PharmaSUG 2021 - Paper EP-124

No Black Boxes: An Utterly Transparent Batch Randomization System in R

Dennis Sweitzer, Ph.D., YPrime

ABSTRACT

Randomized trials are a key part of clinical research, but are often implemented as somewhat of an after thought. In order to accelerate the delivery of high quality randomization schedules, we implement a batch stratified permuted block randomization system which includes: A flexible spreadsheet based syntax to structure the randomization as a series of 'superblocks' within each strata, cohort, or combination, easing the use of best practices such as variable block sizes, and relaxed or mixed permuted block methods; Complete computational transparency, with an audit trail from parameters and random numbers through block selection and unit assignments, and implemented in open-source R code; Simulations by default for testing, validation, or rerandomization analyses using pseudo-random number streams generated by the Wichmann-Hill 2006 algorithm for repeatable and independent parallel random number sequences; Reporting functions to measure quality of generated schedules on metrics such as potential selection bias, treatment imbalance, and risk of under-enrolling strata. The resulting system allows rapid implementation, testing, validation and documentation.

INTRODUCTION

In an ideal experiment, treatments would be applied to experimental units which are identical in all ways. However, in a clinical trial, the experimental units are people who vary greatly in ways that affect clinical outcomes. Some of these confounding variables are measurable and participants can be grouped accordingly, while some are not measurable or poorly defined, and others may not even be known. Furthermore, clinicians may (either deliberately or unconsciously) assign treatments in a way that they perceive will best benefit the patients, or best achieve the perceived goals of the trial. The very real risk is that systemic imbalances of confounding variables between treatments affect the outcomes in a way that is attributed to the treatments—known as allocation bias.

Randomization (with blinding) is a solution to the problem of allocation bias, as it ensures that the impacts of known and unknown confounding variables are identically distributed between treatments with known mathematical properties. This allows valid comparisons between treatment groups, with estimates of how much the differences could vary between them.

While randomization is a key part of research, randomization planning often settles for 'fit-for-purpose' instead of 'best practices', for reasons from narrow study constraints (e.g., small phase 1 studies), to vendor constraints, or to simply lacking the resources to develop and justify alternatives. Often taking the path of least resistance in this way is adequate, but leaves a question of 'Could this be done better?'

To accelerate the delivery of consistently high-quality randomization schedules, YPrime implements a stratified permuted block randomization system built on four core principles:

Flexibility: Implement a simple spreadsheet-based syntax to structure a randomization schedule as a constellation of permuted blocks of specified sizes within strata, cohorts, or combination thereof. For instance, a Phase 1 ascending dose study with sentinel dosing is denoted (for each cohort) as containing 2 subgroups: the 1st group always comes first and contains the sentinel subjects, while the remainder of the cohort is a 2nd group consisting of random samples of varying sized permuted blocks.

Transparency: Generate, by default, a complete and detailed audit trail of all calculations used in generating a schedule. This feature used internally for testing new releases can be included as part of documentation. This solution is coded in standard R and is available for inspection and testing by sponsors (RStudio is recommended, but not necessary). Furthermore, instead of relying on the default random number generator, the Randomizer implements the Wichmann-Hill 2006 algorithm, which is computationally straight forward, and avoids initialization problems.

Resampling and Simulation: Automatically generate parallel and independent randomization schedules, each of which vary only in the random seed used to initialize the generator. During testing, these simulations can be used to verify that the randomization plan is indeed generating distinct random sequences as expected and can be analyzed in depth. During production, the simulated randomization can be used for re-randomization analyses of study data. The Wichmann-Hill 2006 random number generator is particularly well suited for this as it creates independent and non-overlapping sequences of random numbers without special code.

Measurability: While the simulated randomization schedules allow users to analyze the performance in any way they choose, summary metrics can be provided to assess the potential for selection bias, observed treatment imbalances, and risks of under-enrolling strata (such as confounding between treatment and site). By anticipating problems before they occur, often simple changes to the randomization plan will minimize associated risks.

PROCESS AND STRUCTURE

The YPrime randomizer system (Figure 1 below) begins with consultation between the sponsor and Project Manager to formalize requirements as the RSD (Requirements Specification Document) and MSD (Materials Specification Document). Optionally, the YPrime Randomization Specialist may consult on the specifications, especially regarding appropriateness to the study design, robustness to random variation or unplanned events, and implementation issues. During or after specifications are approved, the Randomization Specialist populates an Excel spreadsheet with the parameters of the randomization scheme, which is interpreted by the R program RandomizerCore.R to generate the scheme in a standardized matter. Any required custom programing is done in the program RandomizerShell.R, which takes standardized randomization scheme and adds any custom modifications (such as structure of randomization IDs).

