Constructing Designs for Fault Location

Erin Lanus

Arizona State University
Joint work with Charles J Colbourn and Douglas C Montgomery

DATAWorks 2019

Interaction Testing and Fault Location

Interaction Testing:

- Testing software and hardware for faulty interactions
- Exhaustive testing – k components at v levels $\implies$ v^k tests
- Most faults caused by interactions of 4 to 6 components
- Testing all t-way combinations catches most faults
- Create test-suite from covering array to guarantee every t-way interaction in some test

Fault Location:

- Covering array not guaranteed to locate faulty interaction – two t-way interactions may appear only in same set of tests
- Other fault localization techniques find faults adaptively
- Some environments not suited to adaptive testing

Mixed-level Locating Arrays

MLA$(N; d, t, k, (v_1, v_2, ..., v_k))$

- $N \times k$ array where v_i symbols appear in column i
- exponential notation: v_i^j when j factors have v_i levels
- every set of d t-way interactions appears in unique set of rows
- (d, t)-locating array identifies the interactions causing a fault when no more than d faulty interactions
- $(\bar{d}, t)$-locating array ensures sets of at most d t-way interactions appear in unique sets of rows
- Alternatives to full-factorial designs for screening

Example MLA

Table 1: MLA$(10; 1, 2, 3, (2, 2, 3))$

	c_1	c_2	c_3
1	1	1	2
2	0	0	0
3	1	0	2
4	0	1	0
5	1	0	1
6	1	1	1
7	1	0	0
8	0	0	2
9	0	1	2
10	0	1	1

Table 2: Rows of 2-way interactions in MLA$(10; 1, 2, 3, (2, 2, 3))$

	c_1, c_2	c_1, c_3	c_2, c_3
00	{2,8}	{2,4}	{2,7}
10	{3,5,7}	{7}	{4}
01	{4,9,10}	{10}	{5}
11	{1,6}	{5,6}	{6,10}
02		{8,9}	{3,8}
12		{1,3}	{1,9}

Example MLA

Table 3: Rows of 2-way interactions in MLA$(10; 1, 2, 3, (2, 2, 3))$

	c_1, c_2	c_1, c_3	c_2, c_3
00	{2,8}	{2,4}	{2,7}
10	{3,5,7}	{7}	{4}
01	{4,9,10}	{10}	{5}
11	{1,6}	{5,6}	{6,10}
02		{8,9}	{3,8}
12		{1,3}	{1,9}

Not (2,2)-locating as
$\rho(\{\{(c_1, 0)(c_2, 0)\}, \{(c_2, 0)(c_3, 0)\}\}) =$
$\rho(\{\{(c_1, 0)(c_2, 0)\}, \{(c_1, 1)(c_3, 0)\}\}) = \{2, 7, 8\}.$

Partitioned Search with Column Resampling

Previous work limited to

- constructions from covering arrays with larger t
- restricted values of (d, t), e.g. (1,1), (1,2), (1,t), (2,t)

Partitioned Search with Column Resampling (PSCR)

- constructs mixed-level $(\bar{d}, t)$-locating arrays for general d and t with fewer rows than covering array construction
- builds from "scratch" or from initial set of tests
- adds rows when lack of progress determined
- two parts: verify and repair
- algorithmic framework with tunable parameters: *avoid*, *CHO*, α, *adaptive*-α

PSCR Verifier

Condition 1 (coverage): all t-way interactions covered in some row

Condition 2 (location): rows corresponding to a set of at most d t-way interactions are not the same as those for another set

- Partitions the search space by guessing the smallest row index involved in a collision
- Builds N trees, one for each row
- Only compares sets that could result in a collision, not all possible sets

Collision: two sets of at most d t-way interactions appearing the same rows

PSCR Repair

Failure 1: resample a randomly chosen column of non-appearing interaction until condition 1 passes

- *avoid* = *false* adds a new row after number of resamplings
- *avoid* = *true* add new row only if condition 1 never passed

Failure 2: collision handling option (CHO) determines resampling

- ($CHO = 0$) Greedy – first collision
- ($CHO = 1$) Random – random colliding interaction
- ($CHO = 2$) Interaction – most involved interaction
- ($CHO = 3$) Column – most involved column
- ($CHO = 4$) Adaptive – relative to percent colliding

