DCM Analysis LumA vs. LumB

Kevin O'Connor

6/18/2018

In this document, I will analyze the results of applying DCM to the cancer subtypes, LumA and LumB. For this purpose, the DCM code has been modified to save important information from each iteration. In the case of LumA vs. LumB, we saw that 5 DC sets were observed after running the algorithm. So, the code now stores iteration-level data from each of these.

```
list.files(file.path(getwd(), "Debug_Output"))
```

```
## [1] "Console_Output.txt"   "DCM_1.RData"          "DCM_2.RData"
## [4] "DCM_3.RData"          "DCM_4.RData"          "DCM_5.RData"
## [7] "Initialization.RData"
```

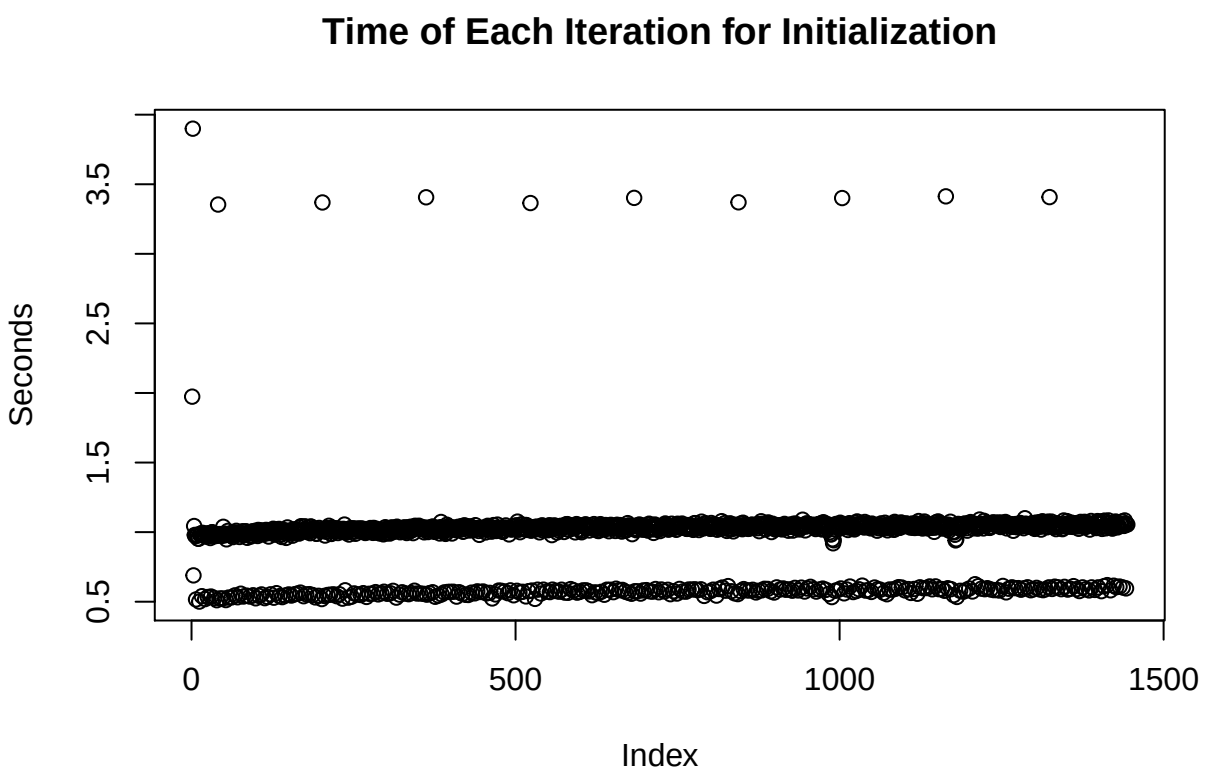
We see that the initialization data and console output is also stored. For now, we will look at the initialization procedure and the search which found the first set.

```
load(file.path(getwd(), "Debug_Output/Initialization.RData"))
attributes(initialization)
```

```
## $names
## [1] "seed"       "found"      "iterations" "time"       "sets"
## [6] "it_times"
```

Looking at the time that the initialization took,

```
n_its <- initialization$iterations
it_time <- initialization$time
plot(unlist(initialization$it_times), main="Time of Each Iteration for Initialization", ylab="Seconds")
```



In total the initialization procedure took 1444 iterations and 22.9 minutes.

Now looking at the first DC search procedure,

```
load(file.path(getwd(), "Debug_Output/DCM_1.RData"))
attributes(DCM)
```

```
## $names
##  [1] "found"     "mc1"       "mc2"       "its"       "time"
##  [6] "pvals"     "startdels" "it_sets"   "it_p_vals" "it_times"
```

As observed before, we have

```
DCM$its
```

```
## [1] 11
```

i.e., the algorithm reached the max number of iterations and never converged. The times that each iteration took are

```
round(unlist(DCM$it_times)/60, 2)
```

```
##  [1] 1.60 0.39 2.42 0.86 2.10 1.21 1.80 1.66 1.71 1.68 1.92
```

where the units are in minutes. This is surprisingly fast considering that the whole procedure took several hours to complete. It might be helpful to look at these times for each of the five searches.

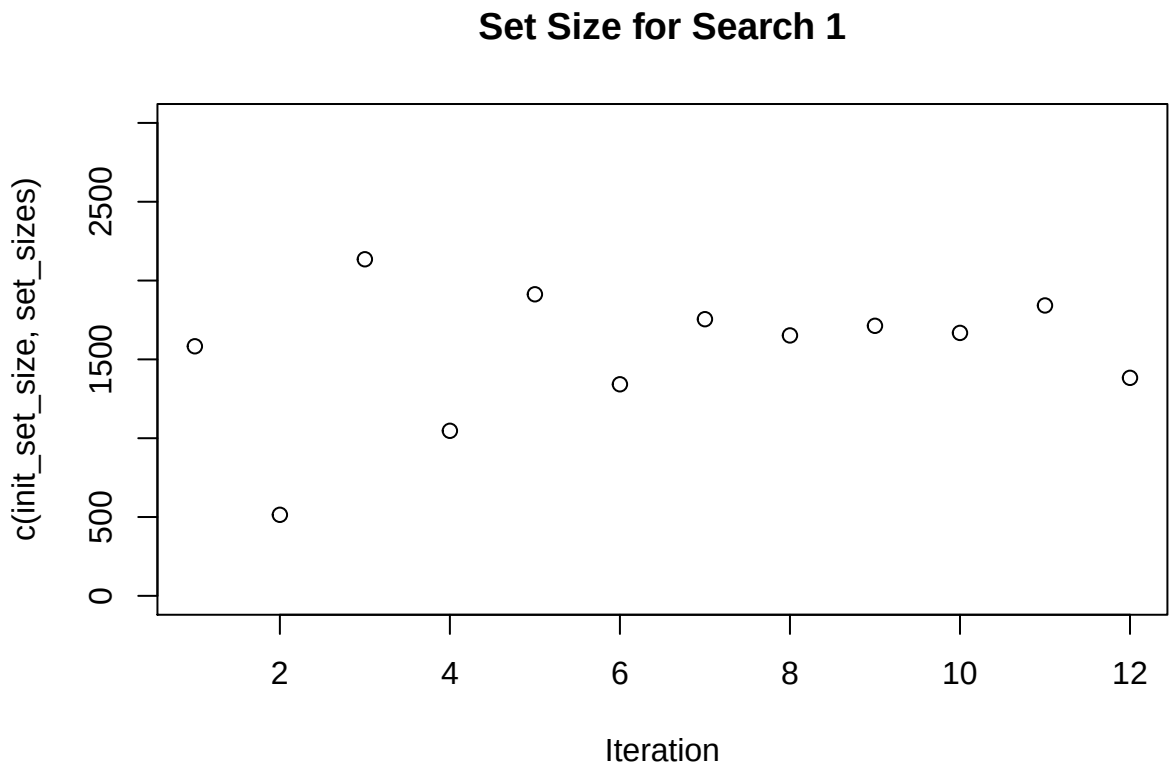
```
no_output <- lapply(1:5, function(i){
  load(file.path(getwd(), paste0("Debug_Output/DCM_", i, ".RData")))
  times <- round(unlist(DCM$it_times)/60, 2)
  print(paste("search:", i))
  print("iteration times:")
  print(times)
  print(paste("total time:", sum(times)))
  print('')
})
```