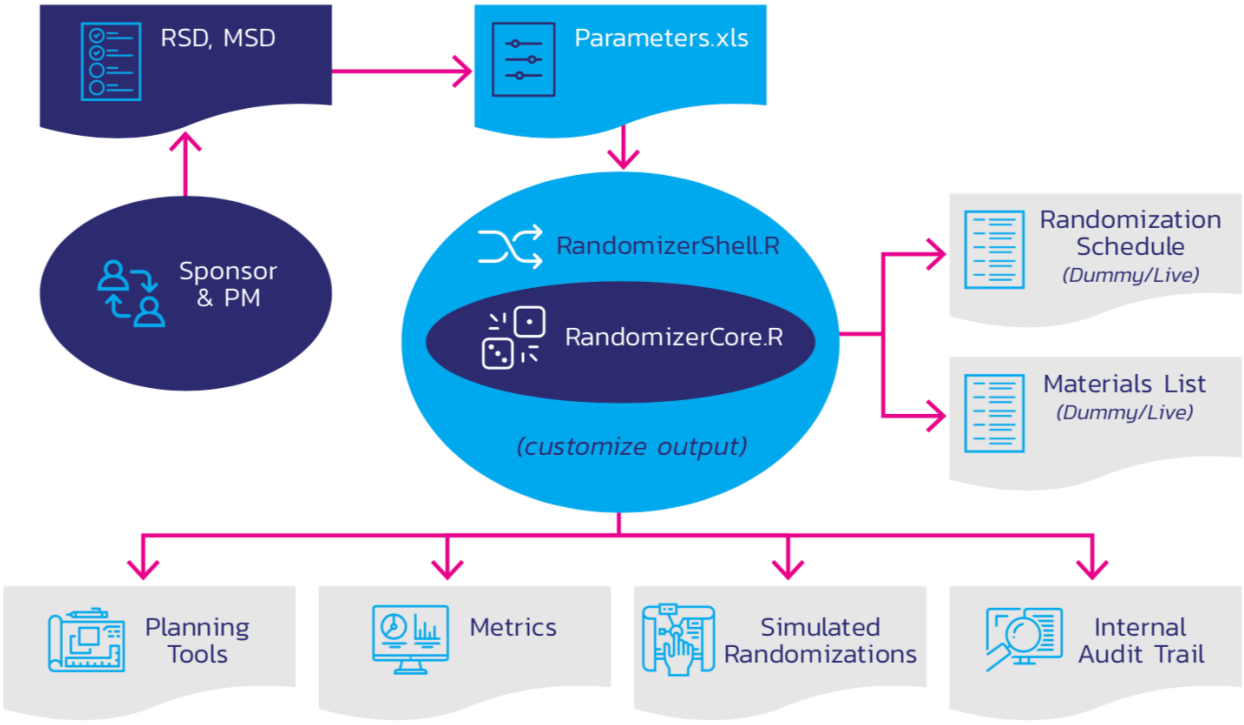

Figure 1. Components of YPrime Randomizer

Randomizer is executed from RStudio, creating Dummy or Live versions of the schedules, along with standard outputs for review or inclusion in any validation. The internal audit trail and simulated randomizations are useful for verifying that the algorithm is executing correctly. Planning tools aid in such tasks as generating all possible treatment sequences for cross-over study, simulating the distribution of patients across strata, or making lists of strata from all combinations of stratification factors (some are

coded by default, and some can be added ad hoc). The metrics primarily allow the comparison of randomization schemes using measures such as potential selection bias (metrics are under development).

SPECIFIC SYSTEM PRINCIPLES

FLEXIBILITY

Most randomization schedules have a simple hierarchical structure: subjects are either grouped into strata or cohorts (or both), and within each of those, subjects are randomized to one of the treatments (usually in permuted block patterns of treatments). If it is an adaptive trial, the study may be segmented above the strata/cohort level, so that the randomization scheme changes between segments. Structure usually exists below the strata/cohort level, which might be sequential grouping (such as sentinels in cohorts) or random groupings (such as permuted blocks).

