Supplementary material for 'An Algorithm for Generating Good Mixed Level Factorial Designs'

Ulrike Grömping and Roberto Fontana

05 November 2018

Contents

1 Approach for optimizing designs for the Hadamard approach

For obtaining OA(12, $2^a3^14^1$, 1) from a Hadamard-based OA(12, $2^{11}12^1$, 2) (the `L12.2.11` from R package DoE.base augmented by a 12-level column), a brute force search for the best A_2 over all selections of a out of the 11 2-level columns was applied; among designs with the same A_2 value, the one with the best generalized resolution was chosen (Grömping and Xu 2014). The search was applied to each of 576 variants of assigning a 3×4 full factorial to the 12 run factor, obtained by

- assigning the first level of the three-level factor to runs 1 to 4, the second level to runs 5 to 8 and the third level to runs 9 to 12
- and assigning levels 1 to 4 of the four-level factor to runs 1 to 4, while combining each of the 24 permutations of the four levels for runs 5 to 8 with each each of the 24 permutations of the four levels for runs 9 to 12 (i.e., 576 variants of allocating the 4-level factor).

This pragmatic approach of creating a 3-level and a 4-level factor was chosen for feasibility reasons:

- on the one hand, working with only a single order of the 3×4 full factorial does not yield the best results; for example, the optimum design for $a = 4$ is not found by this approach.
- on the other hand, for covering all structurally different allocations, one would have had to also vary the possible decompositions of the 12 runs into three groups for the levels of the three-level factor (5775 variants); this would have led to more than three million (5775×576) variants from each of which which to choose a columns (up to 462 column allocations to be tried for each variant); for gaining an impression of the time needed, this latter approach has been taken for the largest array ($a = 11$), for which no column allocations are needed; the time used was about four hours and 40 minutes; the result obtained by the pragmatic approach was not improved (but might be in general).
- furthermore, including resolution II designs in 3 and 4 levels instead of full factorials only could have possibly yielded an improved overall result.
- Even with the chosen limited approach, allocation times were substantial for settings with medium number of 2-level factors.

Optimizing A_2 or $E(\chi^2)$ without watching out for total confounding is questionable for supersaturated designs, since it might be possible but almost never desirable to end up with total confounding between two columns; the Hadamard approach per construction prevents total confounding between the 2-level factors.

However, total confounding between a 2-level factor and the 4-level factor might occur. This indeed happened for $a = 10$; resolving ties by choosing the design with the larger generalized resolution (GR, see Grömping and Xu 2014) removed the total confounding. Depending on the frequency of ties, run times increase by incorporating this feature.

2 Further examples that illustrate the scope and limitations of our algorithm

2.1 Designs of strength at least 2

Coverage of mixed level arrays of larger strength in existing sources is a little patchy; for example, Eendebak and Schoen (2010) provided an OA(64, $2^2 4^1 8^1$, 3), but did not comment on the existence of arrays with more 2-level factors. For three and four 2-level factors, such arrays exist with $(A_4, A_5) = (3, 0)$ or $(A_4, A_5, A_6) = (6, 1, 0)$, respectively, and our algorithm finds non-regular arrays with these GWLPs after short optimization times (optimality of A_4 confirmed with the bound of Proposition 1). Regular arrays for these two scenarii can be found with the method published in Kobilinsky et al. (2017); this method does not optimize A_4 and finds arrays with $(A_4, A_5) = (3, 0)$ and $(A_4, A_5, A_6) = (7, 0, 0)$, respectively (with R package planor).

The remaining three example arrays of this section are related to experimentation with optimizer options from the development phase of R package DoE.MIParray: an OA(128, $2^2 4^3$, 4), and OA(96, $2^2 3^1 4^2$, 3) and an OA(48, $2^2 3^1 4^2$, 2). The factors to be considered were MIQCPMethod (2-level), MIPFocus (2-level) and Heuristics (4-level) for Gurobi parameters, and scenario (initially 4-level, later 3-level; different level patterns) and runsize (4-level, tiny to large with resulting numbers depending on the level pattern) for the experimental situation. A resolution V array for the initial level pattern requires 128 runs. It was created using Gurobi and Mosek, and creation was very fast, because its A_5 equals the lower bound 1. An array like this could also have been taken from the Eendebak and Schoen website (2010), which lists ten non-isomorphic GMA arrays, or could have been produced by the algorithm of Kobilinsky et al. (2017) (regular array without optimizing A_5, but also yielding the optimum $A_5 = 1$ in this case).