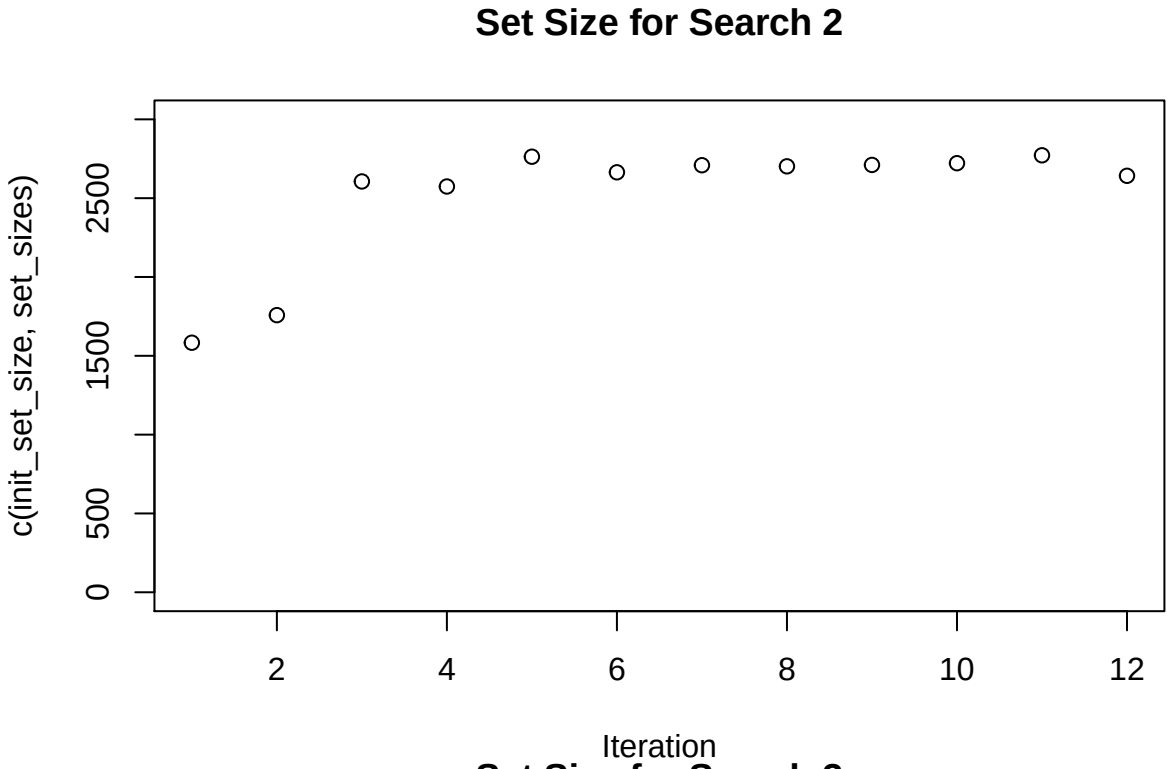
```
## [1] "search: 1"
## [1] "iteration times:"
##  [1] 1.60 0.39 2.42 0.86 2.10 1.21 1.80 1.66 1.71 1.68 1.92
## [1] "total time: 17.35"
## [1] ""
## [1] "search: 2"
## [1] "iteration times:"
##  [1] 1.61 1.90 3.58 3.37 3.73 3.49 3.52 3.58 3.61 3.63 3.63
## [1] "total time: 35.65"
## [1] ""
## [1] "search: 3"
## [1] "iteration times:"
##  [1] 1.58 0.22 3.60 0.62 2.22 1.19 1.65 1.58 1.62 1.58 1.57
## [1] "total time: 17.43"
## [1] ""
## [1] "search: 4"
## [1] "iteration times:"
##  [1] 1.54 0.60 1.08 1.14 1.28 1.29 1.26 2.25 1.38 1.35 1.31
## [1] "total time: 14.48"
## [1] ""
## [1] "search: 5"
## [1] "iteration times:"
```