YPrime Randomizer uses the following randomization hierarchy:

- Segment: portions of the study between which the randomization scheme changes (e.g., by dropping treatments after an interim analysis).
- Strata: Parallel and mostly independent groups of patients, usually receiving the same treatments (randomization schemes can vary between strata).
- Cohorts: Sequentially treated independent groups of patients, usually receiving different treatments (such as in an ascending dose study).
- SuperBlock: Sequential groups within a strata/cohort (e.g., the first 2 subjects might be a sentinel group, who are randomized and treated several days before the rest).
- Block: Exchangeable permutation blocks within a SuperBlock (e.g., 10 blocks of size 4 and 10 of size 6 might be selected in random order for a total of 100 subjects).

SuperBlocks and Blocks together generalize block constellations defined by Uschner et al, 2018, who defined two types of constellations: an ordinary block constellation is simply a sequence of blocks (of either fixed or varying size), where each block is a permutated block of the treatments (so for each position, there is one permuted block of the desired size and allocation within the block is randomly selected), and a random block constellation, in which the blocks themselves are chosen in a random order. Superblocks maintain the fixed order, and blocks occur in a random order within superblocks.

The parameter spreadsheet has tabs defining Strata, Cohorts, Segments, and Blocks.

Strata and Cohorts can be interchanged within the hierarchy (so each Strata can contain Cohorts, or vice versa), while the Segments tab is used to define the relationships between Strata and Cohorts within Segments. For convenience, Strata are grouped by kinds of strata, and the definition applied to all strata of that kind (likewise for Cohort).

Each kind of permuted block (denoted Block Kind) is defined on a tab in the Parameters spreadsheet, as shown in Table 1. The column Block Kind is the identifier used to define randomization constellations, the column “Treatments” lists all the treatments in the block, and the column “Permute?” can have 3 values: “Y” indicates that all permutations of the treatments are to be used (so Block Kind 2of4 is all permutations of AAPP); “N” indicates that the permutations to be used are enumerated (so all must be listed in the Tab); and “X” indicates that the permutation on the line should be excluded from the Block Kind.

Table 1. Permuted Block Definitions

Block Kind	Block Size	Permute?	Treatments
1of2	2	Y	A P
2of4	4	Y	A A P P
3of6	6	Y	A A A P P P
3of6	6	X	A A A P P P
3of6	6	X	P P P A A A
CR	1	N	A
CR	1	N	P
As	1	N	A
Ps	1	N	P
1of3	3	Y	A P P
2of3	3	Y	P A A

This example shows Block Kind 1of2 is a simple permuted block of size 2 (AP or PA), and Block Kind 2of4 is likewise a permuted block of size 4 (AABB, ABAB, ABBA, BBAA, BABA, or BAAB). However, Block Kind 3of6 is not all 20 permutations of AAAPPP, but excludes 2 permutations—AAAPPP and PPPAAA—to use only 18 of them ('X' marks the patterns to be excluded). Why would one want to do this?

In this case, one might want the increased randomness of the longer permuted blocks of 6, while avoiding excessive runs of any single treatment. E.g., block AAAPPP followed by block PPPAAA results in a run of 6 P's in a row. Excluding these 2 permutations avoids runs of length 5 or 6.

Block Kinds CR, As, and Ps are trivial examples of enumerated patterns. A CR (Complete Randomization) block is either A or P, while an As block is one of treatment A, and a Ps block is one of treatment P.

Table 2 defines several different randomization schemes (denoted as Strata Kind) for strata containing 100 units, or a cohort of 20 units. Different strata (or cohorts) can be assigned to different randomization schemes, such as might be needed for different sizes of strata or study segments.

Table 2. Definitions of Randomization Schemes for Strata

Strata Kind	Super Block	Block Kind	Freq.	Replace?
PBR4	1	2of4	25	Y
RBC246	1	1of2	8	Y
RBC246	1	2of4	9	Y
RBC246	1	3of6	8	Y
Sentinal	1	1of2	1	Y
Sentinal	2	2of4	3	Y
Sentinal	2	1of2	3	Y
CR	1	CR	100	Y
RAR	1	As	50	Y
RAR	1	Ps	50	Y
RBC23346	1	1of2	5	Y
RBC23346	1	2of4	6	Y
RBC23346	1	1of3	5	Y
RBC23346	1	2of3	5	Y
RBC23346	1	3of6	6	Y

This example demonstrates the following:

Strata Kind PBR4 is defined as only 1 Super Block, with 25 of Block Kind 2of4. (Replacement='Y' indicates that each block within a Block Kind are sampled with replacement, for example, each permutation can be reused any number of times. If Replacement='N', each permutation could only be used once).