PSCR Determination of Progress

After resampling, number of collisions converted to z-score, compared to mean of the history
$time = \frac{iteration}{maxIterations}$

- Rollback: $z \geq 1 - time$, many more collisions than mean
- Allow, add a row: $0 < z < 1 - time$, positive z-score
- Allow, add a row, record: compare threshold to percent of normal distribution to left of z-score; if $percent > \alpha$, the z-score does not reside in the left tail defined by α, so it is not progressing.
- Allow and record: if $percent \leq \alpha$, the algorithm is progressing

When $adaptive\text{-}\alpha = true$, threshold is $\alpha + .25time$

PSCR Determination of Progress: α vs. *adaptive*-α

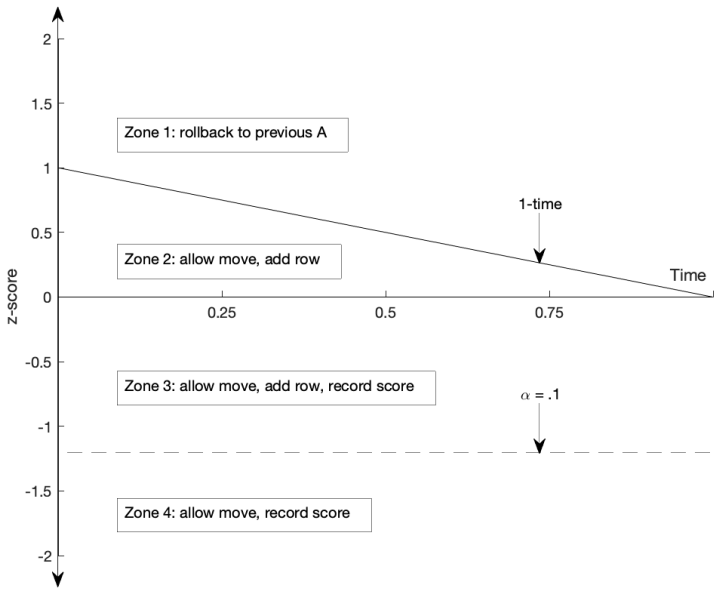

Figure 1: Zones for constant $\alpha = .1$. When $\alpha = .1$,$z = -1.2$ in zone 3, $z = -1.3$ in zone 4.

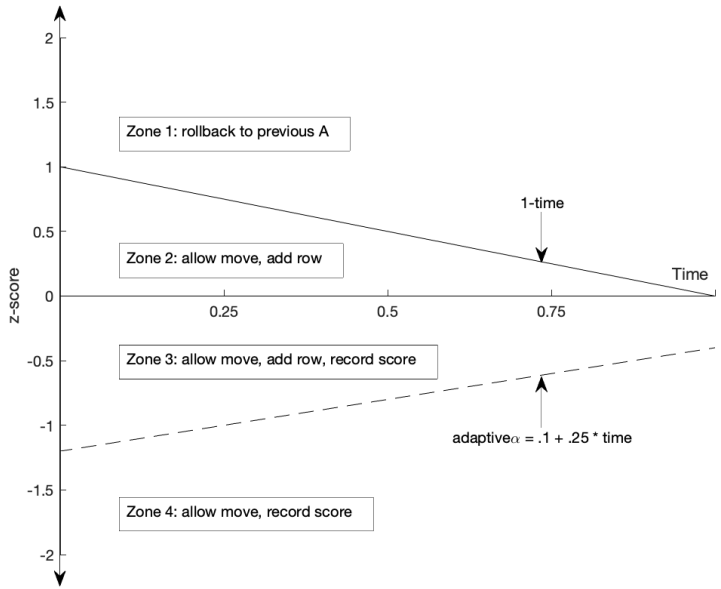

Figure 2: Zones for *adaptive*-α, $\alpha = .1$. End of execution, $z = -.3$ in zone 3, $z = -.4$ in zone 4.