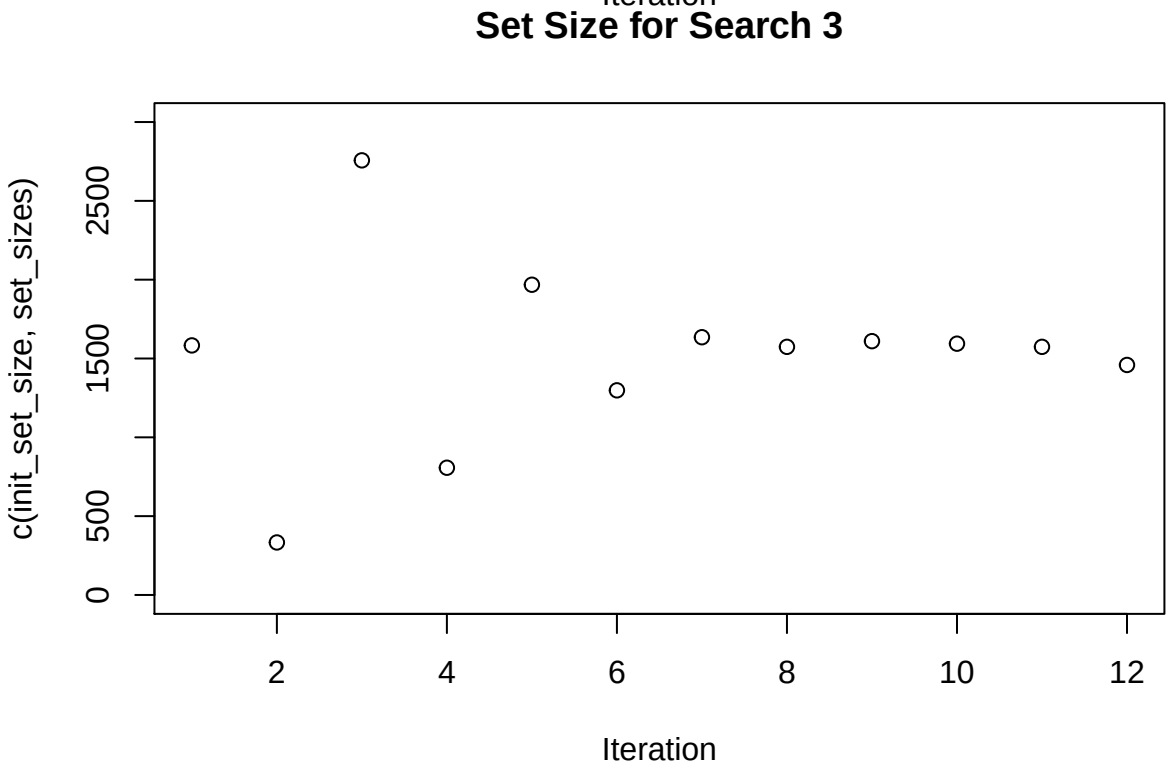
```
##  [1] 1.55 0.05 1.73 0.11 2.00 0.06 2.25 0.06 2.51 0.11 2.28
## [1] "total time: 12.71"
## [1] ""
```

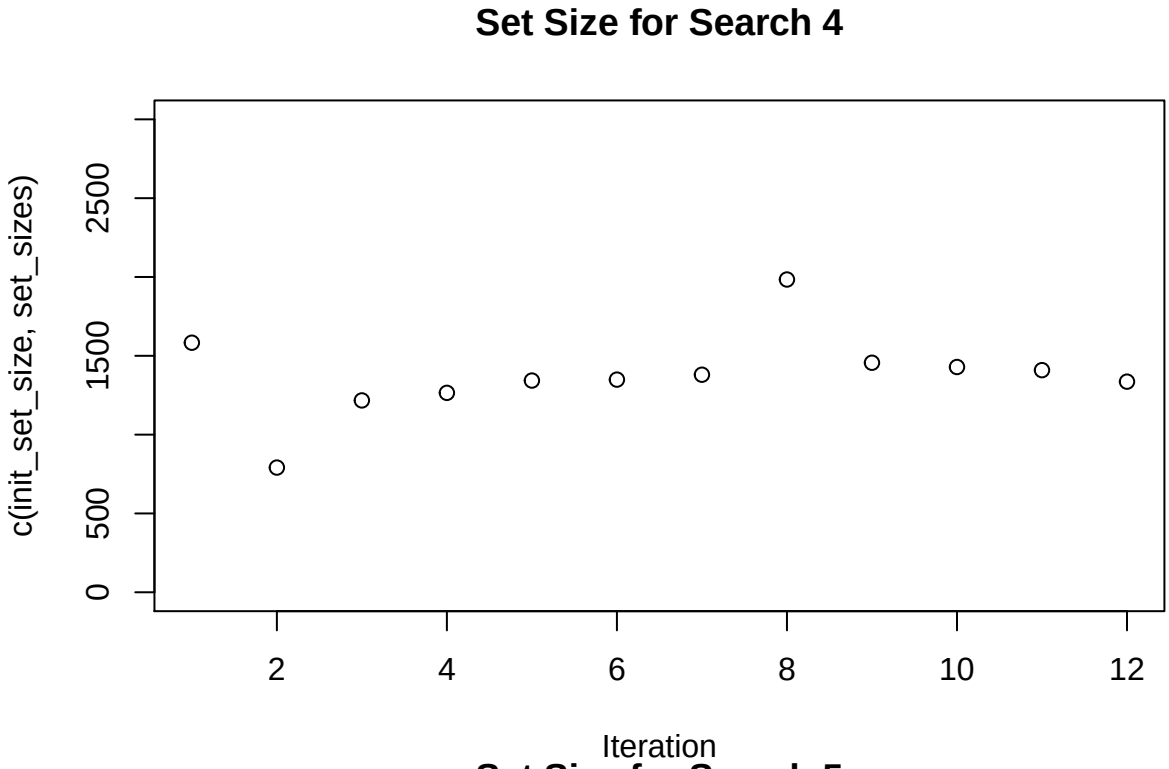
Now let's look at the size of the DC set through each iteration.

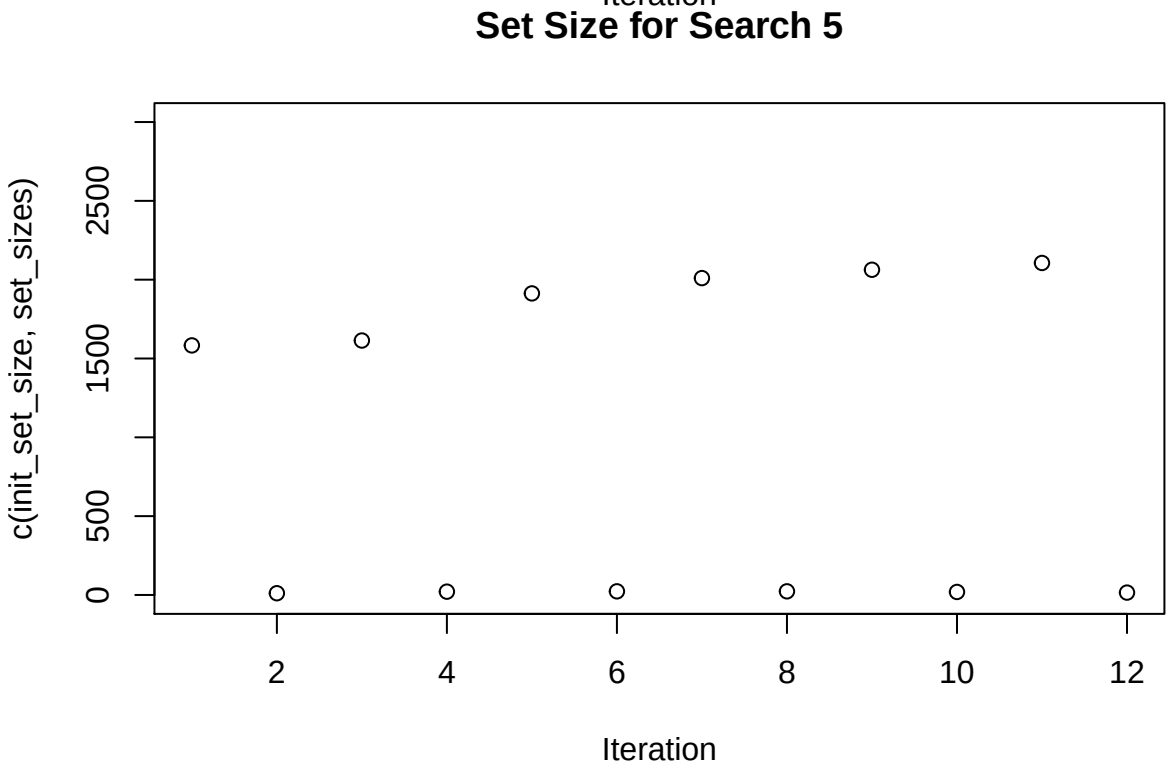
```
load(file.path(getwd(), paste0("Debug_Output/Initialization.RData")))
init_set_size <- length(initialization$"found")
no_output <- lapply(1:5, function(i){
  load(file.path(getwd(), paste0("Debug_Output/DCM_", i, ".RData")))
  set_sizes <- lapply(DCM$it_sets, function(i){length(unlist(i))}) %>% unlist()
  plot(c(init_set_size, set_sizes), main=paste0("Set Size for Search ", i), xlab="Iteration", ylim=c(0,
})
```