After further consideration, it was decided to drop one of the scenarii, because prior experimentation suggested that it would likely add substantial run time with only little information. For the resulting experimental situation, with factors at 2, 2, 3, 4, 4 levels, a resolution V array smaller than the 192 run full factorial is not possible (which follows from the fact that the least common multiple of $2 \cdot 2 \cdot 4 \cdot 4 = 64$ and $2 \cdot 3 \cdot 4 \cdot 4 = 96$ is 192). Since 192 runs were considered too many, a resolution IV half fraction in 96 runs was considered. The optimized array turned out to have an A_4 value of 1/9; this was considered acceptable, since follow-up runs could be used if deemed necessary for resolving potential ambiguities. This array was easily obtained both by Gurobi and by Mosek. An array like this is not available on the website by Eendebak and Schoen (2010), which does not cover any 96 run arrays with resolution III or IV.

Had one been prepared to contend with resolution III (not very sensible in the software development situation at hand), 48 runs would have done the job. That array is much harder to optimize than the 96 run array. Mosek found the GMA $A_3 = 2/9$ (equal to the bound of Proposition 1) in about 45 seconds. Optimizing A_4 did not improve the initial value 2.333, although a better A_4 exists. A search over level orders, as described in Section 4.4 of the paper, storing all 30 results from searches with time limit 60 s, produced a better array from the level order 4,3,2,2,4: $(A_3, A_4, A_5) = (2/9, 17/9, 8/9)$; this is the best GWLP we obtained for this setting. Gurobi produced the same GWLP starting from a different level order (4,3,2,4,2), while the design it produced from the level order that was most successful under Mosek had a worse GWLP ($A_4 = 23/9$). As the website by Eendebak and Schoen (2010) does not cover the situation for this array, it is not known to us whether this GWLP is optimal or whether its A_4 can be improved.

Now suppose that a third 2-level factor is needed. In this case, $A_4 + A_5 + A_6$ becomes 1 for 192 runs, 3 for 96 runs and 7 for 48 runs, according to Proposition 3. The lower bounds for A_R are as follows: $A_5 \geq 1/9$ ($n = 192$), $A_4 \geq 1/3$ ($n = 96$) and $A_3 \geq 1/3$ ($n = 48$). The confirmed GMA design in 192 runs was found within 22 s (lower bound for A_5 met, $A_6 = 8/9$ follows from Proposition 3). For 96 runs, the search over

level orders strategy was used (as mentioned in Section 4.4); after a total search time of about 7 minutes, the sixth order (4,2,2,2,4,3) yielded the lower bound $A_4 = 1/3$; subsequent efforts to reduce A_5 from 8/3 were unsuccessful, and no other level order produced a better A_5 either; thus, we do not know whether the array we found has GMA, or whether an array with smaller A_5 exists. Using Gurobi did not yield a better array and required much more time to yield an equally good one. For 48 runs, it is much harder to find a GMA array and confirm its optimality: an OA(48, $2^3 3^1 4^2$, 2) with $A_3 = 1/3$ (the lower bound) was not found; the best such array found, using the search strategy, had $A_3 = 0.389$ (e.g. for the level order 4,2,2,2,3,4); this value could not be improved, even after rerunning the best orders with 10min each, or after re-running the first optimum order for an entire hour; thus, we have obtained a "good design" with $(A_3, A_4, A_5, A_6) = (7/18, 5, 1.5, 1/9)$, without knowing whether it has a GMA A_3, let alone whether it has GMA overall.

2.2 Designs of strength 1 / resolution II

Liu and Liu (2012) provided a weak equidistant OA(12, $2^9 3^3$, 1); they did not explicitly optimize A_2 or $E(\chi^2)$; however, the array's $E(\chi^2) = 0.515$ is close to the lower bound 0.496 for $E(\chi^2)$. A different array with the same $E(\chi^2)$ was obtained with our algorithm using Mosek (the algorithm was run with a time limit of one hour; an approximately 45 min presolve step was followed by brief improvements). For a related array reported by Liu and Liu with an additional 6-level factor, our computer was not capable of accommodating the resource requirements by our algorithm; the same is true for most other arrays provided in Liu and Liu (2012).

