Journal of
open research software

Grömping, U 2020 DoE.MIParray: An R Package for Algorithmic Creation of Orthogonal Arrays. *Journal of Open Research Software,* 8: 24. DOI: https://doi.org/10.5334/jors.286

SOFTWARE METAPAPER

DoE.MIParray: An R Package for Algorithmic Creation of Orthogonal Arrays

Ulrike Grömping
Department II, Beuth University of Applied Sciences, Berlin, DE
groemping@beuth-hochschule.de

The R package **DoE.MIParray** uses mixed integer optimization for creating well-balanced arrays for experimental designs. Its use requires availability of at least one of the commercial optimizers Gurobi or Mosek. Investing some effort into the creation of a suitable array is justified, because experimental runs are often very expensive, so that their information content should be maximized. **DoE.MIParray** is particularly useful for creating *relatively* small mixed level designs. Balance is optimized by applying the quality criterion "generalized minimum aberration" (GMA), which aims at minimizing confounding of low order effects in factorial models, without assuming a specific model. For relevant cases, **DoE.MIParray** exploits a lower bound on its objective function, which allows to drastically reduce the computational burden of mixed integer optimization.

Keywords: R; statistics; design of experiments; orthogonal arrays; mixed integer optimization; Mosek; Gurobi; well-balanced arrays; generalized minimum aberration
Funding statement: The work on this software is the outcome of research that was funded by Deutsche Forschungsgemeinschaft (grant GR 3843/2-1).

(1) Overview

Introduction

The R package **DoE.MIParray** creates well-balanced experimental designs for experimentation with a set of experimental factors, for which suitable off-the-shelf plans are not readily available. It is particularly useful for *relatively* small experimental situations for which the experimental factors have different numbers of levels (called "mixed level").

For a very basic example, consider an experiment on three brands of baking powder (B1, B2, B3), combined with two types of oven (electric versus gas) and two different recipes (old versus new). A combination of levels with which to conduct the experiment is called an "experimental run". For this simple example, there are 3·2·2 = 12 different level combinations (see **Table 1**). An experiment that runs all these once or replicates them in a balanced way is called a (replicated) full-factorial experiment; replication may be employed in order to draw stable conclusions in spite of random variation. (For experiments with many factors, explicit replication may be replaced by implicit replication, see e.g. [4]). The above small experiment only has the $m = 3$ factors bp (baking powder brand) with 3 levels, oven with 2 levels and recipe with 2 levels. For larger settings, a full factorial experiment will become infeasible fast; for example, Vasilev et al. ([11]) reported results from an experiment in 72 runs for $m = 7$ factors with numbers of levels 2, 2, 2, 2, 3, 3 and 4. A full factorial experiment would have needed 576 runs, which was considered infeasible. The design plan used in [11] was obtained using column selection optimization based on a catalogued 72 run plan of R package **DoE.base** (see [4]). Suitable use of R package **DoE.MIParray** creates a more balanced plan; thus, had **DoE.MIParray** already been available when conducting the experiment reported in [11], a more balanced experiment could have been run. This and other useful designs that have been found by package **DoE.MIParray** have also been reported in [5].

The benefit of balance

DoE.MIParray algorithmically creates a well-balanced array, using mixed integer optimization. The goal is to ensure that a factorial model with main effects and possibly low order interactions can be estimated with as little confounding as possible, without having to specify certain effects to be active or inactive or assuming a particular model. For discussing the benefit of balance, we have to formalize it a little. Imbalance is measured in terms of "generalized words", as introduced by Xu and Wu ([13]). For a design in m factors, the so-called generalized word length pattern (GWLP) is a tuple $(A_0, A_1, \ldots, A_m)$ of non-negative entries. The entry A_0 (for the overall mean) is always 1. In a full factorial design, all other entries are zeroes, i.e. $A_1, \ldots, A_m = 0$ (with m the number of factors, i.e. $m = 3$ in our example). The zeroes indicate that there is no imbalance in any degree of factorial effects (degree 1

for main effects, 2 for 2-factor interactions, and so forth, up to m-factor interactions). The resolution $R > 0$ of an experimental plan is the integer R for which $A_R > 0$ and $A_j = 0$ for all j with $0 < j < R$ (if any). Thus, requesting resolution III (resolution is usually given as a roman numeral), we would request that $A_1 = A_2 = 0$. This is also denoted as strength 2: the strength is one less than the resolution. For strength 2, all pairs (=2-tuples) of factors (when ignoring the presence of the other factors) are full factorials or balanced replications of full factorials. The entry A_j ($j = 1,...,m$) of the GWLP is the sum of the imbalance contributions from all j-tuples among the m factors.