Set Size for Search 2
c(init_set_size, set_sizes)
0
500
1500
2500
2
4
6
8
10
12
Iteration


Set Size for Search 3
c(init_set_size, set_sizes)
0
500
1500
2500
2
4
6
8
10
12
Iteration





Relationship Between Consecutive Sets

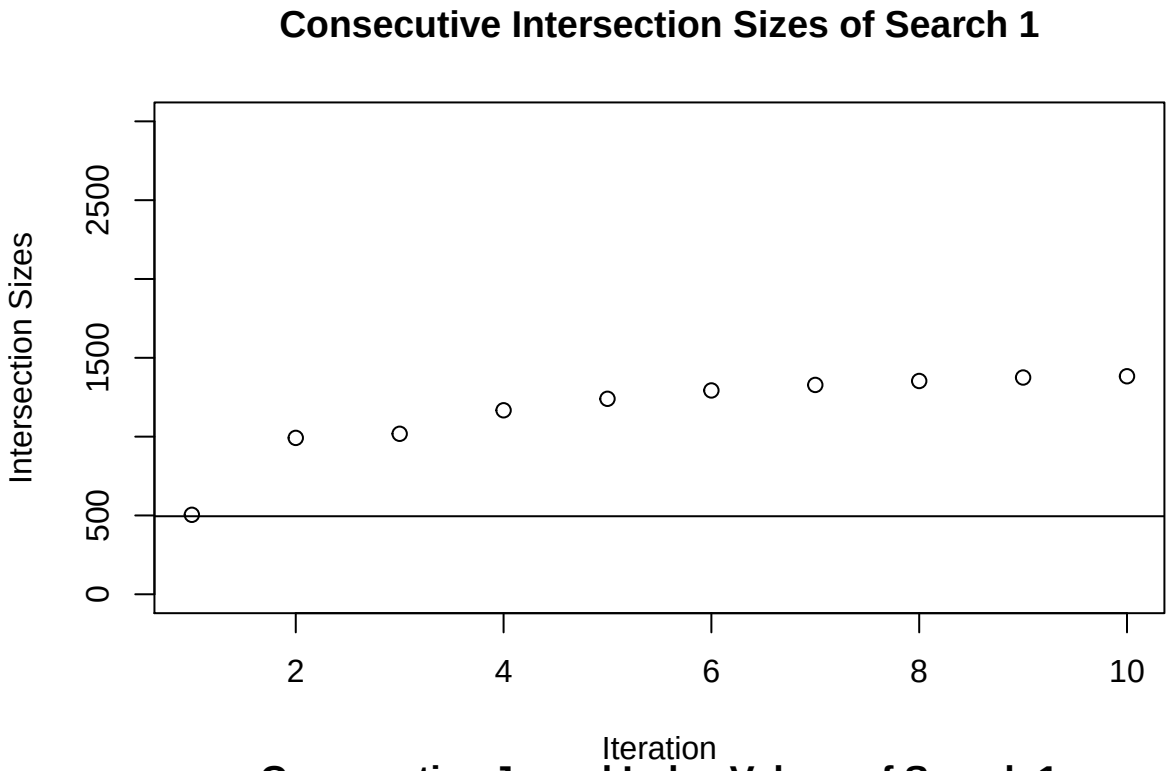
Having looked at the size of the sets changing iteration to iteration, we are also interested in determining whether there is some core set of variables which are present in each iteration in addition to some random

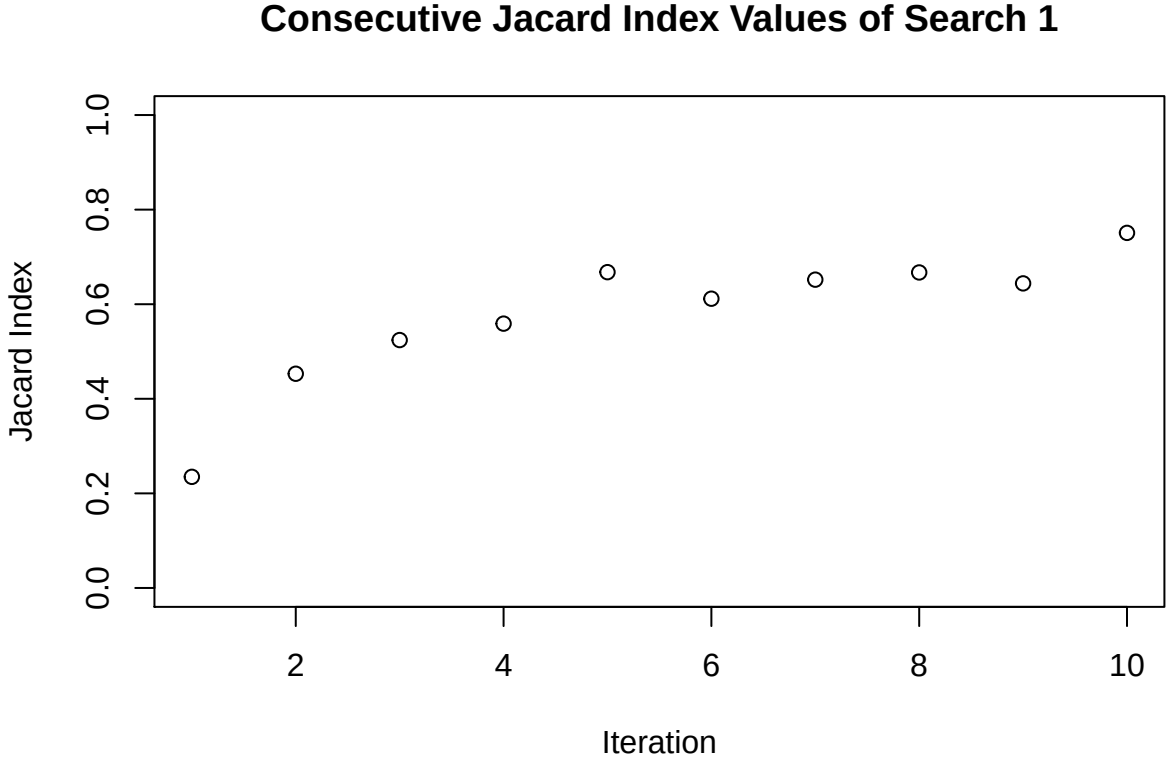
noise. First we look at the sizes of the itersections and the Jacard index between each consecutive set. Here the Jacard index is given by