Parameter Tuning

Parameters and levels:

- *CHO* $\in \{0, .., 4\}$ – 5 levels
- $\alpha \in \{.001, .2, .4\}$ – 3 levels
- *adaptive*-α and *avoid* – 2 levels

Response variables: rows, iterations, and seconds
Design: full factorial with four replicates
Blocks: problem parameters

d	t	k	v	maxIterations
1	1	1000	7	600
1	2	25	6	250
1	3	8	4	300
2	1	20	5	500
2	2	5	3	350

Table 4: Blocks for $5 \times 5 \times 3 \times 2 \times 2$ full factorial

Parameter Tuning Results: CHO

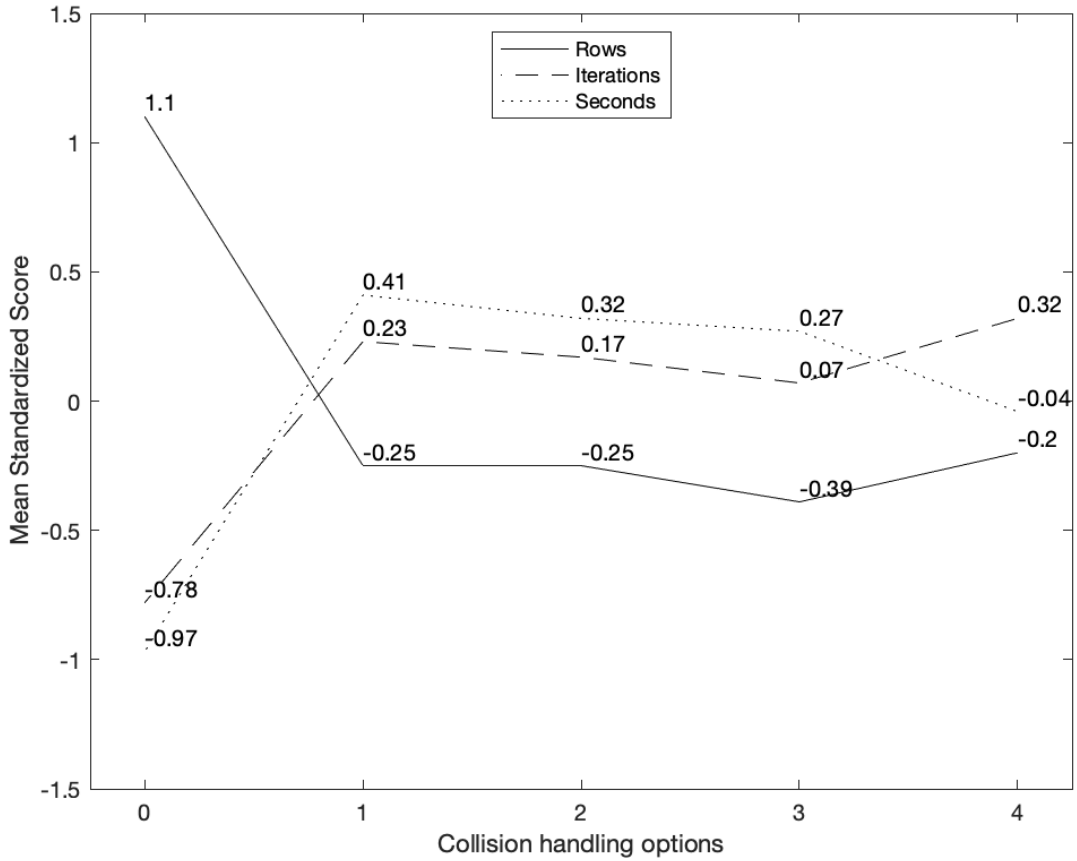

Figure 3: Effect of *CHO* on rows, iterations, and seconds

Parameter Tuning Results: α

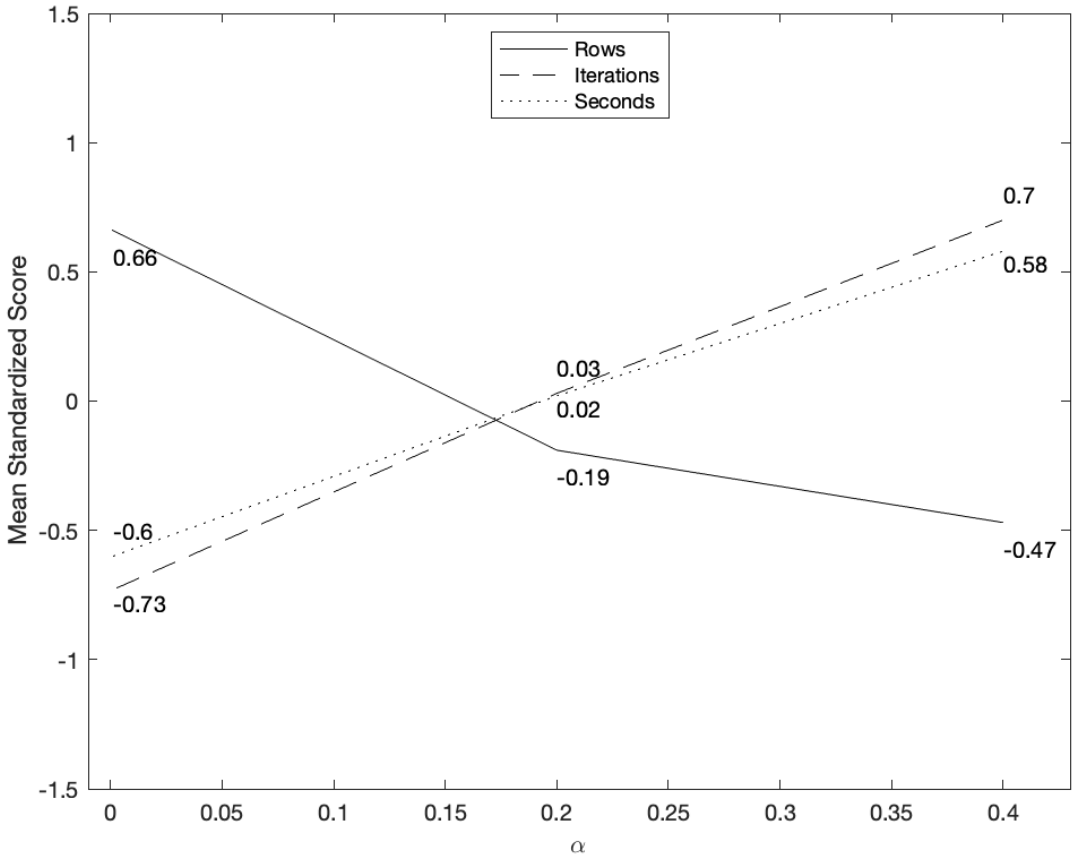

Figure 4: Effect of α on rows, iterations, and seconds

Parameter Tuning Results: *avoid*

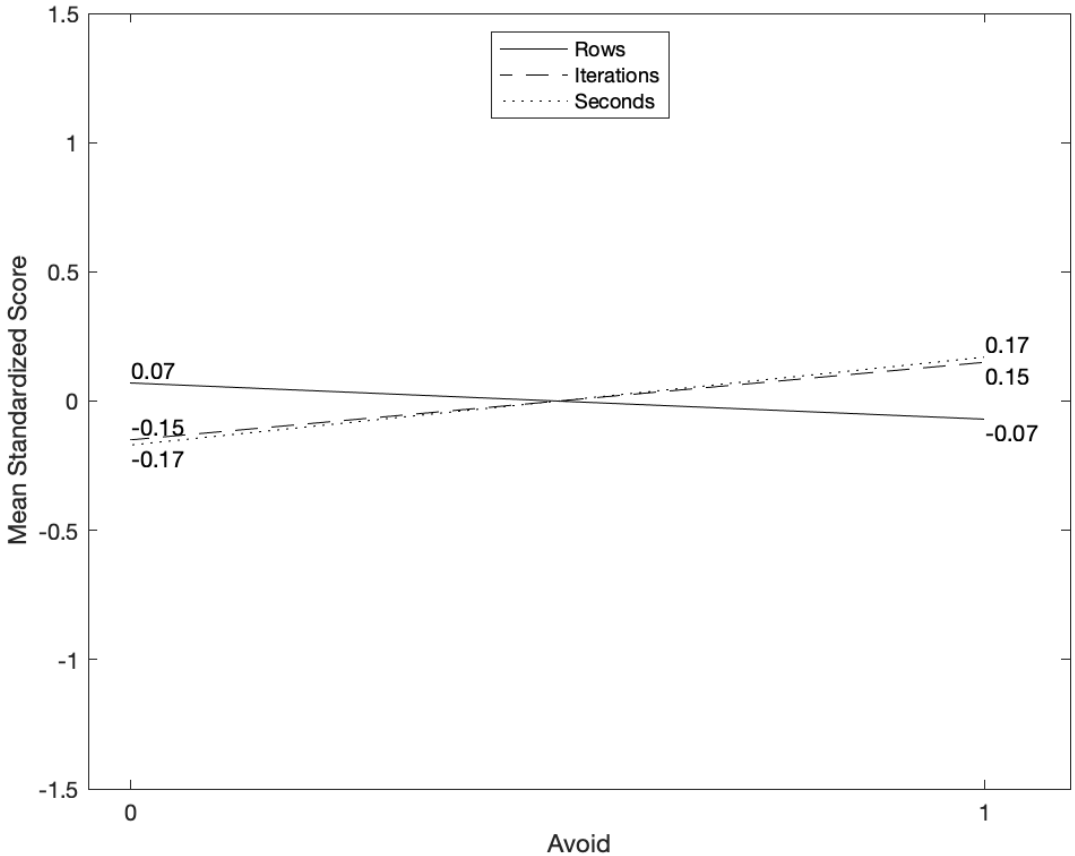

Figure 5: Effect of *avoid* on rows, iterations, and seconds

Parameter Tuning Results: *adaptive-*α

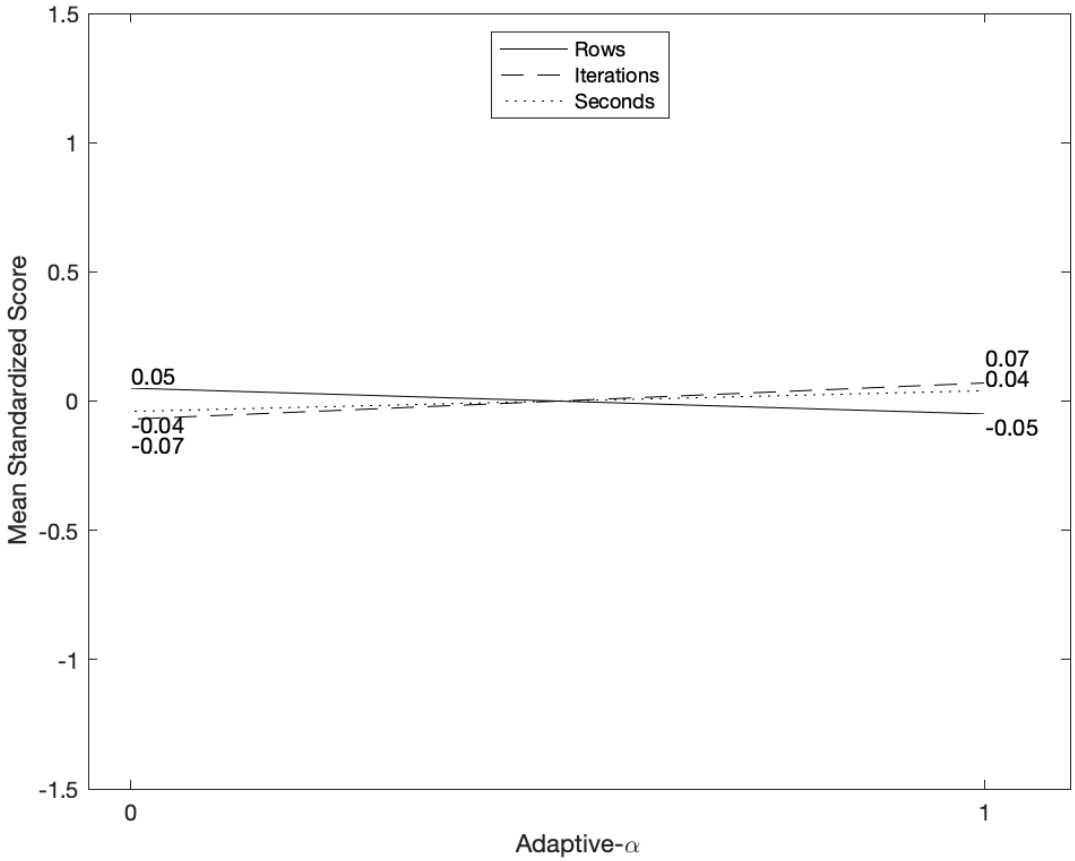