- Strata Kind RBC246 consists of 8 permuted blocks of size 2 (Block Kind 1of2), 9 of size 4 (Block Kind 2of4), and 8 of size 6 (Block Kind 3of6).
- Strata Kind Sentinel defines a cohort of 20 units, with a sentinel group of 2 (randomized to Block Kind 1of2 within SuperBlock#1), followed by the rest of the 18 units (in Super Block #2, consisting of 3 each of Block Kinds 2of4 and 1of2).
- Strata Kind CR defines complete randomization, in which every unit is equally likely to be assigned to either treatment (so potentially, all units could be assigned treatment 'A'). It is one Superblock, with 100 of Block Kind CR.
- Strata Kind RAR treatment (Random Allocation Rule) defines a strata consisting of any permutation of 50 of each. There is only one SuperBlock, 50 of each of BlockKind As and Ps.

Other Tabs which define the Strata, and Cohorts lists the identification, the name, a description, of each Strata or Cohort, plus the randomization scheme assigned to each. Consequently, there is much flexibility in defining randomization structures with each, allowing different sizes of Strata or Cohorts, or structures such as Sentinel Groups within cohorts or strata.

TRANSPARENCY

Every calculation can be traced from parameters within the Parameter.xls spreadsheet, to intermediate steps to the final choice of permuted blocks and treatment assignment. The audit results are automatically generated as part of the output results spreadsheet.

Two audit tabs are generated in the output: one for the random choices of blocks, and a second showing the mapping from the random blocks to the sequential treatments. Since an arbitrary number of simulated randomizations (aka, re-samplings) are also generated, audit results can be displayed for an arbitrary subset of simulations.

The basic Algorithm is simple:

1. Randomly pick blocks according to the randomization constellation within each Segment-Strata-Cohort group.
2. Randomly reorder the selected blocks within each Super Block (within Segment-Strata- Cohort).

The algorithm is thrifty with random numbers, in that only two random numbers are generated for each permuted block in a randomization scheme. For each block, one random number is used to choose a permutation from each Block Kind (so if the Block Kind is 2of4 as defined above, one of the 6 permutations is chosen at random using one random number); and another is assigned to each chosen block and used to reorder the blocks within SuperBlocks.

Table 3 below lists an example of the audit table for choosing random blocks. From left to right are columns identifying the group (idSegment, idCohort, and SuperBlock), the columns identifying the Block Kind along with block attributes and sampling specifications (Block Kind, Block Size, Fid Block, Lid Block, Freq(uency), and Replace?); The random numbers used to choose the permuted block (Rnd Samp), the permuted block chosen (Blk Pick), the random number for re-sorting (Rnd Ordr), and the block in final order within Super Block (Blk Ordr), and the permuted block itself (Blk Perm).

Table 3. Sample Block Audit Table

id Segt	id Cohort	Super Block	Block Kind	Block Size	Fid Block	Lid Block	Freq	Replace?	Rnd Samp	Blk Pick	Rnd Ordr	Blk Ordr	Blk Perm
Part A	A1	1	Sentinel	2	1	2	1	Y	0.6731	2	0.1793	2	P A
Part A	A1	2	Remain	6	3	8	1	Y	0.5974	6	0.6473	6	A A P A A A
Part A	A2	1	Sentinel	2	1	2	1	Y	0.1723	1	0.9953	1	A P
Part A	A2	2	Remain	6	3	8	1	Y	0.9937	8	0.0853	8	P A A A A A
Part B	B1	1	PartB	5	9	18	4	Y	0.1548	10	0.2229	13	A P A P A
Part B	B1	1	PartB	5	9	18	4	Y	0.2859	11	0.8786	10	A A P A P
Part B	B1	1	PartB	5	9	18	4	Y	0.0412	9	0.7062	9	A A A P P
Part B	B1	1	PartB	5	9	18	4	Y	0.4038	13	0.2188	11	A A P P A

Additional tabs in the results spreadsheet contain a list of execution notes from the program itself (such as date, time, input and output file names, execution times of various components, etc), and stores the last state of the Random Number Generator.

RESAMPLING AND SIMULATION

While resampling analyses are not often done, resampling and simulated randomization schedules are useful for testing as well. This allows another level of quality control, allows for testing of the quality of the schedule, and can be used for resampling tests.

By vectorizing the code, sampling is built into the algorithm in a way that generating thousands of simulations with no additional effort than generating only the target randomization. Furthermore, only the Random Number Generator Seeds Vary between simulations, and those seeds are chosen in a way using the Wichmann-Hill 2006 algorithm which guarantees non-overlapping sequences of random numbers.