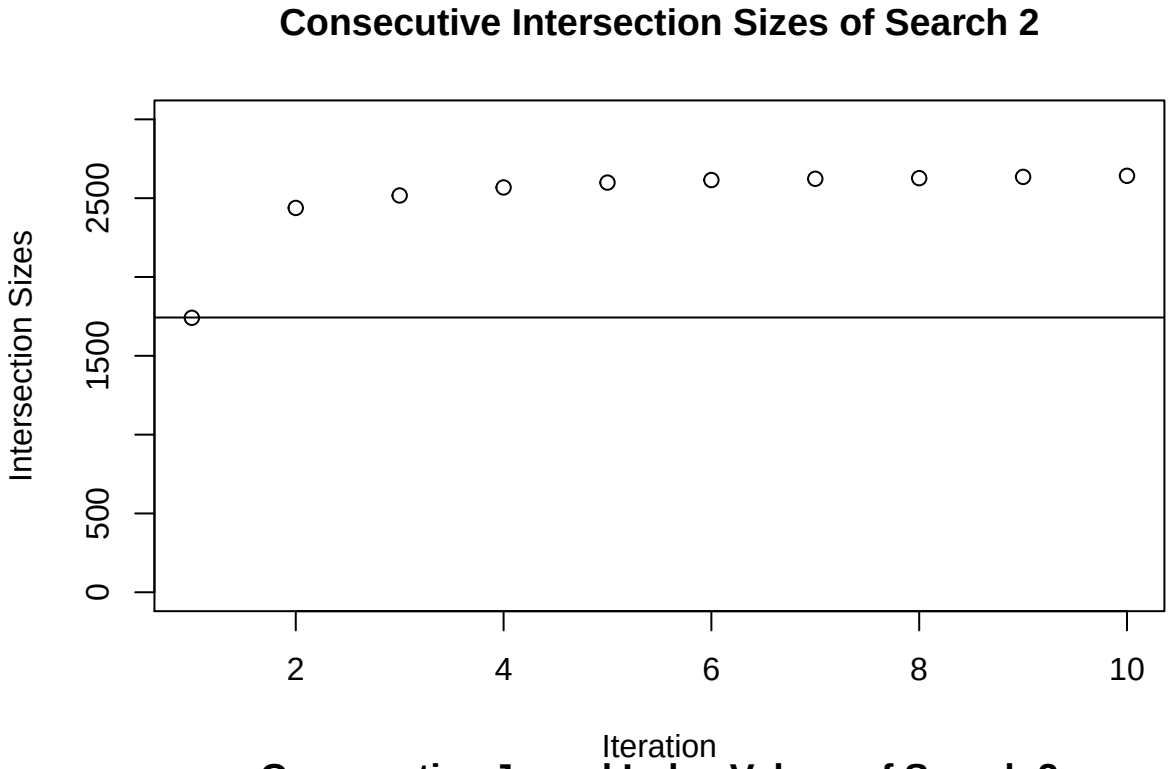
$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

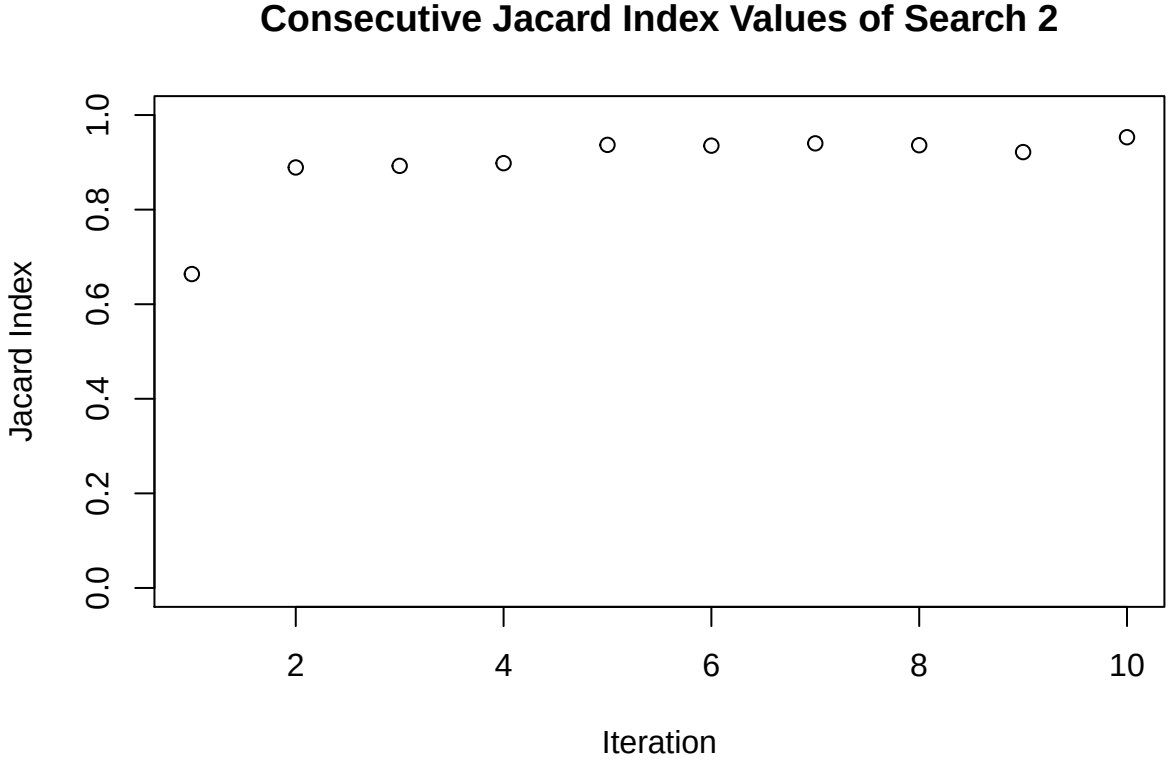
Additionally, in each plot of intersection sizes the size of the total intersection over all iterations is included as a solid horizontal line. In other words, this line represents the number of variables that are included in the set for every iteration.

