortest word length has been optimized (as far as possible with reasonable effort). Further
improvements can be attempted by applying gurobi_MIPcontinue or mosek_MIPcontinue to the
result object returned by the function. Note that it is possible to switch from using Mosek to us-
ing Gurobi or vice versa. An example for an optimization sequence can be found in the package
overview at DoE.MIParray.

In case of long run times, escaping from the gurobi run will most likely be unsuccessful and might
even leave R in an unstable state; thus, one should think carefully about the affordable run times. On
the contrary, it should usually be doable to escape a Mosek run; the remaining code of the function
will still be executed and will return the final state; for large problems, escaping a Mosek run can
also fail; if the console does not show the message "Releasing MOSEK environment" after escaping
a Mosek run, it is advised to execute the command Rmosek:::mosek_clean() afterwards.

Besides the run time, the number of threads is a very important resource control parameter. The
default assumes that the user wants to use two of the computer's multiple cores. For using Gurobi's
or Mosek's default (which is in most cases the use of all available cores), choose nthread=0.

Note that the optimizers' performance may depend on the order among the elements of vector
nlevels; while the problem itself and its global optimum are of course independent of this order,
the starting performance and local behavior, and consequently the speed with which an optimum
or at least a good value is attained, may sometimes strongly depend on that order. Thus, it may be
worthwhile to try a different order, if the outcome from a particular order is not satisfactory.

The default Gurobi parameters have been chosen after systematic experimentation with a limited
set of scenarii und Gurobi version 7.5.1. The choice of MIQCPMethod=0 was instrumental in many
cases; changing it to -1 may occasionally be tried. The MIPFocus parameter also appears beneficial
in many cases; changing it to 0 (leave choice to Gurobi) can be an option; it was slightly better than
choice 1 for mixed level designs with relatively small run sizes, while choice "1" was substantially
better for the other cases. The heuristics proportion has been chosen as 0.5, because this choice
seemed the best compromise for the situations considered. Note, however, that these parameters
deteriorate performance for very simple cases, e.g. the test cases of Fontana (2017). For such
cases, using MIQCPMethod=-1, MIPFocus=0 and heurist=0.05 will be preferrable; the defaults
were chosen in this way, since doubling or even tripling very short run times was decided to be less
detrimental than making more difficult problems completely intractable.

For Gurobi, several optimization parameters are switched off for the initial linear optimization step: the parameters Heuristics and MIPFocus are reset to their defaults (0.05 and 0).

Gurobi always stores the file "gurobi.log" in the working directory; even if storage of the log is suppressed with the default option LogFile="" or directed to another location by specifying a path, the default file "gurobi.log" is created and filled with a small amount of content. Thus, make sure to use a different file name when intentionally storing some log.

For Mosek, storing log output can be accomplished by directing the printed output to a suitable storing location. Note that the setting iparam$LOG_MIO_FREQ = 100 reduces the frequency of printing a log line for branch-and-cut optimization by the factor 10 versus the default. Parallelization in Mosek is not well-protected against interference from screen activity (for example). Thus, one should switch off logging to screen or otherwise, when working with many (all available) threads in parallel (LOG=0 instead of LOG_MIO_FREQ = 100 in the list iparam).

Value

an array of class oa, possibly with the following attributes: MIPinfo, which is either an object of class qco or a simple list with information (which would be the info element of the object of class qco in case the last optimization was not successful), history as a list of problem and solution lists, and matrices as a list of matrix lists. Presence or absence of history and matrices is controlled by option detailed, while MIPInfo is present if the optimization can be potentially improved by improving the last step (stop because of time limit and not because of optimal value) or by improving the number of longer words.

Installation

Gurobi and Mosek need to be separately installed; please follow vendors' instructions; it is necessary to obtain a license; for academic use, free academic licenses are available in both cases.

Both Gurobi and Mosek provide R packages (**gurobi** and **Rmosek**) for accessing the software, and they also provide instructions on their installation.

For Gurobi, the R package **gurobi** is provided with the software installation files and can simply be installed like usual for a non-Web package.