During testing, tables of resampled allocations (see Table 3 below) allow for a simple visual inspection to confirm that distinct treatment assignments are generated, or more sophisticated statistical analyses; perhaps to confirm distributions of sequences. For example, in a small Phase 1 study in which cohorts have 2 sentinels randomized 1:1, followed by 6 other patients randomized 5:1, each cohort would only have 12 possible randomization schedules, which is problematic for resampling and permutation testing.

During production, the resampling and simulated tables may be used for resampling analyses (also known as rerandomization or permutations test) supporting the main study results. Resampling analyses

are not often used but can be a powerful tool to show that the study results reflect a treatment effect, and not the design of the study (e.g., in a resampling analysis, the study may be reanalyzed hundreds or thousands of times with random alternative treatment assignments and the distribution of the primary test statistics is analyzed). So, if the primary analysis returned, say, a p-value =0.03, about 30 out of every 1000 resampling analyses should show a p-value <0.03.)

In this sample (Table 4) each of columns 1 through 5 (and beyond) are an independent, identically distributed randomization schedule (see Table 3). Column 1 is the planned randomization schedule, however, any of the simulations in the additional Columns would serve equally well). In principle, one could measure each column for desired randomization properties (e.g., entropy, potential selection bias, excessive run lengths) and randomly pick a schedule from the ones that meet minimal requirements.

Table 4. Simulated Randomization.

idUnit	idStrata	winStrata	1	2	3	4	5
1	1	1	P	A	P	A	P
2	1	2	A	P	A	P	A
3	1	3	A	P	P	A	A
4	1	4	A	A	A	A	A
5	1	5	P	A	A	A	A
6	1	6	A	A	A	A	P
7	1	7	A	A	A	P	A

MEASURABILITY

Two natural objectives for randomization schemes are Randomness (essentially, unpredictability) and Balance (between treatment arms overall and within subgroups). In this hypothetical illustration below (Figure 2), by scoring both attributes one might be able to compare different randomization schemes and optimize the tradeoffs between both for the study at hand. Note that individual randomization schedules may vary greatly in the measures because they are randomly generated from a randomization scheme, so some care must be taken whether one wishes to optimize the 'average' schedule, or minimize the 'worst-case' schedule.

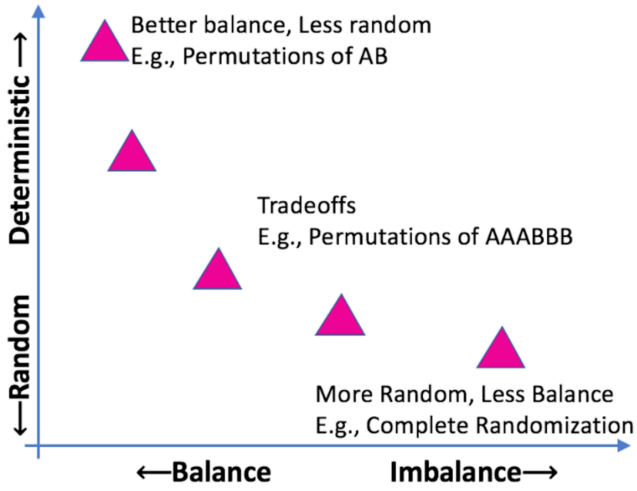

Figure 2. Idealized Plot of Randomness vs Balance

It is easy to see that these are somewhat contradictory objectives. A completely random scheme like tossing a coin for each treatment assignment is likely to drift away from the target treatment ratio, while

restricting the choices to impose balance between treatment arms and reduces the randomness. Calculations of Randomness and Balance depend heavily on the design of the study, especially attributes like stratification, treatment ratios, and blinding. If there are subgroups of special interest, Randomness and Balance are of concern within them as well, and being smaller, have a larger impact. Furthermore, since every schedule generated by a randomization scheme is random, measures of randomness and balance can vary greatly, so simulations are needed to generate point estimates and variability of the measures.

Randomness of a schedule is important to minimize allocation bias, selection bias, and to enable blinding of treatment to observers. Allocation bias is essentially a distribution of subjects among treatments such that another factor affects outcomes; since the existence of hidden variables can never be absolutely excluded, random assignment of treatments provides some assurance that a hidden (or known) factors will equally affect all treatment groups.

Selection bias implies the existence of an observer, who might anticipate the next treatment allocation and adjust his choice of subject accordingly. Blinding of treatment to observers helps ensure that patients in all treatment groups are treated identically regardless of treatment, as well as prevent observers from anticipating the next treatment based on past assignments. Often treatment is not blinded to observers, either because of study requirements, or accidently (e.g., because of distinct drug side effects). For the purposes of randomization metrics, we assume the worst case that observers have knowledge of treatment allocations. A more realistic refinement of this is that an observer's knowledge is restricted to subgroups of subjects (e.g., they may only know or guess the treatment assignments at their site).