As was mentioned above, the unreplicated full factorial design of **Table 1** with its $N = 12$ runs has the GWLP (A_0, A_1, A_2, A_3) = (1, 0, 0, 0). The three fractional factorial designs of **Table 2** with their $n = 6$ runs each have the GWLPs (1, 1/9, 2/9, 2/3), (1, 0, 7/9, 2/9), and (1, 0, 1/9, 8/9), respectively. In all three cases, the sum of the entries is $N/n = 12/6 = 2$, because the designs have $n = 6$ *distinct* runs out of the $N = 12$ possible runs of the full factorial. According to the generalized minimum aberration criterion, Design 1 is worst, since it has only resolution I (strength 0), because A_1 is already positive; this is caused by the imbalance in factor oven. Designs 2 and 3 have at least resolution II (strength 1), i.e. all factors are balanced (usually considered as the minimum requirement), while pairs of factors are not all balanced. Design 2 is worse than Design 3, because its A_2 entry is larger; this is due to the imbalance between factors oven and bp, which is not present in Design 3. One would usually strive to achieve at least resolution III; with this very small design, this is not possible in less than a full factorial. Whether or not a certain strength is possible can be checked using function `oa_feasible`, which tells us that strength 2 is not possible here (`oa_feasible(6, c(2,2,3), 2)`).

Mosaic plots are a good way to demonstrate imbalance (see e.g. [1]). **Figure 1** shows the worst case imbalance among triples of factors in the resolution III design from [11], compared to the worst case imbalance in the optimum design obtained using Mosek software via function `mosek_MIParray` of package **DoE.MIParray**; the latter has the smallest possible A_3 for 72 runs. The reference plots in the top row show the worst case possible balance for a resolution III triple of three factors with 2, 2, and 4 levels, and the best possible balance achievable in a full factorial design (not achievable in 72 runs).

Why does balance matter? This is most easily explained when looking at the worst case picture: in the top left mosaic plot of **Figure 1**, the combinations A = 1 and B = 1 or A = 2 and B = 2 imply that F can only take levels 1 or 3, while A = 1 and B = 2 or A = 2 and B = 1 imply that F can only take levels 2 or 4. Thus, if there were an interaction effect of the factors A and B, this would directly (and heavily) impact estimation of the main effect of factor F. This is called confounding. If one would use this design for an experiment and would not include an interaction between A and B into the model even though an interaction effect exists, conclusions about the main effect of factor F would be seriously biased. The top right mosaic plot shows the perfect balance in the full factorial situation, where such bias can be completely avoided. Both experimental plans shown in the bottom row are at least much better than the worst case reference, with the plan obtained from **DoE.MIParray** being better than the one that was actually used in [11]. Their quality can be compared by calculating their GWLP's with function `GWLP` from R package **DoE.base**; that function is also available within package **DoE.MIParray** directly, for convenience: The worse array has GWLP=(1, 0, 0, 73/162, 263/81, 20/9, 82/81, 11/162), the better one GWLP=(1, 0, 0, 2/27, 221/54, 113/54, 1/2, 13/54). In fact, the A_3-value

Table 1: Full factorial design in lexicographic order, with counting vector ***r*** for Design 3 of Table 2.

r	recipe	bp	oven
1	new	B1	elect
0	new	B1	gas
1	new	B2	elect
0	new	B2	gas
0	new	B3	elect
1	new	B3	gas
0	old	B1	elect
1	old	B1	gas
0	old	B2	elect
1	old	B2	gas
1	old	B3	elect
0	old	B3	gas

Table 2: Three small fractional factorial designs.