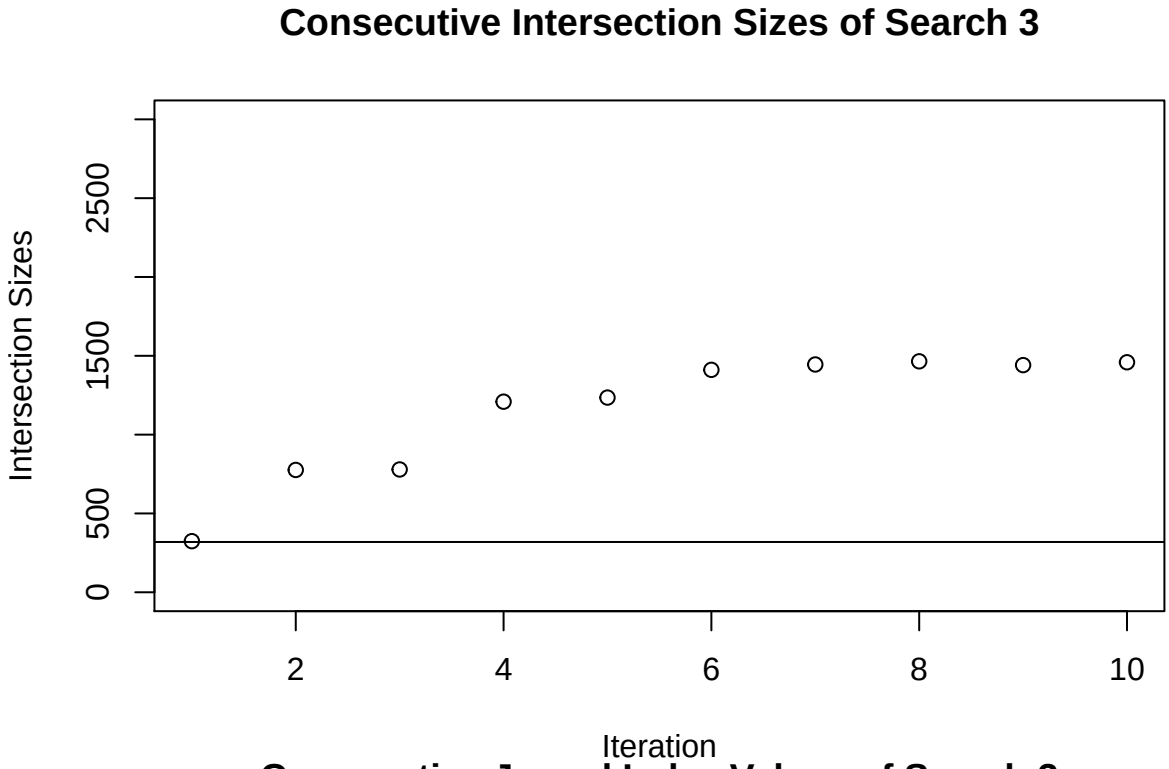
```
jacard <- function(a, b){
  length(intersect(a, b))/length(union(a, b))
}

load(file.path(getwd(), paste0("Debug_Output/Initialization.RData")))
init_set <- length(initialization$"found")
no_output <- lapply(1:5, function(i){
  load(file.path(getwd(), paste0("Debug_Output/DCM_", i, ".RData")))

  # Iterate through consecutive sets and record intersections and jacards.
  tot_int <- DCM$it_sets[[1]]
  vals <- lapply(1:(length(DCM$it_sets)-1), function(j){
    set_a <- DCM$it_sets[[j]]
    set_b <- DCM$it_sets[[j+1]]
    tot_int <- length(intersect(tot_int, set_b))
    if(j < (length(DCM$it_sets)-1)){
      return(list("intersect" = length(intersect(set_a, set_b)),
                  "jacard"    = jacard(set_a, set_b)))
    } else {
      return(list("intersect" = length(intersect(set_a, set_b)),
                  "jacard"    = jacard(set_a, set_b),
                  "tot_int"   = tot_int))
    }
  })
  inter_vals <- lapply(vals, function(v){v$"intersect"}) %>% unlist()
  jac_vals   <- lapply(vals, function(v){v$"jacard"}) %>% unlist()
  tot_int <- vals[[length(vals)]]$"tot_int"

  # Make plots.
  plot(inter_vals,
       main=paste0("Consecutive Intersection Sizes of Search ", i),
       xlab="Iteration",
       ylab="Intersection Sizes",
       ylim=c(0,3000))
  abline(h=tot_int)
  plot(jac_vals,
       main=paste0("Consecutive Jacard Index Values of Search ", i),
       xlab="Iteration",
       ylab="Jacard Index",
       ylim=c(0,1))
})
```