Balance between treatments within subgroups is important to avoid confounding of treatment effect with the effects of prognostic covariates, maximize statistical efficiency, and provide face validity. For example, if drug and placebo were dispensed in a clinical trial so that most women received the drug, and most men received placebo, the effects of drug and sex could not be distinguished, the variance of the treatment estimate would be larger (reducing statistical power), and could result in misleading implications (e.g., that the drug causes elevated prolactin levels).

The graphs below assess the potential selection bias for 3 randomization constellations, in which an unblinded observer guesses that the next treatment allocated will be the one occurring least often up to that point, the Convergence Strategy from (Blackwell & Hodges, 1957). In these graphs below, the Black Line is the Average Probability of Guessing Correctly, while the Orange Region is the range of probability of Guessing Correctly.

In Figure 3, Potential Selection Bias, permutations of AABB, shows how one of the most common randomization approaches has a risk of selection bias because of the well know problem of every 4th assignment being determined by the previous 3 (while this is obviously an issue for Open Label site stratified studies, there is also possibility of unintentional unblinding due to side effects of the treatment).

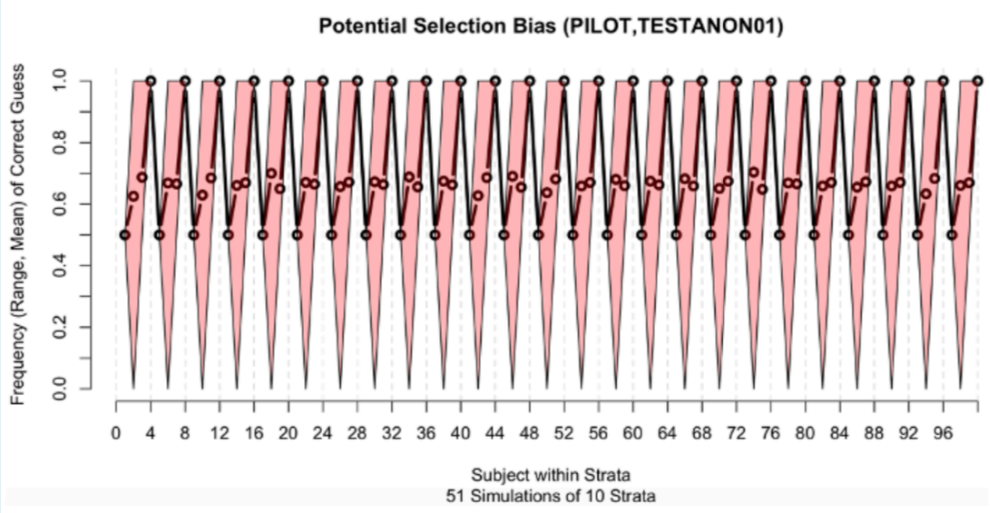

Figure 3. Potential Selection Bias, permutations of AABB

In Figure 4, Potential Selection Bias, permutations of AB, AABB, AAABBB, permuted block sizes vary randomly between 2, 4 and 6, which greatly reduces the potential selection bias, although one can guess even numbered assignments with a substantially better than chance.