For Mosek, there is an outdated R package on CRAN, but it is necessary to obtain a current version of **Rmosek** from the Mosek website. Its installation requires compilation from source. Several prerequisites are needed, basically the same ones needed for compiling packages. Make sure to set up the R environment for this purpose (for Windows, see https://cran.r-project.org/bin/windows/Rtools/; you may have to set some paths yourself; I am not familiar with the proper process for other platforms). If this is accomplished and package **Matrix** is at the latest level (as recommended in the **Rmosek** online documentation), running the install.packages command provided in the **Rmosek** online documentation (Mosek ApS 2017b) will smoothly install the package. Don't be confused by Mosek ApS's strange communication regarding **Rmosek** version numbers; they are currently not kept in sync with Mosek version numbers, although the documentation seems to suggest so: the current version (August 29 2017) of package Rmosek is 8.0.69, while the Mosek version is 8.1.0.23 (the revision versions of Mosek change quite frequently). Whenever Mosek itself is re-installed (at least with a change of minor version like 8.0 to 8.1), the **Rmosek** sources must be re-compiled (e.g. with the above-mentioned install.packages command), even if their version has not changed; this is presumably why Mosek ApS speak of a version number for

the binary that corresponds to the Mosek version number, while the sources keep their version number (somewhat misleading for the R community). From within R, version numbers can be queried by packageVersion("Rmosek") and Rmosek::mosek_version(), respectively.

Note

The functions are not meant for situations, for which a full factorial design would be huge; the mixed integer problem to be solved has at least prod(nlevels) binary or general integer variables and will likely be untractable, if this number is too large. (For extending an existing designs, since some variables are fixed, the limit moves out a bit.)

Please be aware that escaping a Gurobi run will be not unlikely to leave the computer in an unstable situation. If function gurobi_MIParray is successfully interrupted by the <ESC> key or <Ctrl>-<C>, it will usually be necessary to restart R in order to free all CPU usage.

If a Mosek run is interrupted by the <ESC> key, it is advised to execute the command Rmosek:::mosek_clean() afterwards; this may help prevent problems from unclean closes of mosek runs.

Author(s)

Ulrike Groemping

References

Fontana, R. (2017). Generalized Minimum Aberration mixed-level orthogonal arrays: a general approach based on sequential integer quadratically constrained quadratic programming. *Communications in Statistics – Theory Methods* **46**, 4275-4284.

Groemping, U. and Xu, H. (2014). Generalized resolution for orthogonal arrays. *The Annals of Statistics* **42**, 918-939.

Gurobi Optimization Inc. (2017). Gurobi Optimizer Reference Manual. http://www.gurobi.com/documentation/.

Mosek ApS (2017a). MOSEK version w.x.y.z documentation. Accessible at: https://www.mosek.com/documentation/. This package has been developed using version 8.1.0.23 (accessed August 29 2017).

Mosek ApS (2017b). MOSEK Rmosek Package 8.1.y.z. http://docs.mosek.com/8.1/rmosek/index.html. *!!! In normal R speak, this is the documentation of the Rmosek package version 8.0.69 (or whatever comes next), when applied on top of the Mosek version 8.1.y.z (this package has been devoloped with Mosek version 8.1.0.23 and will likely not work for Mosek versions before 8.1). !!! (accessed August 29 2017)*

See Also

See also mosek_MIPcontinue and gurobi_MIPcontinue for continuing an uncompleted optimization, show.oas from package **DoE.base** for catalogued orthogonal arrays, and oa_feasible from package **DoE.base** for checking feasibility of requested array strength (resolution - 1) for combinations of nruns and nlevels.

