Programación Estadística: Búsqueda ciega

Adrián Sosa
Universidad Veracruzana

Búsqueda ciega

La búsqueda ciega completa asume el agotamiento de todas las alternativas, donde cualquier búsqueda previa no afecta la forma en que se prueban las siguientes soluciones. Dado que Se prueba el espacio de búsqueda completo, siempre se encuentra la solución óptima. La búsqueda ciega solo es aplicable a espacios de búsqueda discretos y es fácil de codificar de dos formas.

Primero, configurando el espacio de búsqueda completo en una matriz y luego probando secuencialmente cada fila (solución) de esta matriz.

En segundo lugar, de forma recursiva, configurando el espacio de búsqueda como un árbol, donde cada rama denota un valor posible para una variable dada y todas las soluciones aparecen en las hojas (al mismo nivel).

Dos ejemplos bastante conocidos delos métodos ciegos basados en estructuras de árbol son los algoritmos de profundidad y de amplitud.

El primero comienza en la raíz del árbol y atraviesa cada rama tanto como esposible, antes de dar marcha atrás. El segundo también comienza en la raíz pero busca en un nivel base, buscando primero todos los nodos sucesivos de la raíz y luego el siguiente sucesivo a los nodos, etc.

La principal desventaja de la búsqueda ciega pura es que no es factible cuando el espacio de búsqueda es continuo o demasiado grande, una situación que a menudo ocurre con las tareas del mundo real.

Esta sección presenta dos funciones de búsqueda ciega: fsearch y dfsearch. La primera es una función más simple que requiere que el espacio de búsqueda se defina explícitamente en una matriz en el formato soluciones D (búsqueda de argumento), mientras que la última realiza una implementación recursiva de la búsqueda en profundidad y requiere la definición de los valores de dominio para cada variable a optimizar (dominio de argumento). Ambas funciones reciben como argumentos la función de evaluación (FUN), el tipo de optimización (tipo, un carácter con "min" o "max") y argumentos adicionales, (denotados por ... y que podría ser utilizado por la función de evaluación FUN).

Donde dfsearch es una función recursiva que prueba si el nodo del árbol es una salida, calcula la función de evaluación para la solución respectiva, de lo contrario atraviesa las sub ramas del nodo. Esta función requiere algunas variables de estado de memoria (l, b, x y msol) que se cambian cada vez que se ejecuta una nueva llamada recursiva. El dominio de valores se almacena en una lista de vectores de longitud D, ya que los elementos de este vector pueden tener diferentes longitudes, de acuerdo con sus valores de dominio.

Codificación

La codificación de dichas funciones podemos observala acontinuación:

```
fsearch
```

```
## function (sol, Fx, type = "min", ...)
## {
##     x <- apply(sol, 1, Fx, ...)
##     ib <- switch(type, min = which.min(x), max = which.max(x))
##     return(list(index = ib, sol = sol[ib], eval = x[ib]))
## }
```

```
dfsearch
```

```
## function (domino, Fx, l = 1, b = 1, type = "min", D = length(dominio),
##     x = rep(NA, D), msol = switch(type, min = list(sol = NULL,
##         eval = Inf), max = list(sol = NULL, eval = -Inf)), ...)
## {
##     if ((l - 1) == D) {
##         f <- Fx(x, ...)
##         fb <- msol$eval
##         ib <- switch(type, min = which.min(c(fb, f)), max = which.max(c(fb,
##             f)))
##         if (ib == 1)
##             return(msol)
##         else return(list(index = ib, sol = x, eval = f, bsol = x[ib],
##             beval = f[ib]))
##     }
##     else {
##         for (j in 1:length(dominio[[l]])) {
##             x[l] <- dominio[[l]][j]
```

```
##             msol <- dfsearch(dominio, Fx, l + 1, j, type, D = D,
##                 x = x, msol = msol, ...)
##         }
##         return(msol)
##     }
## }
```

Se crea un espacio de búsqueda con valores entre 0 y 9 en una matriz de 5 *7 y se define una función:

```
# función a evaluar
fx1
```

```
## function (x)
## {
##     return(x^2 + 3 * x - 2)
## }
```

```
m <- 5  # número de variables
n <- 5  # tamaño del espacio


# definimos un espacio de busqueda
dominio <- matrix(data=sample(0:9, m*n,TRUE),n,m)
# mostramos el espacio de búsqueda
dominio
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    9    3    3    7    5
## [2,]    8    4    6    7    7
## [3,]    3    2    3    8    0
## [4,]    5    7    8    1    7
## [5,]    7    4    2    2    7
```

El siguiente código prueba el funcionamiento del método de búsqueda ciega en amplitud:

```
# búsqueda ciega en amplitud
min <- fsearch(dominio,Fx=fx1,type="min")
max <- fsearch(dominio,Fx=fx1,type="max")
```

```
## Solución mínima:  2
##  Evaluación:  -2

## Solución máxima:  9
##  Evaluación:  106
```

El siguiente código prueba el funcionamiento de la fúncion de búsqueda ciega en profundidad:

```
# búsqueda ciega en profundidad
min <- dfsearch(dominio,Fx=fx1,type="min")
max <- dfsearch(dominio,Fx=fx1,type="max")
```

```
## Solución mínima:  7
##  Evaluación:  68

## Solución máxima:  8
##  Evaluación:  86
```