Paper SM08

InnoRand - Innovative Randomization

Yinfei Wu, Boehringer Ingelheim Pharmaceuticals, Inc., Ridgefield, U.S.A
Ivo Marek, Entimo AG, Berlin, Germany

ABSTRACT

Randomization is the foundation for effectiveness research and is used as a basis for submissions of regulatory dossiers in request of marketing authorization for new drugs. In this context, the choice of the randomization design plays an important role. A popular choice is the permuted block design that controls imbalance by making treatment assignments according to a specified allocation ratio at random within blocks. However, the consistent imbalance control increases the risk of selection bias and prediction of upcoming allocations. Better alternatives like the block urn design, the big stick design and others have been developed, and are recently gaining attention. Nevertheless a GxP compliant tool remains hard to be found. We present the conceptual implementation of an R Shiny application which generates randomization lists. InnoRand implements new approaches in addition to the well established randomization methods.

INTRODUCTION

An ideal randomization procedure would maximize the statistical power, minimize selection bias and minimize confounding, see Lachin (1988). Generally, statistical power is achieved by equal group sizes. However, in some situations unequal group sizes may be more powerful. Selection bias may occur if investigators can consciously or unconsciously preferentially enroll patients between treatment arms. A good randomization procedure will be unpredictable so that investigators cannot guess the next subject's group assignment based on prior treatment assignments. The risk of selection bias is highest when previous treatment assignments are known or future assignments can be guessed.

No single randomization procedure meets the properties 'balancing' and 'selection bias control' simultaneously. For example, when block randomization is used, small block sizes lead to balanced groups but increase the risk for selection bias, while larger blocks are better to control selection bias but increase the chance of unbalanced groups. Despite the criticism of tried and prevalent randomization procedures, novelties of the last decades have hardly found their way into clinical practice as mentioned by Uschner et.al. (2018). Berger, Bejleri, and Agnor (2016) remark that clinicians stick with randomization procedures that are easily available, although their poor properties have been widely discussed and better alternatives have been proposed, see Sverdlov et.al. (2023).

randomizeR is the first package that assists the user in choosing a randomization procedure based on scientifically sound criteria, Uschner et.al. (2018). It allows the generation of randomization sequences with different procedures and the assessment of those. A main advantage is that it has a common interface for all randomization procedures. Nevertheless, it has also some drawbacks. For example, the randomized permuted block design is implemented such that prior to the treatment assignments within each block all block sizes are randomly drawn. This makes it impossible to enlarge a given randomization sequence at a later date by using the same seed, however this flexibility in trial adaptation may be desirable when the initial number of enrolled subjects turns out to be too small to achieve the desired statistical power. Alternatively, the randomization sequence can be consistently enlarged when block sizes and treatments are selected in alternation, thereby generating the randomization list in a stepwise fashion. Furthermore, the treatments in randomizeR are drawn via a simple random sample within each block. The more natural way is to draw the treatment patterns instead of each treatment itself leading to less draws. Another drawback is that the block urn design proposed by Zhao and Weng (2011) is not available in randomizeR, although it performs strongly on the desirable measures of consistent imbalance control and lower probabilities of deterministic assignments.

There are two objectives of this presentation: 1. to illustrate an efficient way to draw treatment patterns within the permuted block design; 2. To demonstrate how the block urn design can be implemented in R.

PERMUTED BLOCK DESIGN

The permuted block design involves randomizing patients to treatment groups in sequential blocks. In its simplest form it is based on M blocks with a constant block size, two treatments (say '1' and '2') and m patients per treatment within each block leading to a total sample size 2mM. Furthermore, within each block, a permutation of size 2m is randomly selected such that m patients are assigned to each treatment. Hence, the first block is used for the first set of 2m patients and so on up to block M for the last set of 2m patients. For example, in a design where the block size is four, there are six possible ways to make treatment assignments for a block: '1122', '2211', '1212', '2121', '1221', and '2112'.

This method is implemented in the R package randomizeR. However, there is an important point that needs to be considered by using this R package, namely, that for each block a simple random sample of treatments and of size 2m will be drawn. The natural course of action is to draw the pattern itself instead of treatments.

The following example illustrates how this can be easily performed using R. Using the example above restricted to one block an efficient iterator for permutations and combinations is implemented in the `iterpc` package. Three steps are needed: 1. create an iterator, 2. get all the permutations and 3. draw from the set of all permutations. The iterator is created by invoking `iterpc::iterpc(c(2,2), ordered = TRUE)` and assigning the iterator to `I`. `iterpc::getall(I)` returns the ordered list of all permutations, say `perms`. Using `perms[sample(1:6,1),]` a random pattern is drawn in a last step. This can be repeated until M blocks are allocated.

