Práctica 7: búsqueda local

Ricardo Rosas Macías

21 de mayo de 2019

1. Introducción

La búsqueda local usa el método heurístico, que le permite encontrar la mejor solución de todas las rutas de resultado posibles; a través de descubrir la posición exacta del agente que le deja obtenerla en un tiempo corto.

2. Objetivo

Se realizó cambios en el código proporcionado en la página web [4][1], de manera que el agente en el experimento mantiene un movimiento 3D pero con una visualización en 2D que permite observar la posición óptima en la secuencia del experimento.

2.1. Descripción

Lo que se debe hacer es [4]:

> "La finalidad del experimento es maximizar la función bidimensional del ejemplo $g(x, y)$, con restricciones $-3 \leq x, y \leq 3$, con la misma técnica del ejemplo unidimensional. La posición actual es un par x y y se ocupan dos movimientos aleatorios, $\triangle x$ y $\triangle y$, cuyas combinaciones posibles proveen ocho posiciones vecino, de los cuales aquella que logra el mayor valor para g es seleccionado.
> El primer reto es cambiar la regla del movimiento de una solución x (un vector de dimensión arbitraria) a la siguiente a la de recocido simulado: para optimizar una función $f(x)$, se genera para la solución actual x un sólo vecino $x' = x + \triangle x$ (algún desplazamiento local) Se calcula $\delta = f(x') - f(x)$ (para minimizar; maximizando la resta se hace al revés). Si $\delta > 0$, siempre se acepta al vecino x' como la solución actual ya que representa una mejora. Si $\delta < 0$, se acepta a x' con probabilidad $exp(-\frac{\delta}{T})$ y rechaza en otro caso. Aquí T es una temperatura que decrece en aquellos pasos donde se acepta una empeora; la reducción se logra multiplicando el valor actual de T con $\xi < 1$, como por ejemplo 0.995. Examina los efectos estadísticos del valor inicial de T y el valor de ξ en la calidad de la solución, es decir, qué tan bajo (para minimizar; alto para maximizar) el mejor valor termina siendo."

3. Resultados y conclusiones

En las primeras líneas del código se definió los parámetros de experimentación con las cuales se trabajaría con la función g.

```
g <- function(x, y) {
  return (((x + 0.5)^4 - 30 * x^2 - 20 * x + (y + 0.5)^4 - 30 * y^2 - 20 * y)/100)
}
low <- -3
high <- -low
step <- 0.25
replicas <- 15
```

Posteriormente se generó el código que de manera tuviera movimientos de izquierda o derecha, así como con un patrón de arriba o abajo, como se muestra en las líneas de código. Asimismo, se realizó una combinación de dichos movimientos de modo de recreación de un eje z para el movimiento del agente dentro de la zona creada de $-3 \leq x, y \leq 3$.

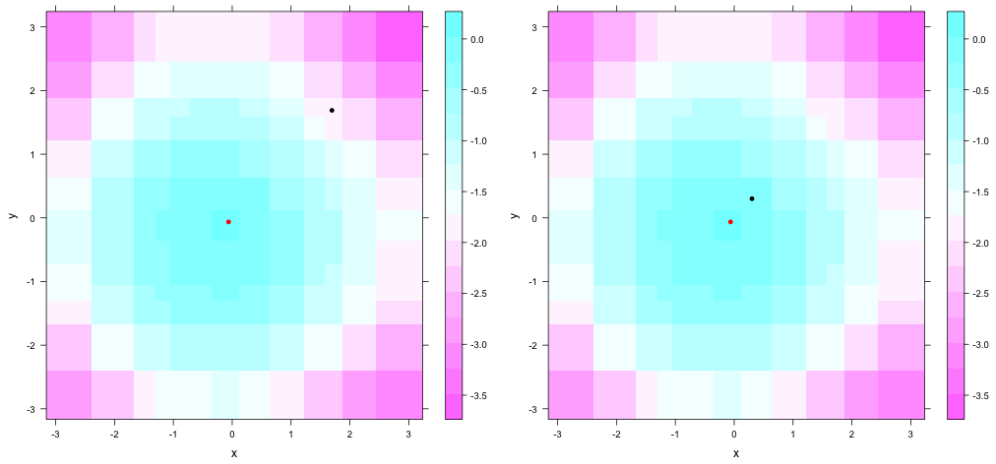
```
replica <- function(t){
  puntosxy<- c()
  curr <- c( x = runif(1, min = low, max = high), y = runif(1, min = low, max = high))
  best <- curr
  for (tiempo in 1:t) {
    delta <- runif(1, 0, step)
    omega <- runif(1, 0, step)
     left <- curr + c(-delta,0) # Eje izquierdo
    right <- curr + c(delta,0) # Eje Derecho
    up <- curr + c(0,-delta) # Eje arriba
    down <- curr + c(0,delta) # Eje abajo
    puntos <- c(left, right, up, down)

 for(k in 1:8){
      if(puntos[k] < (-3)){
        puntos[k] <- puntos[k]+3
      }
      if(puntos[k] > 3){
        puntos[k] <- puntos[k]-3
      }
    }
    vecx <- c()
    vecy <- c()
    for(p in 1:8){
      if(p %%2 == 0){
        vecy <- c(vecy,puntos[p])
      }else{
        vecx <- c(vecx,puntos[p])
      }
    }
    valg <- c()
    for(q in 1:4){
      valg <- c(valg, g(vecx[q], vecy[q]) )
    }
    dm <- which.max(valg)
```

```
    curr <- c(vecx[dm], vecy[dm])
    puntosxy <- c(puntosxy, vecx[dm],vecy[dm])
  }
  return(puntosxy)
}

resultado <- c()
for(q in 1:5){
  resultado <- c(resultado, replica(100))
}

vx <- c()
vy <- c()
for(p in 1:1000){
  if(p %%2 == 0){
    vy <- c(vy,resultado[p])
  }else{
    vx <- c(vx,resultado[p])
  }
}
```