Figure 6: Effect of *adaptive-*α on rows, iterations, and seconds

Parameter Settings for Priorities

Priority	*CHO*	α	*adaptive*-α	avoid
Speed	0	any	0	0
Accuracy	3	$\geq .4$	1	1
Both	4	.2	{0,1}	{0,1}

Table 5: Recommended tunable parameters settings by priority

Example Results

PSCR outperforms the column resampling algorithm on both MLAs and is comparable to the local optimization algorithm from [6]

Factors	PSCR	Col Resampling	Local Op
$5^9 4^5 3^7 2^3$	110	117	114
$10^8 9^1 8^4 7^5 6^1 05^4 4^6 3^9 2^{28}$	534	580	532

Table 6: Rows in smallest LA by algorithm

Conclusions

PSCR

- first algorithm to construct MLAs for general d, t
- competitive with known methods
- parameters tunable to prioritize speed or accuracy

DOE approaches useful for tuning algorithmic parameters for construction of MLAs

Appendix I

Aggregate analysis from JMP 14.0.0

N	Full Factorial	Full no blocks	1,1	1,2	1,3	2,1	2,2
Source	Prob > F	Prob > F	Prob > F	Prob > F	Prob > F	Prob > F	Prob > F
CHO	<.0001*	<.0001*	<.0001*	<.0001*	<.0001*	<.0001*	<.0001*
Blocks*CHO	<.0001*						
Alpha(0.001,0.4)	<.0001*	<.0001*	<.0001*	<.0001*	<.0001*	<.0001*	<.0001*
Blocks*Alpha	<.0001*						
CHO*Alpha	<.0001*	<.0001*	<.0001*	<.0001*	0.6192	<.0001*	<.0001*
Blocks*CHO*Alpha	<.0001*						
adaptive-a	0.0041*	0.0076*	0.8455	<.0001*	0.8424	0.3868	0.1692
CHO*adaptive-a	0.0262*	0.0489*	0.4351	0.4696	0.147	0.8067	0.8103
avoid	<.0001*	0.0001*	0.9546	0.3488	<.0001*	0.5954	0.1219
Blocks*avoid	<.0001*						
Alpha*avoid	0.0398*	0.0559	0.7797	0.7311	0.0154*	0.7982	0.4457
Alpha*adaptive-a*avoid	0.732	0.7502	0.8445	0.0169*	0.3774	0.2674	0.5586
CHO*Alpha*adaptive-a*avoid	0.0293*	0.0537	0.8761	<.0001*	0.2921	0.7482	0.3522
Blocks*CHO*Alpha*adaptive-a*avoid	0.0233*						

Figure 7: Comparison of effects in $5 \times 5 \times 3 \times 2 \times 2$ full factorial on response rows

