Along with preparing power analyses and statistical analysis plans (SAPs), generating study randomization lists is something a practicing biostatistician is occasionally asked to do. While not a particularly interesting activity, it offers the opportunity to tackle a small programming challenge. The title is a little misleading because you should probably skip all this and just use the `blockrand` package if you want to generate randomization schemes; don't try to reinvent the wheel. But, I can't resist. Since I was recently asked to generate such a list, I've been wondering how hard it would be to accomplish this using `simstudy`. There are already built-in functions for simulating stratified randomization schemes, so maybe it could be a good solution. The key element that is missing from simstudy, of course, is the permuted block setup.

Why permuted block randomization?

What is *permuted block* randomization and why even bother? The *block* part indicates that patients will be randomized in sub-groups. If we have blocks of size two, patients will be paired sequentially, with each patient randomized to different arms. This helps with balance over time and when recruitment is less than anticipated. If we were to randomize 100 patients to treatment and control (50 in each arm), it is possible that treatment assignments will cluster close to the start of the trial, just by chance. If the time period is related to the outcome in some way, this would be undesirable. Furthermore, if trial recruitment lagged and had to stop early, there would be an actual lack of balance across the arms.

The argument for block randomization seems strong enough. But if coordinators in the field know that we are using this approach, there is a risk of influencing patient recruitment. If it is known that patients are randomized in blocks of size four, and the first two patients are randomized to drug therapy, the coordinator will know that the next two patients will be randomized to control. This could influence who the coordinator recruits into the study, particularly if they believe drug therapy is superior. They may actively or unconsciously recruit healthier patients when it is known that they are going to get the control. (This, of course, is much less of an issue when recruiters/coordinators are completely blinded to group assignment.) By changing the block sizes in an unpredictable manner, by *permuting* the sizes, this problem is solved. Hence, *permuted block randomization*.

simstudy code

I want to walk through the code that will generate permuted block randomization. In this scenario we are conducting a trial to compare a *drug* therapy with *placebo* in at least 120 patients. We would like to randomize within blocks of size two or four, and the order of the blocks will themselves be randomized. We assume that each block size will have equal probability of being selected, though balance across different block sizes is not guaranteed. The preliminary code shown here implements these specifications:

```
library(simstudy)
library(data.table)

set.seed(1234)

n <- 120
levels <- c("D", "P")     # Drug/Placebo
blk_sizes <- c(2, 4)

n_arms <- length(levels)
```

```
p_blk <- 1/length(blk_sizes)
```

The first step is to generate a sequence of blocks with varying block sizes. We take advantage of the `mixture` distribution option in `simstudy` to generate blocks. This distribution is specified using a string with the format "\(x_1|p_1 + \dots + x_k|p_k\)". In this case, there are \(k=2\) block sizes; \(x_1 = 2\), \(p_1 = 0.5\), \(x_2 = 4\), and \(p_2 = 0.5\). We construct the mixture formula using the predefined block sizes, and use that formula to define the data that we need to generate:

```
v <- sapply(blk_sizes, function(x) paste(x, p_blk, sep = "|"))
mixformula <- paste(v, collapse = "+")

def <- defData(varname = "blk_size", formula = mixformula,
  dist = "mixture", id = "blk_id")

def
##     varname     formula variance    dist     link
## 1: blk_size 2|0.5+4|0.5        0 mixture identity
```

Now, we need generate enough blocks to support the target number of patients to be randomized; that is, the sum of the block sizes should at least as large as the target. If all block sizes were the minimum block size (in this case \(2\)), then we would need at least \(n/2\) blocks. Clearly, we will need fewer, but we will start with \(n/2\) here and remove the excess:

```
maxblocks <- ceiling(n / min(blk_sizes))
dd <- genData(maxblocks, def)

#--- removing the excess

nblocks <- dd[, threshold := (cumsum(blk_size) >= n) * .I][threshold >
0]
dd <- dd[1:nblocks[1, threshold]]

tail(dd)
##    blk_id blk_size threshold
## 1:     36        4         0
## 2:     37        2         0
## 3:     38        2         0
## 4:     39        4         0
## 5:     40        4         0
## 6:     41        4        41
```

In the final step, we use cluster data generation to create the individual patients, defining each block as a cluster. Treatment assignment is stratified by each block:

```
di <- genCluster(dd, cLevelVar = "blk_id", numIndsVar = "blk_size",
level1ID = "id")
dtrt <- trtAssign(di, nTrt = n_arms, strata = "blk_id", grpName =
"arm")

dtrt <- dtrt[, .(id, blk_id, blk_size, arm = factor(arm, labels =
levels))]
```

Here are examples of the block randomization results for four blocks:

```
dtrt[blk_id == 5]
##    id blk_id blk_size arm
## 1: 15      5        4   P
## 2: 16      5        4   D
## 3: 17      5        4   P
## 4: 18      5        4   D
dtrt[blk_id == 8]
##    id blk_id blk_size arm
## 1: 25      8        2   D
## 2: 26      8        2   P
dtrt[blk_id == 19]
##    id blk_id blk_size arm
## 1: 59     19        2   P
## 2: 60     19        2   D
dtrt[blk_id == 26]
##    id blk_id blk_size arm
## 1: 73     26        4   D
## 2: 74     26        4   P
## 3: 75     26        4   P
## 4: 76     26        4   D
```

A real-world application

I've created a function `blkRandom` based on this code so that I can illustrate this functionality in a more realistic setting. In a current multi-site study that I'm working on (already did the power analysis, finalizing the SAP), we need to provide a randomization list so that subject recruitment can begin. Randomization will be stratified by each of the sites (1 through 7), by sex (M and F), and by location of recruitment (A or B); in total, there will be $7\times 2 \times 2 = 28$ strata. For each of the 28 strata we want to randomize 50 potential subjects using permuted block randomization; for particular strata, this is certainly too large a number, but it doesn't hurt to overestimate as long as the clinical trial software system can handle it.