En las líneas finales se puede ver que esta indicado una proximidad máxima local para los agentes, lo cual permitirá que seleccione el mejor de los vecinos y así este converja en un punto óptimo.

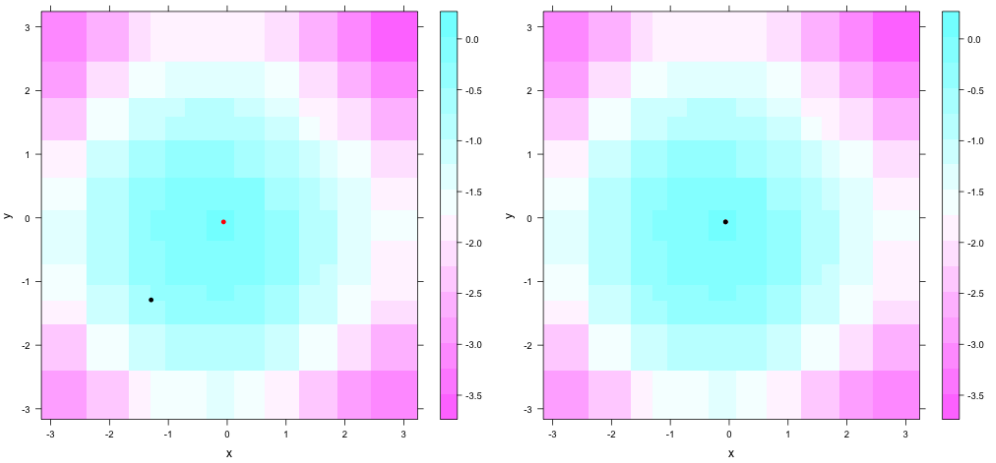
```
for (pot in 2:4) {
  tmax <- 10^pot
  resultados <- foreach(i = 1:replicas, .combine="rbind") %dopar% replica(tmax)
  valores <- outer(resultados[,1], resultados[,2], g)
  mejor <- which.max(valores)
  dimnames(valores)<-list(resultados[,1], resultados[,2])
  funcion <- melt(valores)
  names(funcion) <- c("x", "y", "z")
  zona <- levelplot(z ~ x * y, data = funcion)
```

Se obtuvieron representaciones gráficas con la ayuda de la paquetería *lattice* [3] de las 15 réplicas simultáneas, se seleccionó las mejores representaciones; como se muestra en la figura 1, en donde se puede observar que las posiciones iniciales son proporcionadas de manera aleatoria, de modo que con las repéticiones y con un incremento de 10000 en los pasos el resultado se refinó hasta obtener la posición optima del agente, como se puede observar en la figura 1(d).