Design 1				Design 2				Design 3			
run	recipe	bp	oven	run	recipe	bp	oven	run	recipe	bp	oven
1	new	B1	elect	1	new	B1	elect	1	new	B1	elect
2	new	B2	elect	2	new	B2	gas	2	new	B2	elect
3	new	B3	gas	3	new	B3	gas	3	new	B3	gas
4	old	B1	elect	4	old	B1	elect	4	old	B1	gas
5	old	B2	gas	5	old	B2	gas	5	old	B2	gas
6	old	B3	elect	6	old	B3	elect	6	old	B3	elect

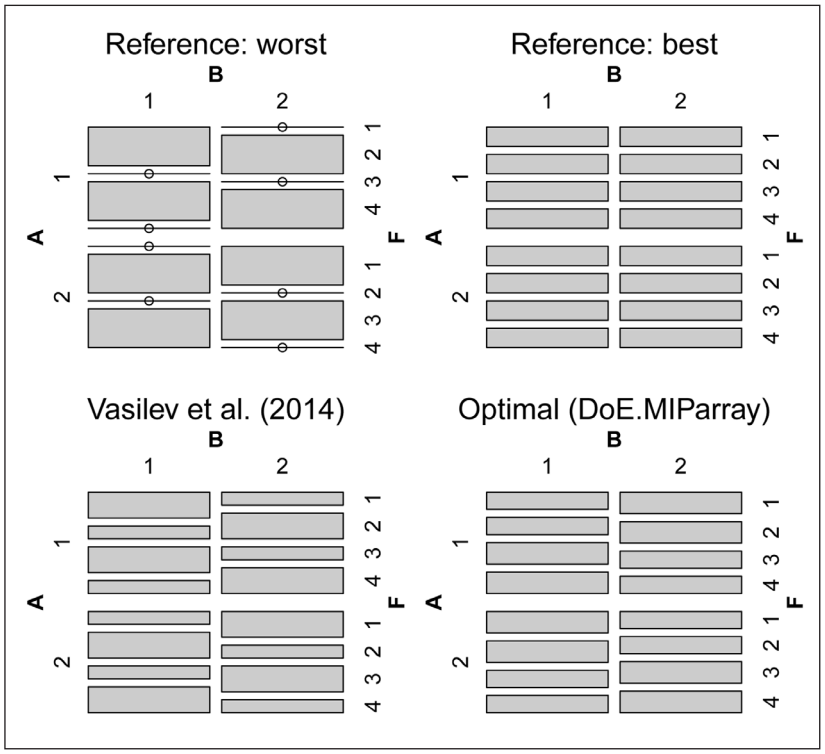

Figure 1: Worst case balance for two plans, compared to worst and best case references.

of the better array consists of six identical non-zero contributions from all six triples with 2, 2, and 4 levels that exhibit the balance shown in the bottom right plot of **Figure 1**. In the absence of any complete confounding (i.e. no *R*-tuples like those in the worst-case mosaic plot exist), all effects can in principle be estimated; the less severe the confounding, the less bias do we get from wrongly omitting a relevant effect.

Counting vector representation of a design

The counting vector **r** in **Table 1** indicates that six of the level combinations occur once while six other level combinations do not occur at all. The fractional factorial Design 3 from **Table 2** contains exactly those experimental runs, for which the vector **r** contains a "1" entry; thus, the vector **r** is the counting vector representation of Design 3. Using a counting vector representation requires an agreement on a natural order for the full factorial design; it is customary to use the lexicographic order that is also depicted in **Table 1**: the levels of the left-most factor change most slowly, the levels of the right-most factor change fastest.