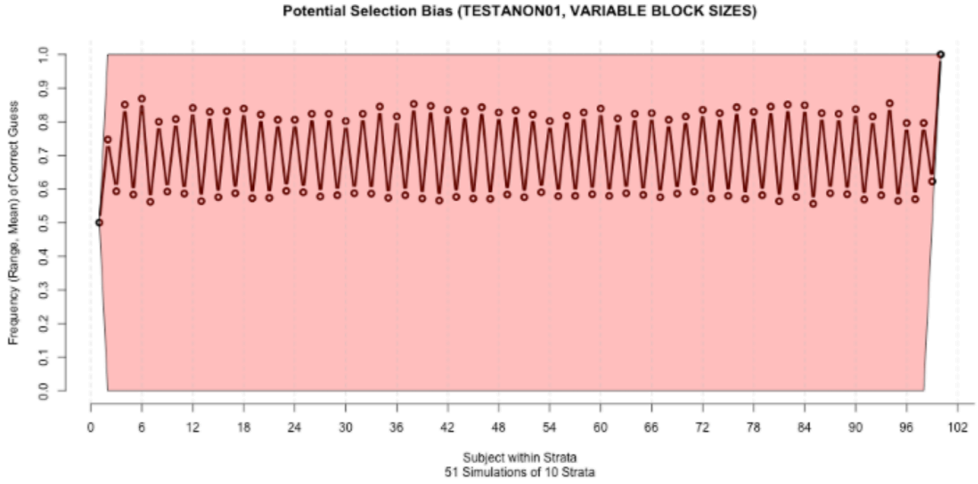

Figure 4. Potential Selection Bias, permutations of AB, AABB, AAABBB

In Figure 5 Potential Selection Bias, permutations of AB, AABB, AAABBB, plus AAB, ABB, blocks of size 3 are introduced to allow the balance to drift and confound anyone trying to use the Convergence Strategy. In this approach, the blocks are introduced in pairs, such as permutations of AAP paired with a permutation of APP, which will ensure treatment balance is restored by the end of the study.

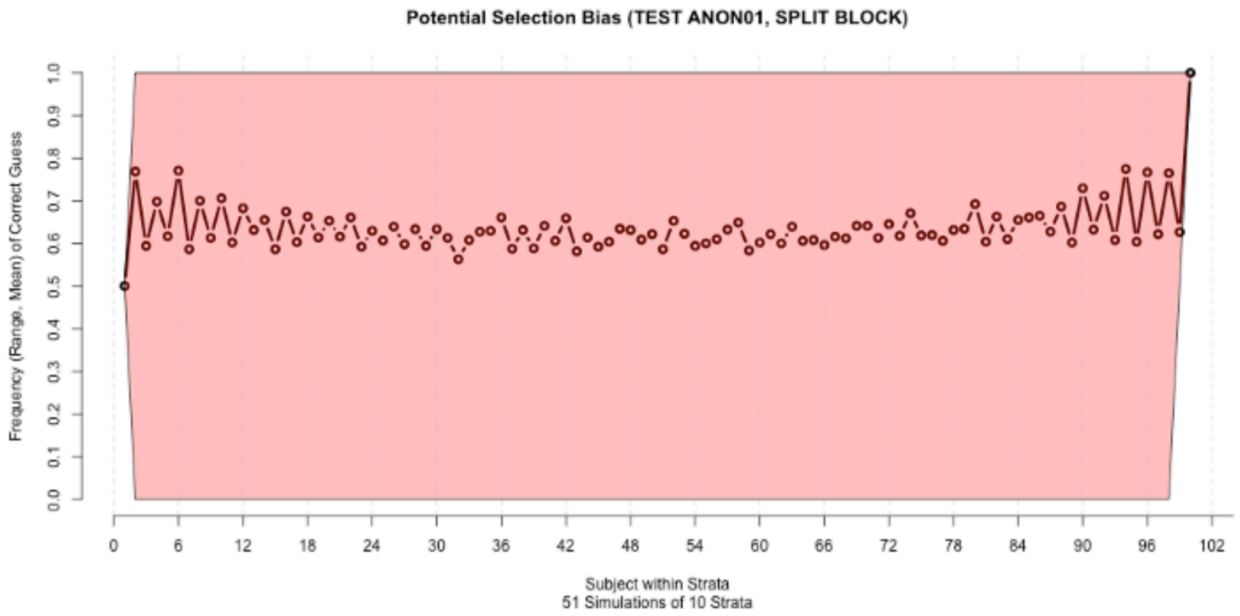

Figure 5. Potential Selection Bias, permutations of AB, AABB, AAABBB, plus AAB, ABB

RANDOM NUMBER GENERATION

The Random Number Generator (RNG) for the YPrime Randomizer is the Wichmann-Hill 2006 algorithm, motivated by these criteria:

- Pseudo-random number algorithm: This algorithm should ensure that starting with the same seed will result in the same sequence of random numbers.
- Transparency: the algorithm should be straightforward to understand from the code.
- Efficient Initialization: many random number generators require a run-in period after initialization to ensure a completely random sequence; this should be minimized.
- Parallelization: each resampling or simulation is generated from an independent identically distributed random sequence of random numbers, which are each independent of each other and of the number of simulations.
- Flexibility: flexible seeding procedures to meet client requirements (e.g., for maximum security parts of the RNG seed can come from different sources).

Other common criteria for random number generators are unneeded for this application, such as:

- Long periods before repeating. Even large clinical trials do not require more than a few thousand subjects.
- Cryptographic grade quality.
- Extreme precision. Since the algorithm is picking from a relatively small number of alternatives at each step, choices are almost entirely determined by the most significant digits of the random numbers.
- Unpredictability. While the internal state of some RNGs can be computed from the output stream of random numbers, it suffices to keep the keep the RNG output secure with the randomization schedule (or not store it all, since it can be regenerated with the seed).