Appendix II

Iterations	Full Factorial	Full no blocks	1,1	1,2	1,3	2,1	2,2
Source	Prob > F	Prob > F	Prob > F	Prob > F	Prob > F	Prob > F	Prob > F
CHO	<.0001*	<.0001*	<.0001*	<.0001*	<.0001*	<.0001*	<.0001*
Blocks*CHO	<.0001*						
Alpha(0.001,0.4)	<.0001*	<.0001*	<.0001*	<.0001*	<.0001*	<.0001*	<.0001*
Blocks*Alpha	<.0001*						
CHO*Alpha	<.0001*	<.0001*	<.0001*	<.0001*	0.0335*	<.0001*	<.0001*
Blocks*CHO*Alpha	<.0001*						
adaptive-a	<.0001*	0.0002*	0.0734	0.0011*	0.4195	0.064	0.0092*
Alpha*adaptive-a	0.1142	0.1728	0.4597	0.0453*	0.715	0.3103	0.3743
avoid	<.0001*	<.0001*	0.2825	0.0045*	<.0001*	0.2294	0.0003*
Blocks*avoid	<.0001*						
CHO*avoid	0.0136*	0.0524	0.922	0.3173	0.0007*	0.7684	0.6837
Blocks*CHO*avoid	0.0079*						
Alpha*avoid	0.0285*	0.0588	0.9702	0.8139	0.0055*	0.2408	0.0133*
Blocks*Alpha*avoid	0.0030*						
CHO*Alpha*adaptive-a*avoid	0.0473*	0.1272	0.7361	0.0163*	0.311	0.2109	0.9899