The optimization problem

Grömping and Fontana ([5]) describe the specifics of the optimization problem that the R package **DoE.MIParray** solves for creating a design in *n* runs. An overview is given below:

- The design is expressed in terms of the counting vector **r** (see **Table 1** for an example), constrained to non-negative integer elements with sum *n*, and possibly constrained to 0/1 entries for ensuring distinct experimental runs.
- The requested resolution *R* can be ascertained by a linear optimization step, using a suitably-normalized model matrix of a factorial model with up to *R*–1-factor interactions for the linear constraints.
- Subsequently, n^2A_R is minimized. The objective function $n^2A_R = \mathbf{r}^T\mathbf{H}\mathbf{r}$ is a quadratic form (with suitably chosen matrix **H** based on the model matrix of *R*-factor interactions). This integer-constrained quadratic optimization problem with linear equality and inequality constraints is recast into a linear problem with conic quadratic constraints for treating it with Gurobi or Mosek.
- Potentially, subsequent optimization steps may take care of optimizing further elements of the GWLP, with additional conic quadratic constraints for preserving optimality of already optimized elements of the GWLP.

Dependence on external optimizers

DoE.MIParray itself is open source and freely available on the Comprehensive R Archive Network (CRAN). However, its functionality relies on the availability of at least one of Gurobi ([6]) or Mosek ([10]). The reason is that mixed integer optimization for conic quadratic optimization problems is at present far superior in the commercial tools. Both Gurobi and Mosek offer a free academic license and R packages that act as interfaces. Users of package **DoE.MIParray** do not need to work with the commercial solvers directly, except for going through the installation instructions and obtaining license files as appropriate.

Feasibility of the optimization

In principle, it is possible to apply function `gurobi_MIParray` or `mosek_MIParray` for obtaining a GMA design: this requires to start from a specified resolution,

and to successively minimize A_R,...,A_m. However, this is only practically feasible for very small designs.

In many applications of practical relevance, one will have to constrain optimization to the most important entry A_R of the GWLP, i.e., to the most severe type of imbalance. This is the default behavior of functions `gurobi_MIParray` and `mosek_MIParray` (requested through the default setting `kmax = resolution`). As mixed integer optimization is notoriously difficult, it will also happen that the balance is improved (i.e. A_R is reduced) but optimality cannot be confirmed. According to [5], there are lower bounds on A_R; if these are attained, optimality regarding the most severe type of imbalance is confirmed. The 72 run design for the application in [11] is a case for which the lower bound on A_3 confirmed optimization success.

Note, that the properties of the space in which optimization is conducted may strongly depend on the ordering of the experimental factors (through the ordering of the numbers of factor levels). An optimum design can be found very fast for some level orderings, while it may take much longer to find the optimum for other cases. Therefore, package **DoE.MIParray** offers search functions `gurobi_MIPsearch` or `mosek_MIPsearch` for screening through all relevant level orderings. Although searching through level orderings may seem like substantial effort, the search approach can sometimes yield an optimum or at least a good solution much faster than the application of optimization for a fixed level ordering. With version 9 (released in May 2019), Mosek has introduced a seed (with default 42) that can be modified by the user and whose effect it is to influence the path through the search space. Potentially, users of Mosek 9 can use the varying of seeds as an alternative or a supplement to using the search over level orderings.

Related R packages

The CRAN Task View "Design of Experiments (DoE) & Analysis of Experimental Data" discusses R packages that are on CRAN and support experimental design tasks. This brief section comments on a small selection of those packages that could be used for similar applications as **DoE.MIParray**: relatively small experiments with several factors, some or all of which are qualitative, and all of which have a relatively small number of levels, not necessarily the same for different factors. **DoE.base** (see [4]) handles such situations using catalogued orthogonal arrays, possibly optimizing balance by column selection from these. **DoE.MIParray** algorithmically creates well-balanced arrays by mixed integer optimization; the author is not aware of any other software tool, in R or elsewhere, that implements the same approach.