Perhaps the most popular and powerful general-purpose random number generator used by default in many systems is the Mersenne-Twister algorithm. This has excellent properties for general use, such as a very long period before the random number sequence repeats (~2^{19937} -1 random numbers), is

computationally fast, and passes most common statistical tests for random number generators, such as the DIEHARD test suite (Marsaglia, 1985), and TestU01 tests (L'Ecuyer and Simard, 2005). However, in this application, the Mersenne-Twister has some drawbacks:

- The long period is unnecessary
- It uses a very large state variable consisting of 624 32-bit integers, which is cumbersome to store the final state (if needed)
- The original algorithm had some issues with initializing the algorithm, which while corrected in later versions, there are questions of how long of a burn-in period is sufficient.
- Adapting the algorithm to support parallel random number streams (for resampling) adds complexity to the algorithm and slows execution.
- In the R-implementation, the RNG is seeded with only one 32-bit integer, while the state variable is 624 32-bit integers (hence, a maximum of 2^{32} unique randomization schedules are possible).

R also provides easy access to other common random number generators, most notably the Wichmann-Hill 1982 algorithm which was the default in many applications (such as Python through version 2.2 or a variation in older versions of Microsoft Excel) until replaced by the Mersenne-Twister. This algorithm was designed to provide long periods using only 32-bit integer arithmetic (or 16-bit integers with additional code) by combining the output of three generators. Testing showed that it passed early versions of the DIEHARD suite, but not some tests in the TestU01 suite.

Wichmann-Hill published an updated version in 2006 which uses 4-32 bit integers as state variables, and 64 bit arithmetic (or 32 bit arithmetic with additional code). This version passed both the DIEHARD and TestU01 test suites and has a period of 2120 (1.329×10^{36}) random numbers before repeating. Most importantly for simulation applications, they provide an algorithm to generate state variable for independent sequences of random numbers which will not overlap "for over 2.3×10^{18} values at the very minimum" (Wichmann & Hill, 2006). The R-language has no problem with 64-bit arithmetic and the code is easily vectorized.

Furthermore, the algorithm can be seeded using all of the state variables, essentially 4 31-bit (binary), or 4 9-digit (base 10) integers. Consequently, for maximum security, each random seed can come from a different source, which would ensure that no single source could determine the randomization. For instance, one might come from the sponsor, YPrime might roll dice for another, while a 3rd could be the date and time stamp from a future document, and the 4th could be from an internet random number generator website.

CONCLUSION

The new YPrime Randomization system avoids a 'one-size-fits-all' solution, and strives to go beyond a 'fit-for-purpose' solution by making it easy to implement 'best practices' in study randomization appropriate to study designs. We do so: (1) By building flexibility into the system with a parametrically driven core program augmented with some custom programming; (2) By making all algorithms (including the random number generator) fully transparent with audit functions of all calculations; (3) Adding Resampling and Simulation capabilities for both testing purposes, and to support rerandomization analyses; (4) Adding tools to evaluate randomization strategies by objective measurements. With this system in place, we plan to expand functionality further with a wider variety of algorithms than permuted blocks (such as urn or bias coin), and implementation as an API (which could support covariate adjusted randomization).

REFERENCES

Blackwell, D. and J.Hodges Jr (1957). Design for the control of selection bias. Ann Math Stat 28, 449-460

Rosenberger W, Lachin J (2016). *Randomization in Clinical Trials: Theory and Practice.* Wiley Series in Probability and Statistics. Wiley. ISBN 9781118742242

Sweitzer, D (2013). Randomization Metrics: Jointly Assessing Predictability and Efficiency Loss in Covariate Adaptive Randomization Designs. Proceedings of the 2013 Joint Statistical Meeting (DOI: 10.13140/RG.2.1.3590.6329)

Uschner D, Schindler D, Hilgers R, Heussen N (2018). "randomizeR: An R Package for the Assessment and Implementation of Randomization in Clinical Trials." *Journal of Statistical Software,* 85(8), 1-22. doi: 10.18637/jss.v085.i08 (URL: https://doi.org/10.18637/jss.v085.i08)

Wichmann, B.A., Hill, I.D., 2006. Generating good pseudo-random numbers. *Computational Statistics and Data Analysis* 2006.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Dennis Sweitzer, Ph.D.
YPrime
dsweitzer@yprime.com
Dennis-Sweitzer.com