Efficiently enumerate all subsets with difference constraints in R

Asked today Modified today Viewed 35 times Part of R Language Collective

4

I have a vector `v` of consecutive integers of length `l`, e.g., `1, 2, 3, 4, 5, 6, 7`. I want to find all subsets of size `k` such that the difference between any two numbers in the subset can be no less than `m`, e.g., `2`. Using the example above with `l = 7`, `k = 3`, and `m = 2`, the subsets are

```
1, 3, 5
1, 3, 6
1, 3, 7
1, 4, 6
1, 4, 7
1, 5, 7
2, 4, 6
2, 4, 7
2, 5, 7
3, 5, 7
```

One approach is to enumerate all possible subsets of size `k` and discard any that fail to meet the `m` constraint, but this procedure explodes even when the number of solutions is small.

My current approach is a brute-force algorithm in which I start from the subset with the lowest possible integer and increase the last entry by 1 until the upper limit is reached, increment the previous entry and reset the last entry to the lowest it can be given the increase in the previous entry. That is, I start with `1, 3, 5`, then increase the last digit by one to get `1, 3, 6` and `1, 3, 7`, and then since the upper limit is reached I increment the middle by 1 (to `4`) and set the last digit to the smallest it can be given that digit (to `6`) to get `1, 4, 6`, and carry on as such. This ends up being quite slow in R for large `l`, and I'm wondering if there is a clever vectorized solution that can instantly generate all the combinations, possible by capitalizing on the sequential nature of the entries. Implementing this algorithm in `Rcpp` speeds this up a bit, but I'm still hoping for a more elegant solution if one is available.

r algorithm combinations combinatorics enumeration Edit tags

Share Edit Follow Close Flag

asked 8 hours ago
Noah
3,415 1 11 27

Hey, I find this question fascinating, as well as Allan's answer below. Forgive my ignorance, but what is the use case for this? (there doesn't have to be one) I just wonder if this might be something I might come across for practical reasons in the future. – ScottyJ 4 hours ago

@ScottyJ My application is enumerating all possible segments for a segmented regression in which each segment has at least `m` data points in it. – Noah 2 hours ago

Sorted by:

2 Answers

Reset to default

Date modified (newest first)

0

Here is another attempt with `utils:::combn()` (to generate subsets) and `dist()` with `method='manhattan'` to enforce the constraints:

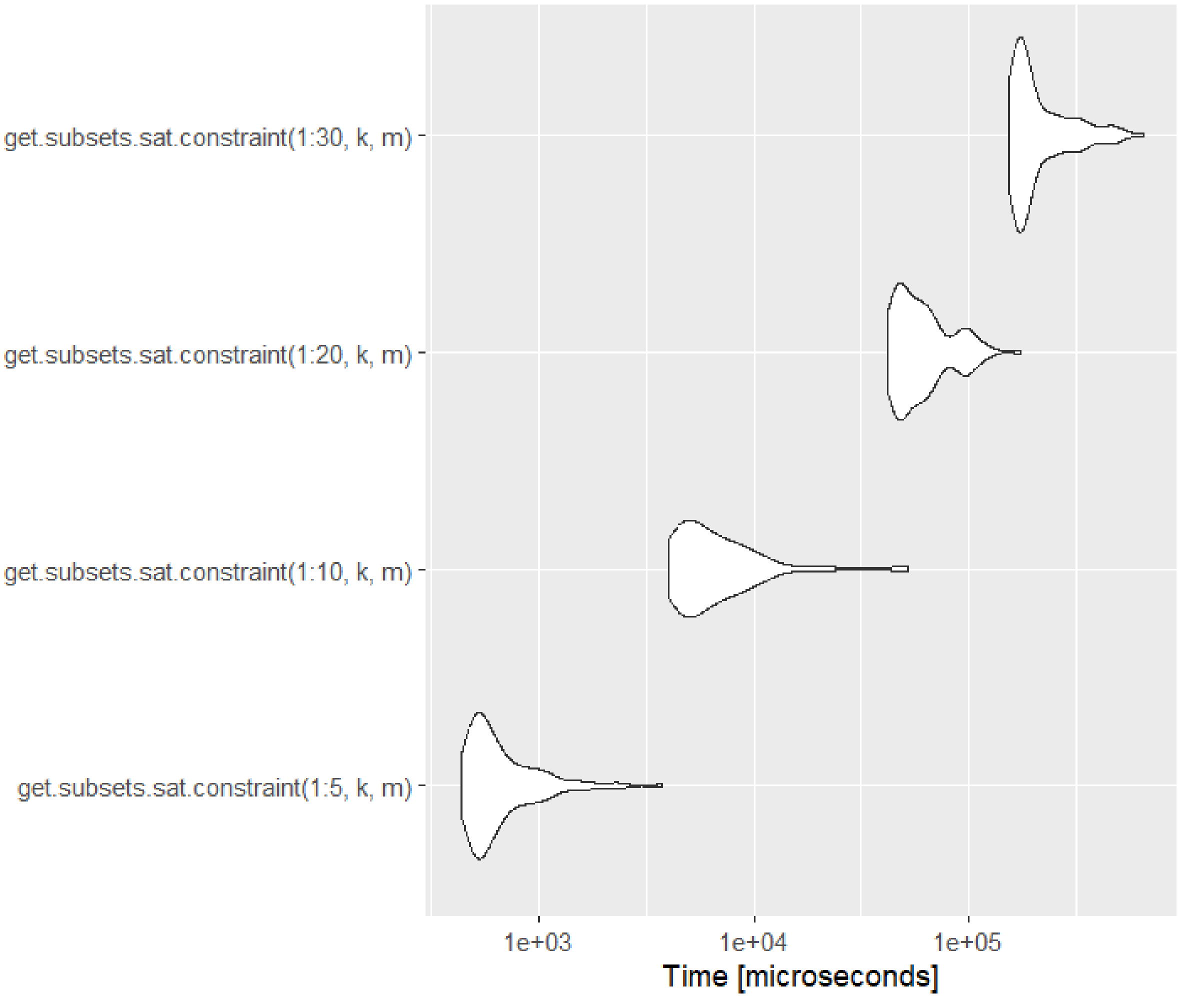
```
library(utils)

get.subsets.sat.constraint <- function(x, k, m) {
  combs <- combn(x, k)
  indices <- apply(combs, 2, function(y) all(dist(y, method='manhattan') >= m))
  t(combs[,indices])
}

n <- 7
k <- 3
m <- 2
x <- 1:n
get.subsets.sat.constraint(x, k, m)
#      [,1] [,2] [,3]
# [1,]    1    3    5
# [2,]    1    3    6
# [3,]    1    3    7
# [4,]    1    4    6
# [5,]    1    4    7
# [6,]    1    5    7
# [7,]    2    4    6
# [8,]    2    4    7
# [9,]    2    5    7
#[10,]    3    5    7
```

Now, let's see how the time complexity scales with `n`:

```
library(microbenchmark)
library(ggplot2)
tm <- microbenchmark(get.subsets.sat.constraint(1:5, k, m),
               get.subsets.sat.constraint(1:10, k, m),
               get.subsets.sat.constraint(1:20, k, m),
               get.subsets.sat.constraint(1:30, k, m),
               times=100L)
autoplot(tm)
```



Share Edit Delete Flag

answered 1 min ago

Sandipan Dey
21.4k 2 49 63

2

A reasonable approach might be to build the vectors recursively with the given constraints:

```
f <- function(v, k, m, existing = numeric()) {
  new_vals <- min(v):(max(v) - ((k - 1) * m))
  updated <- lapply(new_vals, function(x) c(existing, x))
  if(k == 1) return(updated)
  purrr::list_flatten(lapply(updated, function(x) {
    if((max(x) + m) == max(v)) return(c(x, max(v)))
    if(max(x) + m > max(v)) return(x)
    f(v[v >= (max(x) + m)], k - 1, m, x)
  }))
}
```

This gives us the same list for the given start parameters:

```
f(v = 1:7, k = 3, m = 2)
#> [[1]]
#> [1] 1 3 5
#>
#> [[2]]
#> [1] 1 3 6
#>
```

```
#> [[3]]
#> [1] 1 3 7
#>
#> [[4]]
#> [1] 1 4 6
#>
#> [[5]]
#> [1] 1 4 7
#>
#> [[6]]
#> [1] 1 5 7
#>
#> [[7]]
#> [1] 2 4 6
#>
#> [[8]]
#> [1] 2 4 7
#>
#> [[9]]
#> [1] 2 5 7
#>
#> [[10]]
#> [1] 3 5 7
```