DoE.base and **DoE.MIParray** refrain from assuming a specific model. Rather, they provide designs for estimating a factorial model with main effects and possibly low order interactions with as little confounding as possible, without having to specify certain effects to be active or inactive. R package **planor** (see [8]) also supports this model-robust concept for mixed level applications, however without optimizing worst case balance and with a restriction to regular designs, due to a different algorithmic approach; the latter restriction implies limitations for screening designs, even if overall optimum balance is achieved (see also [4]). There is another strategy that is also often chosen for experimenting on some factors with a few specific levels: if a suitable model for the response of interest can be reasonably assumed, this model can be pre-specified, and the design can be chosen to optimize aspects of estimation or prediction within that pre-specified model. This will lead to different optimality decisions; see for example the R packages **AlgDesign** (see [12]), **skpr** (see [9]) or **OptimalDesign** (see [7]); **planor** also permits the specification of a model (but not with the typical focus assumed by packages on optimal design). Like **DoE.MIParray**, package **OptimalDesign** uses mixed integer optimization with Gurobi (for providing exact designs), however with an entirely different optimization criterion.

Implementation and architecture

The R package **DoE.MIParray** supplements an R package suite provided by the author. It is closely related to the R package **DoE.base** (see [4]). Arrays created with **DoE.MIParray** can be used as input to function `oa.design` of that package. Most of the namespace from **DoE.base** is imported; a small selection of its functions is exported (see below).

DoE.MIParray offers three strategies:

- For a specific number of runs, a vector of numbers of factor levels, and a resolution R (default: $R = 3$), optimize A_R, ..., $A_{k\max}$; if $k_{\max} = m$, a GMA design is requested, if $k_{\max} = R$, only the most severe imbalance is minimized (which is the default). This strategy is implemented in functions `gurobi_MIParray` and `mosek_MIParray`, respectively. These functions allow to incorporate a starting array (argument `start`), or to extend an existing array by forcing a given array to be part of the returned design (argument `forced`).
- Where the previous strategy is not successful, try to optimize A_R by searching over level orderings, with a (relatively) low time limit for each level ordering. This strategy is implemented in functions `gurobi_MIPsearch` or `mosek_MIPsearch`, respectively (see also the section on "Feasibility of the optimization" in the introduction).
- Continue optimization efforts, starting from an earlier optimization result. One can try to further improve the previous attempt, e.g. further reduce A_R (`improve=TRUE`) or to reduce the next (not quite as serious) level of imbalance, while preserving the balance results that were already optimized (`improve=FALSE`). This strategy is implemented in functions `gurobi_MIPcontinue` or `mosek_MIPcontinue`, respectively. Note that it is possible to apply function `gurobi_MIPcontinue` to a result from a Mosek optimization, and vice versa; the internal functions `mosek2gurobi` and `gurobi2mosek` take care of translating the optimization problem between the two optimizers. Unfortunately, since

the optimization history cannot be stored, the `improve=TRUE` variant of continuation has to repeat a large portion of the optimization work.

The above-mentioned user-visible optimization functions implement the overall algorithm that is described in [5]. For individual optimization steps within the algorithm, the functions prepare inputs for functions `gurobi` from package **gurobi** or `mosek` from package **Rmosek**, respectively, and postprocess the resulting outputs (possibly modifying them to provide inputs to further optimization steps). These activities are supported by internal functions `..._modelAddConeQobj`, `..._modelAddLinear`, and `..._modelLastQuadconToLinear` (replace `...` with `gurobi` or `mosek`, respectively) that take care of casting certain aspects of the optimization problem into the form required by `gurobi` or `mosek`.

Apart from a few parameters that are explicitly accessed by specific arguments of the functions of package **DoE.MIParray** (`maxtime` for maximum search time, `nthread` for number of threads to use, plus a few more settings for Gurobi), users can specify all valid parameters for the respective optimizer through the function arguments `gurobi.params` or `mosek.params`, respectively. Furthermore, with Mosek there is also an argument `mosek.opts` that controls different types of settings (verbosity and solution detail). The parameter controlling the stopping bound for optimality is provided by package **DoE.MIParray**, if not overridden by the user.

The internal functions `countToDmixed` and `dToCount` switch back and forth between an array and its counting vector representation in lexicographic order, the internal function `ff` creates a full factorial design in the lexicographic order corresponding to the counting vector.