Examples

```
## Not run:
## can also be run with gurobi_MIParray instead of mosek_MIParray
## there are of course better ways to obtain good arrays for these parameters
## (e.g. function FrF2 from package FrF2)
feld <- mosek_MIParray(16, rep(2,7), resolution=3, kmax=4)
feld
names(attributes(feld))
attr(feld, "MIPinfo")$info

## using a start value
start <- DoE.base::L16.2.8.8.1[,1:5]
feld <- mosek_MIParray(16, rep(2,5), resolution=4, start=start)

## counting vector representation of the start value could also be used
DoE.MIParray:::dToCount(start-1)
    ## "-1", because the function requires values starting with 0
    ## 32 elements for the full factorial in lexicographic order, 16 ones for the runs

## extending an existing array
force <- matrix(as.numeric(as.matrix(DoE.base::undesign(DoE.base::oa.design(L8.2.7)))), nrow=8)
feld <- mosek_MIParray(16, rep(2,7), resolution=3, kmax=4, forced=force)
attr(feld, "MIPinfo")$info

## End(Not run)
```

---

mosek_MIPcontinue   *Functions to Continue Optimization from Stored State*

Description

These functions continue optimization for a MIP-based array from a stored state.

Usage

```
mosek_MIPcontinue(qco, improve = TRUE, maxtime = Inf, nthread = 2,
      mosek.opts = list(verbose = 10, soldetail = 1),
      mosek.params = list(dparam = list(LOWER_OBJ_CUT = 0.5,
      MIO_TOL_ABS_GAP = 0.2, INTPNT_CO_TOL_PFEAS = 1e-05,
      INTPNT_CO_TOL_INFEAS = 1e-07),
      iparam = list(PRESOLVE_LINDEP_USE="OFF", LOG_MIO_FREQ=100)))
gurobi_MIPcontinue(qco, improve = TRUE, maxtime = 60, nthread = 2,
      heurist = 0.05, MIQCPMethod = 0, MIPFocus = 0,
      gurobi.params =list(BestObjStop = 0.5, LogFile=""))
```

Arguments

qco   object of class qco, created by a function in the package; or object of class oa
               that has a qco object as its MIPinfo attribute

improve	logical; if TRUE (default), try to improve the already obtained solution for word length `qco$info$last.k`, otherwise optimize the next word length
maxtime	time in seconds for the optimization call; defaults differ for Mosek (Inf) and Gurobi (60), because a Mosek run can be easily escaped (<ESC>-key), contrary to a Gurobi run
nthread	number of cores to use (0=all cores) CAUTION: `nthread` should not exceed the available number of cores. Gurobi warns that performance might deteriorate. For Mosek, performance WILL strongly deteriorate, and for extreme choices the R session might even crash (even for small problems)!
heurist	for `gurobi_MIPcontinue` only: the percentage of time (number between 0 and 1) spent on heuristics
MIQCPMethod	for `gurobi_MIPcontinue` only: the choice of optimization method; the default "0" has been observed to be better than Gurobi's default for most cases (Gurobi version 7.5.1); "-1" leaves the choice to Gurobi, "1" chooses the other method
MIPFocus	for `gurobi_MIPcontinue` only: the choice of strategy; the default "0" leaves this choice to Gurobi; for finding better feasible solutions, "1" is recommended; for improving the speed of increasing the lower bound for eventually proving optimality, "3" can be tried
mosek.opts	mosek options
mosek.params	Mosek parameters
gurobi.params	Gurobi parameters

Details

Note that it is possible to continue optimization with Gurobi, if it was started with Mosek, and vice versa. The tool will transform the problem into the respective other format.

Usage of options is analogous to functions [mosek_MIParray](#) and [gurobi_MIParray](#), respectively, where these are described in more detail.

For some applications, usability of `mosek_MIPcontinue` is currently hampered by the fact that Mosek's presolve routines identify additional integer variables and fail to recognise user-specified starting values for these that are not exactly integer-valued. According to Mosek ApS, this is scheduled to be remedied with Mosek Version 9.

Value

an array of class "oa", if not optimized to GMA with info for further continuation (see documentation of [mosek_MIParray](#) or [gurobi_MIParray](#))

Author(s)

Ulrike Groemping

See Also

See also [DoE.MIParray](#) for examples of the role of the `MIPcontinue` functions, [mosek_MIParray](#) and [gurobi_MIParray](#) for more detail on the optimization arguments.

Index