And correctly finds the only set with k = 4 and m = 3 in the vector 1:10

```
f(v = 1:10, k = 4, m = 3)
#> [[1]]
#> [1]  1  4  7 10
```

Or the large set of k = 3, m = 2 in 1:10

```
f(v = 1:10, k = 3, m = 2)
#> [[1]]
#> [1] 1 3 5
#>
#> [[2]]
#> [1] 1 3 6
#>
#> [[3]]
#> [1] 1 3 7
#>
#> [[4]]
#> [1] 1 3 8
#>
#> [[5]]
#> [1] 1 3 9
#>
#> [[6]]
#> [1]  1  3 10
#>
#> [[7]]
#> [1] 1 4 6
#>
#> [[8]]
#> [1] 1 4 7
#>
#> [[9]]
#> [1] 1 4 8
#>
#> [[10]]
#> [1] 1 4 9
```

```
#>
#> [[11]]
#> [1]  1  4 10
#>
#> [[12]]
#> [1] 1 5 7
#>
#> [[13]]
#> [1] 1 5 8
#>
#> [[14]]
#> [1] 1 5 9
#>
#> [[15]]
#> [1]  1  5 10
#>
#> [[16]]
#> [1] 1 6 8
#>
#> [[17]]
#> [1] 1 6 9
#>
#> [[18]]
#> [1]  1  6 10
#>
#> [[19]]
#> [1] 1 7 9
#>
#> [[20]]
#> [1]  1  7 10
#>
#> [[21]]
#> [1]  1  8 10
#>
#> [[22]]
#> [1] 2 4 6
#>
#> [[23]]
#> [1] 2 4 7
#>
#> [[24]]
#> [1] 2 4 8
#>
#> [[25]]
#> [1] 2 4 9
#>
#> [[26]]
#> [1]  2  4 10
#>
#> [[27]]
#> [1] 2 5 7
#>
#> [[28]]
#> [1] 2 5 8
#>
#> [[29]]
#> [1] 2 5 9
#>
#> [[30]]
#> [1]  2  5 10
#>
#> [[31]]
#> [1] 2 6 8
#>
#> [[32]]
#> [1] 2 6 9
```

```
#>
#> [[33]]
#> [1]  2  6 10
#>
#> [[34]]
#> [1] 2 7 9
#>
#> [[35]]
#> [1]  2  7 10
#>
#> [[36]]
#> [1]  2  8 10
#>
#> [[37]]
#> [1] 3 5 7
#>
#> [[38]]
#> [1] 3 5 8
#>
#> [[39]]
#> [1] 3 5 9
#>
#> [[40]]
#> [1]  3  5 10
#>
#> [[41]]
#> [1] 3 6 8
#>
#> [[42]]
#> [1] 3 6 9
#>
#> [[43]]
#> [1]  3  6 10
#>
#> [[44]]
#> [1] 3 7 9
#>
#> [[45]]
#> [1]  3  7 10
#>
#> [[46]]
#> [1]  3  8 10
#>
#> [[47]]
#> [1] 4 6 8
#>
#> [[48]]
#> [1] 4 6 9
#>
#> [[49]]
#> [1]  4  6 10
#>
#> [[50]]
#> [1] 4 7 9
#>
#> [[51]]
#> [1]  4  7 10
#>
#> [[52]]
#> [1]  4  8 10
#>
#> [[53]]
#> [1] 5 7 9
#>
#> [[54]]
#> [1]  5  7 10
```

```
#>
#> [[55]]
#> [1]  5  8 10
#>
#> [[56]]
#> [1]  6  8 10
```

You will still run up against combinatorial explosion if you try to select large values of k in a large vector, but this solution is surprisingly fast - it can generate all 152,000 possible length-3 vectors where m = 2 between 1 and 100 in less than half a second. With k = 4, there are almost 3.5 million possible solutions, so it takes a few seconds.

I wouldn't attempt k = 5 for a vector that large...

Created on 2023-08-24 with [reprex v2.0.2]

Share Edit Follow Flag

edited 6 hours ago

answered 6 hours ago

Allan Cameron

146k 7 48 86

Recognized by R Language Collective