The optimization functions of package **DoE.MIParray** use functions from package **DoE.base** for pre-checking feasibility of a resolution/strength requirement (`oa_feasible`), for checking optimization success from the theoretical lower bound provided in [5] (`lowerbound_AR`), for calculating design properties (`GWLP`, `length2`, `length3`, `length4`, `length5`, `contr.XuWu`) and for printing designs (`print.oa`). These functions are also exported in order to make them accessible for users who do not want to load the large namespace of package **DoE.base**. Two additional functions for investigating design properties, `SCFTs` and `ICFTs`, are not used within **DoE.MIParray** but are likewise imported from **DoE.base** and exported for users' convenience, so that the more detailed metrics from [2] and [3] can also be directly used for inspecting optimization results.

Quality control

DoE.MIParray is on CRAN and has thus been checked with the usual CRAN checks. Due to the dependence on Mosek and/or Gurobi, functionality checks of the optimization functions are not possible as part of the CRAN checks, but only offline. Specific test cases with reference results are collected in a test directory (within directory INSTALL), which contains functional checks for various different parameter combinations for small instances of arrays. Users who are equipped to use `R CMD check` can use the test files for automatic checks whether the package works as intended, by running `R CMD check DoE.MIParray_0.13.tar.gz --test-dir=inst/testsWithGurobiAndMosek` on the package tarball; this will provide the differences between current and saved test outputs; the saved results cannot be expected to be reproduced under all circumstances (see below), but differences for the small test case examples should only refer to interim results. (Users who are not equipped to use `R CMD check` can run the R scripts from the test directory and compare their output to the saved reference output (ASCII format) by hand.) Furthermore, example code in the documentation provides functionality checks; the example optimizations are commented out (by `\dontrun`) for CRAN tests, but can be run using function `run_examples` from package **devtools** (running them takes a long while). All offline testing has been done on Windows 7 and Windows 10 machines only, using R versions 3.3 and higher.

DoE.MIParray has been used for creating a larger number of arrays for publication in [5], using Mosek 8.1 under Windows operating systems with two threads. The results are valid and reasonable and have been documented with successful level orderings and run times in the paper. Generally, within a software version, the results themselves are deterministic (i.e. repeating the same call will in principle yield the same result), but may be influenced by computational power of the machine, number of threads used, amount of simultaneous activity on the machine, operating system and so forth; this is especially true, if code is run with a time limit, which is usually recommended.

Changes in results have occurred with version updates of Gurobi and particularly Mosek; this is due to changed algorithmic decisions within the softwares and can for specific examples be both beneficial or detrimental: for example, the code that produced the optimum design that could have replaced the design used in [4] with version 8 of Mosek (under Windows systems using two threads) no longer yields a design with globally optimal A_3 in Mosek version 9. Different results than those reported in [5] for Mosek 8 have also been observed for other examples (sometimes better, sometimes worse). It is therefore not reasonable to implement more extensive automated testing; in case of software modifications, the test cases from the package should be run, and results inspected with care by a human.

(2) Availability

Operating system

All systems that run R 3.3 or higher

Programming language

R 3.3 or higher

Additional system requirements

Availability of several cores is beneficial.
Availability of large RAM is beneficial.

Dependencies

Availability of Mosek 8 or higher and/or Gurobi 7.5. or higher.

R packages: combinat, DoE.base, gurobi (from vendor) and/or Rmosek (from vendor), slam (>= 0.1–9), Matrix (>= 1.1.0).

List of contributors

Hongquan Xu (UCLA) contributed code for function GWLP which is imported from package **DoE.base**.

Software location

Archive

Name: Comprehensive R Archive Network (CRAN).
Persistent identifier: https://cran.r-project.org/package=DoE.MIParray
Licence: GPL (>= 2)
Publisher: Ulrike Grömping
Version published: 0.13
Date published: 13/07/19

Code repository

Name: Github
Identifier: https://github.com/cran/DoE.MIParray
Licence: GPL (>= 2)
Date published: 14/07/19

Language

English

(3) Reuse potential

DoE.MIParray can be used by everybody who wants to create an array for experimentation. It is particularly suitable for creating relatively small mixed level arrays. The arrays can be used in any context, inside and outside of R.