3 The new designs from the paper

The R workspaces stored in files `72runsGOOD.rda`, `largedesignsTable5.rda` and `MIP12.rda` contain new good designs created with our algorithm, as described in Sections 5.2.2 and 5.3 of the paper. These can be loaded with the following commands (provided they are stored in the current working directory). The 72 run designs are stored as separate objects with names as usual in R package DoE.base (for example, `L72.2.1.3.1.4.1.6.1` denotes an array in 72 runs with one 2-level factor, one 3-level factor, one 4-level factor and one 6-level factor); the 12 run designs are stored in the list object `MIP12`, which contains the OA(12, $2^a 3^1 4^1$, 1) as its ath element; the code below displays the 10th element of that list. The commented code lines below that array show how to export arrays to `csv` files (remove the comment character `#` for running the code).

```
load("72runsGOOD.rda")
load("largedesignsTable5.rda")
load("MIP12.rda")
ls()
```

```
##  [1] "L144.2.1.3.2.4.2"     "L144.2.2.3.2.4.2"     "L192.2.1.3.1.4.3"
##  [4] "L192.2.2.4.2.6.1"     "L192.2.3.3.1.4.2"     "L216.2.1.3.2.4.1.6.1"
##  [7] "L288.3.2.4.2.6.1"     "L384.2.4.3.1.4.2"     "L432.2.1.3.3.4.2"
## [10] "L576.2.2.3.1.4.2.6.1" "L576.3.1.4.3.6.1"     "L72.2.1.3.1.4.1.6.1"
## [13] "L72.2.1.3.2.6.1"      "L72.2.1.3.3.4.1"      "L72.2.1.3.3.6.1"
## [16] "L72.2.2.3.1.4.1.6.1"  "L72.2.2.3.2.4.1"      "L72.2.2.3.2.6.1"
## [19] "L72.2.2.3.3.4.1"      "L72.2.2.3.3.6.1"      "L72.2.3.3.2.4.1"
## [22] "L72.2.3.3.2.6.1"      "L72.2.3.3.3"          "L72.2.3.3.3.4.1"
## [25] "L72.2.3.3.3.6.1"      "L72.2.4.3.2.4.1"      "L72.2.4.3.2.6.1"
## [28] "L72.2.4.3.3"          "L72.2.5.3.2.4.1"      "L72.3.2.6.1"
## [31] "L72.3.3.4.1"          "L72.3.3.6.1"          "MIP12"
```

```
MIP12[[10]]
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
##  [1,]    1    1    1    1    1    1    1    2    1     2     2     4
##  [2,]    1    1    2    1    1    2    2    1    2     1     2     3
##  [3,]    1    1    2    1    2    2    1    1    1     2     1     2
##  [4,]    1    2    1    2    1    1    2    1    1     1     1     1
##  [5,]    1    2    1    2    2    2    2    1    2     2     3     4
##  [6,]    1    2    2    2    2    1    1    2    1     1     3     3
##  [7,]    2    1    1    1    2    2    2    2    1     1     3     1
##  [8,]    2    1    1    2    1    2    1    2    2     2     1     3
##  [9,]    2    1    1    2    2    1    1    1    2     1     2     2
## [10,]    2    2    2    1    1    1    1    1    2     2     3     1
## [11,]    2    2    2    1    2    1    2    2    2     1     1     4
## [12,]    2    2    2    2    1    2    2    2    1     2     2     2
## attr(,"class")
## [1] "oa"     "matrix"
## attr(,"origin")
## attr(,"origin")$package
## [1] "DoE.MIParry"
##
## attr(,"origin")$call
## mosek_MIParray(12, c(rep(2, a), 3, 4), 2, mosek.params = list(dparam = list(LOWER_OBJ_CUT = A2sMIP[a]
##     144 + 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)))
```

```
# write.csv("L72.2.2.3.1.4.1.6.1", file="L72.2.2.3.1.4.1.6.1.csv")
# write.csv(MIP12[[10]], file="SSA12.2.10.3.1.4.1.csv")
```

4 Additional reference

Liu, Y. and Liu, M.Q. (2012). Construction of equidistant and weak equidistant supersaturated designs. *Metrika* **75**, 33–53. DOI 10.1007/s00184-010-0313-9