Here is how the function would work for a single strata (just showing the first and last blocks):

```
blkRandom(n = 50, levels = c("A", "B"), blk_sizes = c(2, 4))[c(1:4,
47:50)]
##    blk_id blk_size threshold id arm
## 1:      1        4         0  1   A
## 2:      1        4         0  2   B
## 3:      1        4         0  3   A
## 4:      1        4         0  4   B
## 5:     13        4        13 47   B
## 6:     13        4        13 48   A
## 7:     13        4        13 49   A
## 8:     13        4        13 50   B
```

Here is a wrapper function for `blkRandom` that incorporates a specific strata s. This will enable us to do permuted block randomization within different subgroups of the population, such as males and females, or sites:

```
sBlkRandom <- function(s, n, levels, blk_sizes) {

  dB <- blkRandom(n, levels, blk_sizes)
```

```
  dB[, .(id = paste0(id, s), stratum = s, arm)]

}

sBlkRandom(s = "M1A", n = 30, levels = c("A", "B"), blk_sizes = c(2,
4))[1:5]
##      id stratum arm
## 1: 1M1A     M1A   B
## 2: 2M1A     M1A   A
## 3: 3M1A     M1A   A
## 4: 4M1A     M1A   B
## 5: 5M1A     M1A   B
```

All the pieces are now in place.

We need to create a list of strata, each of which requires its own permuted block randomization list:

```
library(tidyr)

#--- specify all strata variables

site <- c(1 : 7)
sex <- c("M", "F")
location <- c("A", "B")

#--- create strata

strata <- expand.grid(sex = sex,site = site, location = location)
strata <- unite(strata, "stratum", sep = "")$stratum

head(strata)
## [1] "M1A" "F1A" "M2A" "F2A" "M3A" "F3A"
```

With the list of strata in hand - we create the randomization lists using `lapply` to repeatedly call `sBlkRandom`:

```
rbindlist(lapply(
    strata,
    function(s) sBlkRandom(s, n = 50, levels = c("A", "B"), blk_sizes =
c(2, 4))
  )
)
##           id stratum arm
##    1:  1M1A     M1A   B
##    2:  2M1A     M1A   A
##    3:  3M1A     M1A   A
##    4:  4M1A     M1A   B
##    5:  5M1A     M1A   B
##   ---
## 1420: 46F7B     F7B   B
## 1421: 47F7B     F7B   B
## 1422: 48F7B     F7B   A
```

```
## 1423: 49F7B     F7B   B
## 1424: 50F7B     F7B   A
```

Going with the tried and true

This has been fun and I think successful, but as I mentioned, you might want to stick with the established blockrand package that is designed around this very specific goal. I have written a simple wrapper function that rectifies one slightly minor shortcoming (block sizes are specified as blk_size\(/ 2\)) and allows us to use `lapply` to make repeated calls across the strata:

```
library(blockrand)

rand_stratum <- function(stratum, n, levels, blk_sizes) {

  blk_sizes <- blk_sizes / 2

  dB <- data.table(blockrand(
    n = n,
    num.levels = length(levels),
    levels = levels,
    id.prefix = stratum,
    block.prefix  = stratum,
    stratum = stratum,
    block.sizes = blk_sizes)
  )

  dB[, .(id, stratum, treatment)]

}

rbindlist(lapply(strata, function(s) rand_stratum(s, 50, c("A", "B"),
c(2, 4))))
##          id stratum treatment
##    1: M1A01     M1A         B
##    2: M1A02     M1A         A
##    3: M1A03     M1A         B
##    4: M1A04     M1A         A
##    5: M1A05     M1A         B
##   ---
## 1420: F7B46     F7B         A
## 1421: F7B47     F7B         A
## 1422: F7B48     F7B         B
## 1423: F7B49     F7B         A
## 1424: F7B50     F7B         B
```

In case the elegance and simplicity (not to mention all the other features that it provides, but I didn't show you) are not compelling enough, the speed comparison isn't even close: `blockrand` is 7 times faster than my `simstudy` solution (albeit on the millisecond scale - so you might not actually notice it).

```
library(microbenchmark)

microbenchmark(
```

```
 rbindlist(lapply(strata, function(s) sBlkRandom(s, 50, c("A", "B"),
c(2, 4)))),
 rbindlist(lapply(strata, function(s) rand_stratum(s, 50, c("A", "B"),
c(2, 4))))
)
## Unit: milliseconds
##
expr
##    rbindlist(lapply(strata, function(s) sBlkRandom(s, 50, c("A",
"B"), c(2, 4))))
##  rbindlist(lapply(strata, function(s) rand_stratum(s, 50, c("A",
"B"), c(2, 4))))
##  min  lq mean median  uq max neval cld
##  276 290  331    304 344 698   100   b
##   38  40   48     44  49 120   100  a
```