Riddler Lottery Search

Ami Wulf, PhD Student, UCLA Statistics

11/22/2019

This Document details the thought process I used to answer the FiveThirtyEight Riddler Classic for November 22, 2019 (https://fivethirtyeight.com/features/can-you-decode-the-riddler-lottery).

Primary Problem:

Five friends with a lot in common are playing the Riddler Lottery, in which each must choose exactly five numbers from 1 to 70. After they all picked their numbers, the first friend notices that no number was selected by two or more friends. Unimpressed, the second friend observes that all 25 selected numbers are composite (i.e., not prime). Not to be outdone, the third friend points out that each selected number has at least two distinct prime factors. After some more thinking, the fourth friend excitedly remarks that the product of selected numbers on each ticket is exactly the same. At this point, the fifth friend is left speechless. (You can tell why all these people are friends.)

What is the product of the selected numbers on each ticket?

We start by constructing a dataset with numbers that satisfy the 2nd and 3rd friends' observations

Start with all numbers from 1 to 70

```r
non_primes <- data.frame(number = 1:70)

#prime factorization code taken from
#https://stackoverflow.com/questions/49974601/r-prime-factorization
find_factorization=function(x){
  n=c()
  i=2
  r=x
  while(prod(n)!=x){
    if(!r%%i) {
      n=c(n,i)
      r=r/i
      i=1
    }
    i=i+1
  }
  n
}

#Find the factorization of each number, the unique factors in each,
#and the number of unique factors
non_primes$factorization <- sapply(non_primes$number, find_factorization)
non_primes$length <- sapply(non_primes$factorization, length)
non_primes$unique_factors <- sapply(non_primes$factorization, unique)
non_primes$num_unique <- sapply(non_primes$unique_factors, length)
```

```r
#subset to numbers with 2+ unique factors, a la the 3rd friend's observation,
#which also excludes the primes, a la the 2nd friend's observation
non_primes <- non_primes[non_primes$num_unique >= 2,]
#This leaves 41 numbers.

table(unlist(non_primes$factorization))
```

```
##
##  2  3  5  7 11 13 17 19 23 29 31
## 46 26 13  8  5  4  3  2  2  1  1
```

Given that each number was only picked once, all of their sets of 5 numbers have the same product, and that product (like all numbers) can be uniquely expressed as a product of primes, we know that any prime that exists in the prime factorization of one picked set of 5 numbers must exist in the other set of 5 numbers. So we can remove all numbers with a prime that does not show up in at least 5 different numbers. In particular, numbers with 13, 17, 19, 23, 29, and 31 in their factorization must be removed

```r
#expression in the sapply picks those rows without (!) any (max) successful
#finding (%in%) of those primes
non_primes <- non_primes[sapply(non_primes$factorization, function(x)
                         (!max(c(13, 17, 19, 23, 29, 31) %in% x))), ]
table(unlist(non_primes$factorization))
```

```
##
##  2  3  5  7 11
## 36 22 12  8  5
```

The leaves 28 potential numbers. Given that the problem implies that there is only one valid product, we must be able to lower these 28 to 25 numbers. Just as we wanted to remove primes that had fewer than 5 instances, we also must remove primes that do not have a multiple of 5 instances. We see that the three we remove must have a total of three 7s in their prime factorization to lower the number of 7s to 5. The same goes for two 5s, so we can start off by removing numbers with 7 and 5 in their prime factorization.

```r
non_primes <-  non_primes[!(non_primes$number %in% c(35,70)),]
table(unlist(non_primes$factorization))
```

```
##
##  2  3  5  7 11
## 35 22 10  6  5
```

Finally, we are left with one number to remove, and the prime factor list has two 3s and one 7 above multiples of 5 of each. Let's remove 63.

```r
non_primes <-  non_primes[!(non_primes$number == 63),]
table(unlist(non_primes$factorization))
```

```
##
##  2  3  5  7 11
## 35 20 10  5  5
```

Great, we have the prime factors in the remaining 25 numbers, and can infer an individual 5-number group – and the product of interest – by dividing by five. Each 5-group will be made up of the following: seven 2s, four 3s, two 5s, one 7, and one 11. This gives us the answer to the central question:

$$2^7 * 3^4 * 5^2 * 7^1 * 11^1 = 19{,}958{,}400$$

Extra credit:

How many different ways could the friends have selected five numbers each so that all their statements are true?

Now that we know the prime factorization of each draw of 5 numbers, we can (1) search for valid such draws, and then (2) valid groups of 5 draws of 5 numbers. The former task will be completed by winnowing down draws of 5 numbers to those that qualify. The latter would take far too long to run (running a series of several operations and checks over a billion times) so we'll do a sort of depth-first search, attempting to stop reaching the vast majority of those checks. First, the search for valid draws of 5 numbers:

```r
#Gets a list of valid quintets of numbers
find_valid_5groups <- function(){
  #combinatorics function getting all 5-groups of 25 numbers
  combinations <- combn(25, 5)
  combinations <- subset_correct_total_length(combinations)
  combinations <- subset_correct_product(combinations)
  #return valid 5-groups
  return(combinations)
}

#Check each group of 5 numbers to see if they have a total of 15
#prime factors. A quick time-saver to avoid multiplying 5 numbers and
#checking the result for each combination
subset_correct_total_length <- function(combinations){
  lengths <- apply(combinations, 2, function(x) sum(non_primes[x,]$length))
  return(combinations[,lengths == 15])
}

#Checking each group of 5 numbers to see if their product is our product
#of interest
subset_correct_product <- function(combinations){
  products <- apply(combinations, 2, function(x) prod(non_primes[x,]$number))
  return(combinations[,products == 19958400])
}

combinations <- find_valid_5groups()
ncol(combinations)
```

```
## [1] 275
```