However, in practice very often there is more than one treatment leading to block sizes larger than 4. Let m[a], a = 1, 2, ..., A, denote the number of patients which should be assigned to treatment a = 1, 2, ..., A. Hence, the block size equals the sum of m, i.e. n = sum(m), and the number of different permutations equals mul(n, m) = n!/(m[1]!*m[2]!* ... * m[A]!). For example, if m = (3,4,2,3,5) there are 1715313600 different permutations, i.e. > 2^{30} and < 2^{31-1} (`.Machine$integer.max`) and `iterpc::getall(I)` will allocate more than 100 Gb memory causing an R error.

One possible solution is to express the multinomial coefficient as a product of binomial coefficients: mul(n, m) = bin(n, m[1]) * bin(n-m[1], m[2]) * ... * bin(m[A], m[A]). Let s[a] denote the sum of m[a+1], m[a+2], ..., m[A] for a < A and s[A] = 0, and bin[a] = bin(s[a]+m[a], m[a]). Moreover, let w[a] denote the product of bin[a+1], bin[a+2], ... , bin[A] for a < A.

By executing `iterpc::iterpc(c(m[a],s[a]), ordered = TRUE, labels = c(a,0))` for a = 1, 2, ..., A-2 and `iterpc::iterpc(c(m[a],s[a]), ordered = TRUE, labels = c(a,a+1))` for a = A-1 the iterators are assigned to the list `I`. After all iterators are created, the next step is to create all lists of the permutations, i.e. `lapply(I, iterpc::getall)`, and assign them to the list `perms`. Each permutation contains the values a and 0 or A-1 and A, respectively. That is, the overall number of generated permutations is bin[1]+bin[2]+...+bin[A-1].

For the generation of one block the following procedure is applied. First, R[1] is selected randomly from the set of all permutations {0, 1, 2, ..., (mul(n,m)-1)}. Second, from R[1] the subsequent numbers are calculated using R[a+1] = R[a] mod w[a] for a = 2, ..., A and X[a] = 1 + (R[a] - R[a+1]) / w[a] for a = 1, 2, ..., A-1. Third, from the numbered list of permutations `perms`, the X[A-1]-th permutation is selected and assigned to the variable `block` by `perms[[A-1]][X[A-1],]`. Finally, for a = A-2, A-3, ..., 1 the zeros are successively replaced with the treatments, i.e. `block <- replace(perms[[a]][X[a],], perms[[a]][X[a],]==0, block)` is repeated.

The following example illustrates the above described procedure. The binomial coefficients for m = (3,4,2,3,5) are bin(17,3) = 680, bin(14,4) = 1001, bin(10,2) = 45, bin(8,3) = 56 and bin(5,5) = 1 and the factors are w[1] = 2522520, w[2] = 2520, w[3] = 56 and w[4] = 1. w[5] is not defined. Let say that R[1] = 1071253633 is randomly selected. The remainders are R[2] = 1705153, R[3] = 1633, R[4] = 9 and R[5] = 0. This leads to the selection of the 425-th permutation, 677-th permutation, 30-th permutation and the 10-th permutation (X[1] to X[4]) of the respective list of permutations. The selected permutations are

```
0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0,
0 0 0 2 2 2 0 0 0 0 0 2 0 0,
0 0 0 3 0 0 0 0 0 3 and
4 5 4 5 5 5 4 5
```

In the reverse order the zeros are replaced. The zeros in `0 0 0 3 0 0 0 0 0 3` are replaced by `4 5 4 5 5 5 4 5` leading to `4 5 4 3 5 5 5 4 5 3`. Then the zeros in `0 0 0 2 2 2 0 0 0 0 0 2 0 0` are replaced by `4 5 4 3 5 5 5 4 5 3` leading to `4 5 4 2 2 2 3 5 5 5 4 2 5 3`. And the final sequence is `4 5 4 2 1 2 2 3 1 1 5 5 5 4 2 5 3`.

For randomized permuted blocks, it is straightforward. The block sizes are a multiple of n, i.e. nb for b in a set of integers B. Thus, for each block size b and treatment a, w[a,b] denote the product of bin[a+1,b], bin[a+2,b], ... , bin[A,b] for a < A where bin[a,b] = bin(b*(s[a]+m[a]), b*m[a]).

By executing `iterpc::iterpc(c(b*m[a],b*s[a]), ordered = TRUE, labels = c(a,0))` for a = 1, 2, ..., A-2, and `iterpc::iterpc(c(b*m[a],b*s[a]), ordered = TRUE, labels = c(a,a+1))` for a = A-1 the iterators are assigned to the list entries `I[[b]][[a]]`. After all iterators are created, the next step is to create all lists of the permutations, i.e. `lapply(I, function(b) lapply(b, function(i) if (length(i)!=0) iterpc::getall(i))`, and assign them to the list `perms`. Finally, after Y is selected randomly with probability 1/#B from B, R[1] is selected randomly from the set {0,1,2,...,(mul(Y*n,Y*m)-1)} and the above process for filling the block is processed on `perms[[Y]]`.