Figure 8: Comparison of effects in $5 \times 5 \times 3 \times 2 \times 2$ full factorial on response iterations

Appendix III

Seconds	Full Factorial	Full no blocks	1,1	1,2	1,3	2,1	2,2
Source	Prob > F	Prob > F	Prob > F	Prob > F	Prob > F	Prob > F	Prob > F
CHO	<.0001*	<.0001*	<.0001*	<.0001*	0.0486*	<.0001*	<.0001*
Blocks*CHO	<.0001*						
Alpha(0.001,0.4)	<.0001*	<.0001*	<.0001*	<.0001*	0.0768	<.0001*	<.0001*
Blocks*Alpha	<.0001*						
CHO*Alpha	<.0001*	<.0001*	<.0001*	<.0001*	0.796	<.0001*	<.0001*
Blocks*CHO*Alpha	<.0001*						
adaptive-a	0.0063*	0.0333*	0.0919	0.0175*	0.6103	0.0010*	0.0095*
avoid	<.0001*	<.0001*	0.3025	<.0001*	<.0001*	0.1334	<.0001*
Blocks*avoid	<.0001*						
CHO*avoid	0.0098*	0.0879	0.918	0.3041	0.1208	0.8759	0.0001*
Alpha*avoid	0.0553	0.1353	0.9256	0.7262	0.1981	0.2579	0.0014*

Figure 9: Comparison of effects in $5 \times 5 \times 3 \times 2 \times 2$ full factorial on response seconds

References I

[1] D. R. Kuhn, R. N. Kacker, and Y. Lei, *NIST Special Publication 800-142. Practical combinatorial testing*.
National Institute of Standards & Technology, 2010.

[2] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, "Software fault interactions and implications for software testing," *IEEE Transactions on Software Engineering*, vol. 30, pp. 418–421, June 2004.

[3] D. R. Kuhn, R. Bryce, F. Duan, L. S. Ghandehari, Y. Lei, and R. N. Kacker, "Combinatorial testing: Theory and practice," in *Advances in Computers* (A. Memon, ed.), vol. 99, ch. 1, pp. 1 – 66, Elsevier, 2015.

[4] L. S. G. Ghandehari, Y. Lei, T. Xie, R. Kuhn, and R. Kacker, "Identifying failure-inducing combinations in a combinatorial test set," in *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pp. 370–379, Apr. 2012.

[5] X. Niu, C. Nie, H. K. Leung, Y. Lei, X. Wang, J. Xu, and Y. Wang, "An interleaving approach to combinatorial testing and failure-inducing interaction identification," *IEEE Transactions on Software Engineering*, pp. 1–33, 2018.

References II

[6] S. A. Seidel, K. Sarkar, C. J. Colbourn, and V. R. Syrotiuk, "Separating interaction effects using locating and detecting arrays," in *Combinatorial Algorithms* (C. Iliopoulos, H. W. Leong, and W.-K. Sung, eds.), (Cham), pp. 349–360, Springer International Publishing, 2018.0

[7] C. J. Colbourn and D. W. McClary, "Locating and detecting arrays for interaction faults," *Journal of Combinatorial Optimization*, vol. 15, pp. 17–48, Jan. 2008.

[8] E. Lanus, C.J. Colbourn, and D.C. Montgomery, "Partitioned Search with Column Resampling for Locating Array Construction," *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops*, to appear (accepted 2/2019).

[9] "JMP® statistical software from SAS, version 14.0.0.."