275 different draws of 5 result in the correct product. The first few columns (first 3 of 275 draws of 5 numbers) of the data structure looks like this:

```r
combinations[,1:3]
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    5    6    6
## [3,]   23   20   22
## [4,]   24   23   23
## [5,]   25   25   24
```

. . . which doesn't make much sense since we replaced the actual numbers with placeholders between 1 and 25. But we can switch back to interpretable draws easily:

```r
apply(combinations[,1:3], 2, function(x) non_primes[x,]$number)
```

```
##      [,1] [,2] [,3]
## [1,]    6    6    6
## [2,]   15   18   18
## [3,]   56   50   55
## [4,]   60   56   56
## [5,]   66   66   60
```

Each of these groups of 5 numbers multiply to 19,958,400, but clearly there are multiple overlaps between even these first three of 275.

Therefore, we have to search for draws of all 25 numbers for which each group of 5 is in our valid group of 275. Checking each potential grouping for validity would take far too long, so we'll create a custom process to cut the number of checks down. The function effectively cuts the vast majority of the checks out of contention before we even get to them.

```r
#place the 275 valid combinations of 5 numbers into a list
combs <- split(combinations, rep(1:ncol(combinations), each = nrow(combinations)))
working_5_5_divisions <- NULL

#start the search
find_working_5_5_divisions <- function(combs, divs){
  working_5s_list <- NULL
  divs <- add_another_5(working_5s_list = working_5s_list,
                        remaining_combs = combs, divs = divs)
  return(divs)
}

#For each of the combinations that remain valid when the function is
#called, and consideringthe groups of 5 that we've already selected in
#this depth-first search, recurse on any new group of 5 that doesn't
#overlap any of the already-selected groups.
add_another_5 <- function(working_5s_list, remaining_combs, divs){
  n <- length(remaining_combs)
  #for each of the remaining combinations of 5
  for(i in 1:n){
    #add it (the name of the combination) to the working list
    working_5s_list <- c(working_5s_list, names(remaining_combs[i]))
    #if that makes a set of 5 groups of 5, add that valid draw to a list
    #of valid draws
    if(length(working_5s_list) == 5){
      divs <- cbind(divs, working_5s_list)
      #and exit the for loop (as there can only be one valid 5th group of
      #5 for a given first 4 groups of 5)
      return(divs)
    }
    if(i < n){
      #find a list of the combinations that would be valid if the current
      #combination was added to the working list of combinations. This is
      #decided by checking each of the remaining combinations for intersecting
      #numbers in the current combination and accepting it as valid if there are
      #no intersecting numbers
      non_intersecting <- c(rep(FALSE, i), sapply(remaining_combs[(i+1):n], function(x)
        length(base::intersect(x, remaining_combs[[i]])) == 0))
      #If any such valid combinations exist, recurse with those valid combinations
      #as the new valid list, and with the working group list now including this
      #new appended group
      if(sum(non_intersecting) > 0){
        divs <- add_another_5(working_5s_list, remaining_combs[non_intersecting], divs)
      }

    }
    #if there are no valid combinations, or once the valid ones have been
    #traversed, continue on in the for loop, removing the group that we appended
    working_5s_list <- working_5s_list[-length(working_5s_list)]
  }
  return(divs)
}

#Run through this recursive search
working_5_5_divisions <- find_working_5_5_divisions(combs, working_5_5_divisions)
working_5_5_divisions <- matrix(as.numeric(working_5_5_divisions), nrow = 5)

ncol(working_5_5_divisions)
```

```
## [1] 12781
```

We find **12,781 valid draws** of 5 groups of 5 numbers in which all of the friends' numbers have a product of 19,958,400.

We can look at any of these draws to see what primes they are made up of, and to verify they are valid:

```
random_draw <- working_5_5_divisions[,sample(1:12781, 1)]
draw_indices <- combinations[,random_draw]
draw_numbers <- apply(draw_indices, 2, function(x) non_primes[x,]$number)
draw_numbers <- rbind(draw_numbers, apply(draw_numbers, 2, prod))
colnames(draw_numbers) <- c("Friend1","Friend2","Friend3","Friend4","Friend5")
rownames(draw_numbers) <- c("Number1","Number2","Number3","Number4","Number5","Product")
draw_numbers
```

```
##          Friend1  Friend2  Friend3  Friend4  Friend5
## Number1        6       10       12       14       15
## Number2       33       24       20       18       21
## Number3       36       28       30       40       22
## Number4       50       54       42       44       48
## Number5       56       55       66       45       60
## Product 19958400 19958400 19958400 19958400 19958400
```

Success. This is a valid draw, since the 5 numbers each friend pulls all multiply to 19,958,400.

It is important to note that a slightly different reading of the question could give a much larger answer than 12,781. I am choosing to view the same division of the 25 numbers into 5 groups of 5 as one "way" no matter which friend gets which grouping of 5. If instead we took the random valid draw above, labeled column 1 "Friend2" and column 2 "Friend1", and declared that to be a new valid draw, we would have far more valid draws. This marks the difference between a combination and a permutation (without getting into the ordering of numbers *within* each friend's draw), and the correct answer would be

$$12{,}781 * 5! = 1{,}533{,}720.$$