BLOCK URN DESIGN

The block urn design was proposed by Zhao & Weng (2011). It has consistent imbalance control, lower probabilities of deterministic assignments and is generally applicable to trials with two or more treatments with or without balanced allocations. Zhao & Weng (2011) note that deterministic assignment could occur, but the timing of such occurrences

is random, making correct prediction less likely to occur. Currently, no R package provides an implementation of the block urn design, although it can be easily implemented.

Let m[a], a = 1, 2, ..., A, denote the allocation ratios. The allocations ratios are all integers with the greatest common divisor of 1. Moreover, let b an positive integer. The block size is nb, where n = sum(m). m[a]*b, a = 1, 2, ..., A is the target allocation for one block.

Starting with u[a] = 0, a = 1, 2, ..., A, and v = 0, the following steps are repeated until one block is filled. First, the conditional probabilities p[a] = m[a]b - u[a] / (nb - v) for a = 1, 2, ..., A are calculated. Note that at start the probabilities are p[a] = m[a]b/nb. Second, a sample of size one is drawn, i.e. `X <- sample(c(1, 2, ..., A), prob = p)` and appended to the randomization list. u[X] as well as v is increased by one. In a third step u is compared with m and if u[a] >= m[a] for all a: u and v will be updated by u - m and v - n, respectively. Note that the conditional probabilities reflect the current imbalance.

For example, let m = (3,4,2,3,5) and b = 2 with target allocation mb = (6,8,4,6,10) and block size nb = 34. Suppose the current sequence is `5 2 5 3 2 5 2 3 5 4 1 2 4 4 5 2 1 5 5` (length 19), then u equals (2,5,2,3,7). When during the next iteration X = 1 is drawn u becomes (3,5,2,3,7) and all entries are no longer less than the entries of m. In this case u and v are updated, i.e. u = u-m = (0,1,0,0,2), and v = v-n = 3. The next part is created, by starting with the current u and v. When u equals (3,7,2,5,7) and v equals 24, the sequence becomes: `5 2 5 3 2 5 2 3 5 4 1 2 4 4 5 2 1 5 5 1 2 5 5 4 4 5 3 1 2 2 5 4 2 5 4 3 2 4 2 1 1` The sequence length is now 41, v = 7 and u = (0,3,0,2,2). The length of the sequence is not 44 because one of treatment 2 and two of treatment 5 are 'carried forward'.

The following R function illustrates the application of the procedure:

```
blockUrnRand <- function(N, b = 4L, A = 2L, m = rep(1L, A)) {

    n <- sum(m)
    bm <- b*m
    bn <- b*n

    u <- integer(A)
    v <- 0L
    l <- 0L
    s <- integer(0L)

    while (l < N) {

        p <- (bm-u) / (bn-v)
        a <- sample(1L:A, 1L, prob = p)

        u[a] <- u[a] + 1L
        v <- v + 1L
        s <- c(s, a)
        l <- l + 1L

        if (all(u >= m)) {
            u <- u-m
            v <- v-n
        }
    }

    s
}
```

CONCLUSION

The presentation covers only a part of a new randomization library. It is planned to extend the library by the latest state of the art randomization methods. All randomization methods will be embedded in a GxP compliant R Shiny application so that randomization lists can be created user-friendly without knowledge about R.

REFERENCES

[1] Lachin JM (1988). Statistical properties of randomization in clinical trials, Controlled Clinical Trials. 9(4): 289–311.
[2] Uschner D, Schindler D, Hilgers RD, 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.

[3] Berger VW, Bejleri K, Agnor R (2016). Comparing MTI Randomization Procedures to Blocked Randomization. Statistics in Medicine 35(5). 685-694.
[4] Sverdlov O, Carter K, Hilgers R-D, Everett CC, Berger VW, Luo YA, Chipman JJ, Ryeznik Y, Ross J, Knight R, Yamada K (2023). Which Randomization Methods Are Used Most Frequently in Clinical Trials? Results of a Survey by the Randomization Working Group. Statistics in Biopharmaceutical Research.
[5] Zhao W, Weng Y (2011). Block urn design — A new randomization algorithm for sequential trials with two or more treatments and balanced or unbalanced allocation. Contemporary Clinical Trials 32. 953–961.

ACKNOWLEDGMENTS

We would like to express our deepest appreciation to Kerstine Carter and Louis J. Bour for supporting our work and all the helpful comments. We would also like to extend our deepest gratitude to the METEOR (Methods and Theory of Randomization) team at Boehringer Ingelheim, especially Annika Scheffold and Johannes Krisam who validated all the randomization procedures.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:
Yinfei Wu
Boehringer Ingelheim Pharmaceuticals, Inc.
900 Ridgebury Road
Ridgefield CT 06877

Email: yinfei.wu@boehringer-ingelheim.com
Web: www.boehringer-ingelheim.com