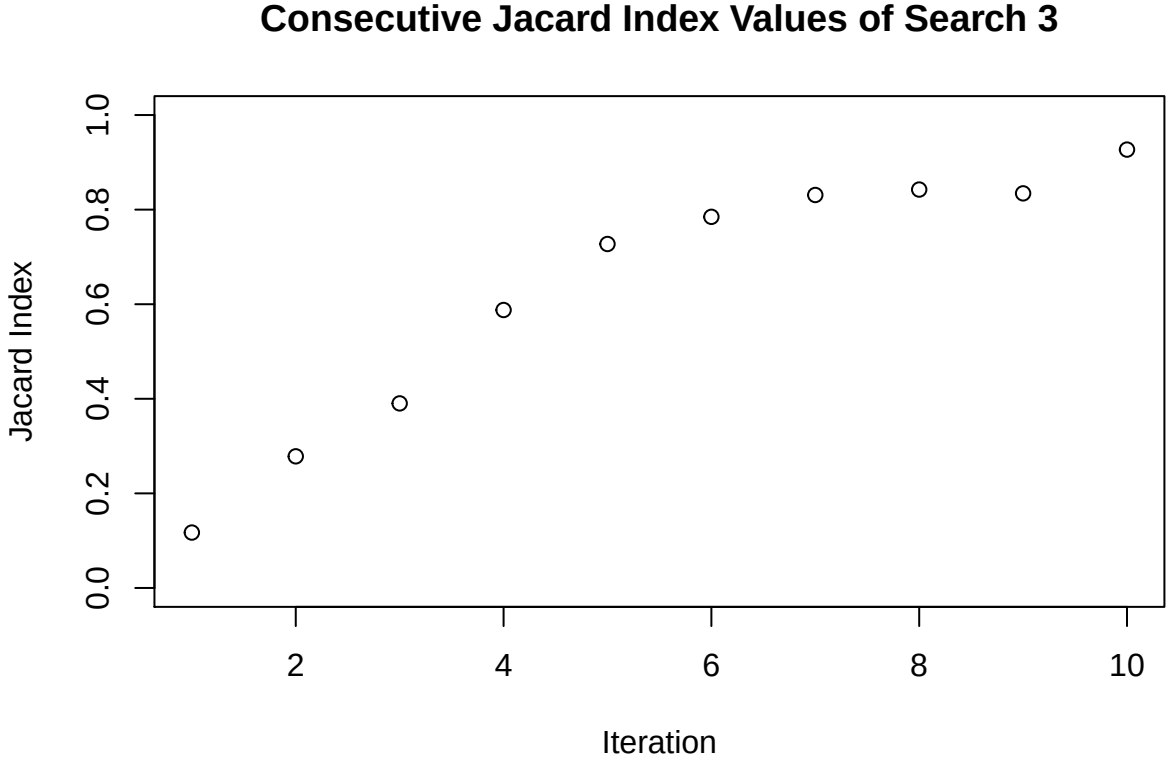

Consecutive Intersection Sizes of Search 1
Intersection Sizes
0
500
1500
2500
2
4
6
8
10
Iteration

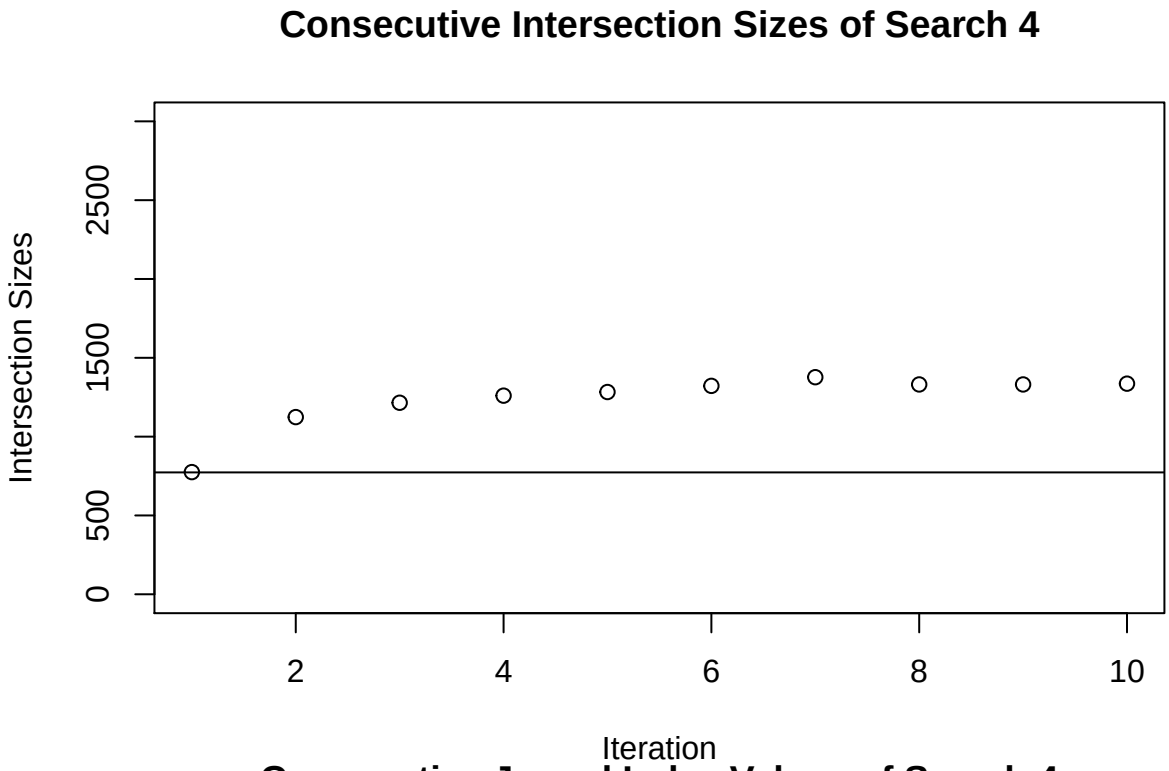

Consecutive Jacard Index Values of Search 1
Jacard Index
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Iteration

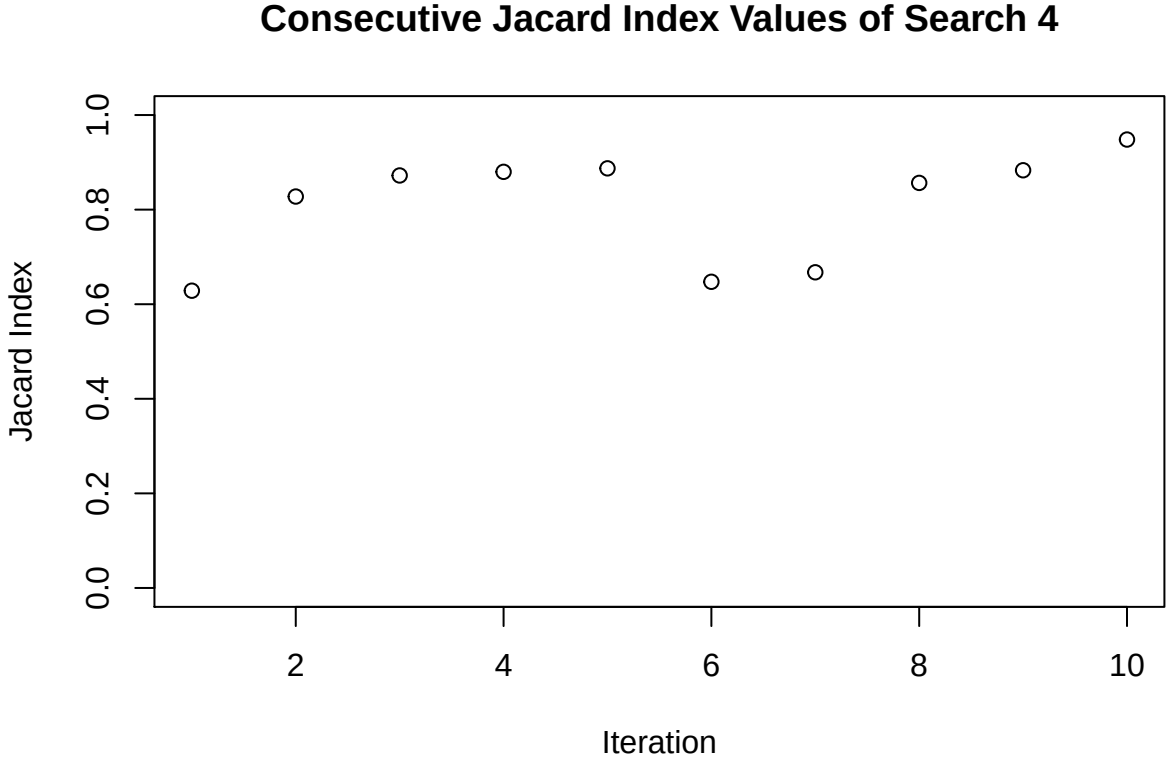

Consecutive Intersection Sizes of Search 2
Intersection Sizes
0
500
1500
2500
2
4
6
8
10
Iteration