(a) Repetición 1

(b) Repetición 2



(c) Repetición 3

(d) Repetición 5

Figura 1: Proyección plana del experimento a 10000 pasos

3.1. Reto1

Para realizar el reto 1, se tomo como ejemplo lo anteriormente reportado [2], como se muestra en las líneas siguientes.

```
g <- function(x, y) {
  a<- (((x + 0.5)^4 - 30 * x^2 - 20 * x + (y + 0.5)^4 - 30 * y^2 - 20 * y)/100)
  return(a)
}
valeng <- c()
temperaturas <- c(5,10,15,20,40)
for(tem in temperaturas){
  low <- -3
  high <- -low
  step <- 0.25
```

```
replicas <- 15
t <- tem
ep <- 0.80
replica <- function(t){
  curr <- c(runif(1, low, high), runif(1, low, high))
  best <- curr
  for (tiempo in 1:t) {
    delta <- runif(1, 0, step)
    x1 <- curr + c(-delta,0)
    x2 <- curr + c(delta,0)
    y1 <- curr + c(0,-delta)
    y2 <- curr + c(0,delta)
    puntos <- c(x1,x2,y1,y2)
    for(k in 1:8){
      if(puntos[k] < (-5)){
        puntos[k] <- puntos[k]+10
      }
      if(puntos[k] > 5){
        puntos[k] <- puntos[k]-10
      }
    }
    vecx <- c()
    vecy <- c()
    for(p in 1:8){
      if(p %%2 == 0){
        vecy <- c(vecy,puntos[p])
      }else{
        vecx <- c(vecx,puntos[p])
      }
    }
    u <- sample(1:4,1)
    x.p <- c(vecx[u],vecy[u])
    delt <- g(x.p[1],x.p[2]) - g(curr[1],curr[2])
    if(delt > 0){
      curr <- x.p
    }else{
      if(runif(1)< exp((delt) / (t * ep))){
        curr <- x.p
        if(t == 1){
          t <-t
        }else{
          t <- t-1
        }
      }
    }
    if(g(curr[1],curr[2]) > g(best[1],best[2])){
      best <- curr
    }
  }
  return(best)
}
tmax <- 100
resultados <- c()
for(indi in 1:100){
  resultados <- c(resultados, replica(tmax))
```

```
  }
  vecx <- c()
  vecy <- c()
  aux <- 200
  for(p in 1:aux){
    if(p %%2 == 0){
      vecy <- c(vecy,resultados[p])
    }else{
      vecx <- c(vecx,resultados[p])
    }
  }
  valores <- c()
  for(q in 1:100){
    valores <- c(valores, g(vecx[q], vecy[q]))
  }
  valeng <- c(valeng, valores)
}
```

En la figura 2 se puede observar el valor del recocido a diferentes temperaturas a un valor de ξ de 0.1, en donde se puede determinar que la temperatura no altera el resultado final, debido a que hay una diferencia despreciable como se muestra en las cajas bigote.

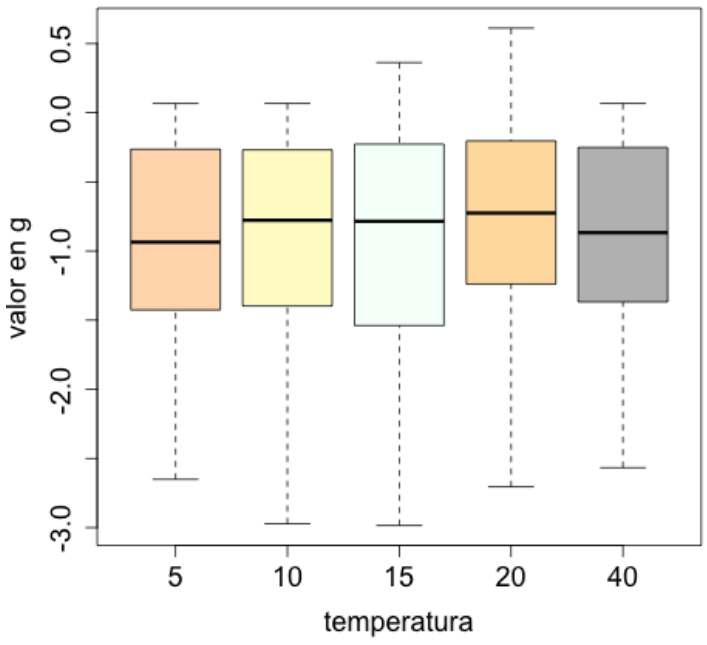


Figura 2: Resultados de recocido simulado

Referencias

[1] Ricardo Rosas Macías. Práctica 7: búsqueda local, 2019. URL https://github.com/RicardoRosMac/Simulation/tree/master/HWP7.

[2] Ricardo Parga. Práctica 7: búsqueda local, 2018. URL https://github.com/RParga/R-simulation/tree/master/p7.

[3] Deepayan Sarkar. Lattice: trellis graphics for r, 2011. URL http://lattice.r-forge.r-project.org.

[4] Elisa Schaeffer. Práctica 7: búsqueda local, 2019. URL https://elisa.dyndns-web.com/teaching/comp/par/p7.html.