Extensions to other optimizers (e.g. CPLEX [www.cplex.com] or suitable open source ones) would be very welcome; if there is an R API for an optimizer, functions `gurobi2mosek` and `mosek2gurobi` could serve as role models for switching inputs between API's. If someone volunteers to contribute such functions, I would be more than willing to include them into the package.

Likewise, Gurobi and Mosek have APIs for other softwares, e.g. Python or Matlab. Software developers are welcome to adapt **DoE.MIParray** to other APIs.

There is no official support for this software, but interested users can contact the author by e-mail. Bug reports are of course welcome and will be acted upon.

Acknowledgements

The collaboration with Roberto Fontana triggered the creation of the software.

Competing Interests

The author has no competing interests to declare.

References

1. **Grömping, U** 2014 Mosaic Plots Are Useful for Visualizing Low-Order Projections of Factorial Designs. *The American Statistician*. 68(2): 108–16. Taylor & Francis. DOI: https://doi.org/10.1080/00031305.2014.896829
2. **Grömping, U** 2017 Frequency Tables for the Coding Invariant Quality Assessment of Factorial Designs. *IISE Transactions*. 49: 505–17. DOI: https://doi.org/10.1080/0740817X.2016.1241458
3. **Grömping, U** 2018a Coding Invariance in Factorial Linear Models and a New Tool for Assessing Combinatorial Equivalence of Factorial Designs. *Journal of Statistical Planning and Inference*. 193: 1–14. DOI: https://doi.org/10.1016/j.jspi.2017.07.004
4. **Grömping, U** 2018b R Package DoE.Base for Factorial Experiments. *Journal of Statistical Software*. 85(5). Foundation for Open Access Statistic. DOI: https://doi.org/10.18637/jss.v085.i05
5. **Grömping, U** and **Fontana, R** 2019 An Algorithm for Generating Good Mixed Level Factorial Designs. *Computational Statistics & Data Analysis*. 137 (September): 101–14. Elsevier BV. DOI: https://doi.org/10.1016/j.csda.2019.01.020
6. **Gurobi Optimization LLC** 2018 Gurobi Optimizer Reference Manual. http://www.gurobi.com.
7. **Harman, R** and **Filova, L** 2019 *OptimalDesign: A Toolbox for Computing Efficient Designs of Experiments*. https://CRAN.R-project.org/package=OptimalDesign.
8. **Kobilinsky, A, Bouvier, A** and **Monod, H** 2018 *planor: An R Package for the Automatic Generation of Regular Fractional Factorial Designs*. https://CRAN.R-project.org/package=planor. DOI: https://doi.org/10.1016/j.csda.2016.09.003
9. **Morgan-Wall, T** and **Khoury, G** 2018 *skpr: Design of Experiments Suite: Generate and Evaluate Optimal Designs*. https://CRAN.R-project.org/package=skpr.
10. **Mosek ApS** 2018 *MOSEK Modeling Cookbook*. https://docs.mosek.com/modeling-cookbook/index.html.
11. **Vasilev, N, Schmitz, C, Grömping, U, Fischer, R** and **Schillberg, S** 2014 Assessment of Cultivation Factors That Affect Biomass and Geraniol Production in Transgenic Tobacco Cell Suspension Cultures. *PLoS ONE*. 9(8). DOI: https://doi.org/10.1371/journal.pone.0104620
12. **Wheeler, B** 2014 *AlgDesign: Algorithmic Experimental Design*. https://CRAN.R-project.org/package=AlgDesign.
13. **Xu, H** and **Wu, CFJ** 2001 Generalized Minimum Aberration for Asymmetrical Fractional Factorial Designs. *The Annals of Statistics*. 29(4): 1066–77. DOI: https://doi.org/10.1214/aos/1013699993

How to cite this article: Grömping, U 2020 DoE.MIParray: An R Package for Algorithmic Creation of Orthogonal Arrays. *Journal of Open Research Software,* 8: 24. DOI: https://doi.org/10.5334/jors.286

Submitted: 17 July 2019 **Accepted:** 17 September 2020 **Published:** 07 October 2020

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press.

OPEN ACCESS