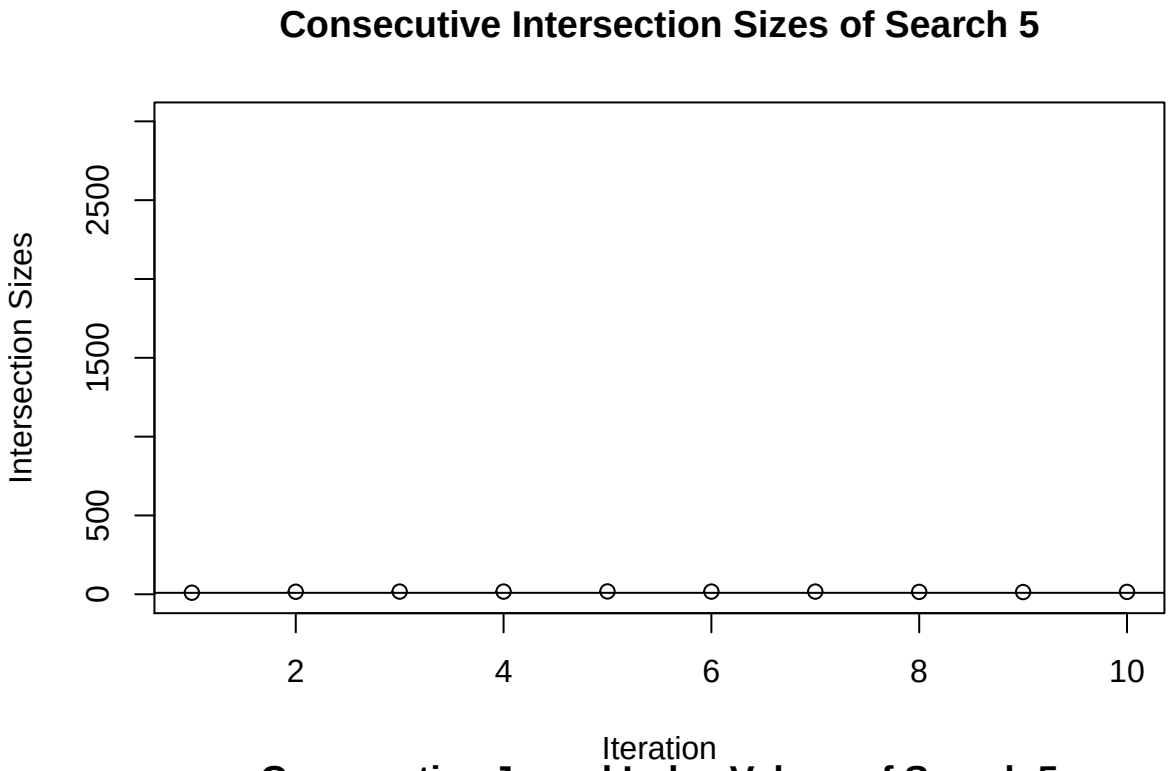

Consecutive Jacard Index Values of Search 2
Jacard Index
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Iteration

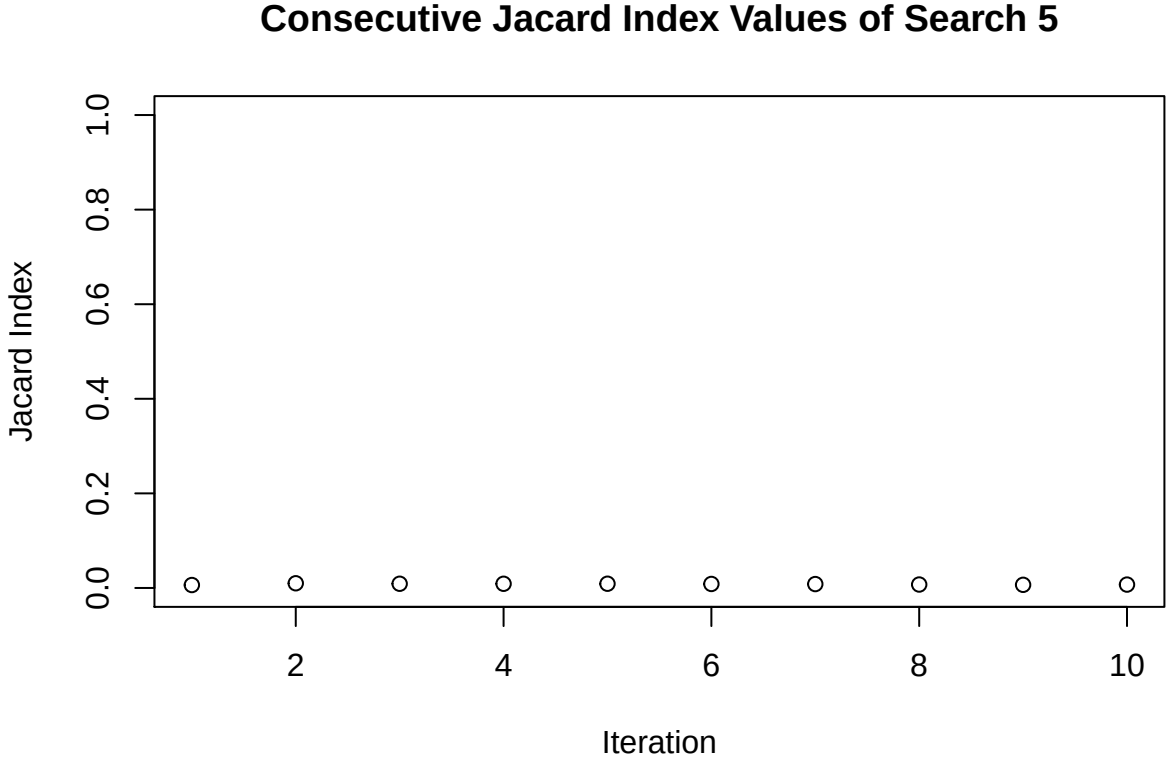

Consecutive Intersection Sizes of Search 3
Intersection Sizes
0
500
1500
2500
2
4
6
8
10
Iteration


Consecutive Jacard Index Values of Search 3
Jacard Index
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Iteration


Consecutive Intersection Sizes of Search 4
Intersection Sizes
0
500
1500
2500
2
4
6
8
10
Iteration


Consecutive Jacard Index Values of Search 4
Jacard Index
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Iteration


Consecutive Intersection Sizes of Search 5
Intersection Sizes
0
500
1500
2500
2
4
6
8
10
Iteration


Consecutive Jacard Index Values of Search 5
Jacard Index
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Iteration