Algoritmo genético

Abraham Azael Morales Juárez 1422745

9 de abril de 2019

1. Introducción

En esta práctica se trata de la implementación de un algoritmo genético y compararlo con un algoritmo exacto que en este caso es el de mochila (en inglés "knapsack"), que es un problema clásico de optimización, el cual consiste en seleccionar objetos de un grupo en específico pero que no exceda la capacidad de soporte en peso que tenemos y que además el valor de los objetos sea el máximo posible [1].

2. Objetivos

Paralelizar el algoritmo genético con el algoritmo exacto para poder observar las diferencias entre los tiempos de estas y observar en que casos el tamaño de la instancia del algoritmo genético es mejor que el algoritmo exacto en términos de valor total obtenido por segundo de ejecución [1].
Generar condiciones con estas 3 reglas:
El peso y el valor de cada objeto se generan independientemente con una distribución normal.
El peso de cada objeto se generan independientemente con una distribución normal y su valor es correlacionado con el peso, con un ruido normalmente distribuido de baja magnitud.
El peso de cada objeto se generan independientemente con una distribución normal y su valor es inversamente correlacionado con el peso, con un ruido normalmente distribuido de baja magnitud.

3. Resultados

Se tomaron en cuenta varios parámetros para realizar está práctica los cuales se observan en el siguiente fragmento del código donde además se le agrego un código para que porporcione el tiempo en que tarda en realizar los análisis.

```
for(init in c(20, 50, 80)){
  print(init)
  for(replica in 1:5){
    n <- 50
    pesos <- generador.pesos(n, 2, 30)
    valores <- generador.valores(pesos, 10, 200)
    capacidad <- round(sum(pesos) * 0.65)
    optimo <- knapsack(capacidad, pesos, valores)
    pm <- 0.05
    rep <- 50
    tmax <- 50
    startpar=Sys.time()
    p <- as.data.frame(t(parSapply(cluster,1:init,function(i){return(round(runif(n)))})))
      clusterExport(cluster,"p")
      mutan=sample(1:tam,round(pm*tam)) #Elegir cuales van a mutar con pm
      p <- rbind(p,(t(parSapply(cluster,mutan,function(i){return(mutacion(unlist(p[i,]), n))}))))

      clusterExport(cluster,"tam")
      clusterExport(cluster,"p")
      padres <- parSapply(cluster,1:rep,function(x){return(sample(1:tam, 2, replace=FALSE))})
      clusterExport(cluster,"padres")
      hijos <- parSapply(cluster,1:rep,function(i){return(as.matrix(unlist(reproduccion(p[padres[1,i
          ],], p[padres[2,i],], n)),ncol=n))})
      p = rbind(p,hijos)
```

```
startSec=Sys.time()
    p <- poblacion.inicial(n, init)
    tam <- dim(p)[1]
    pm <- 0.05
    rep <- 50
    tmax <- 50
    mejores <- double()
    for (iter in 1:tmax) {
      p$obj <- NULL
      p$fact <- NULL
      for (i in 1:tam) { # cada objeto puede mutarse con probabilidad pm
        if (runif(1) < pm) {
          p <- rbind(p, mutacion(p[i,], n))
        }
      }
      for (i in 1:rep) { # una cantidad fija de reproducciones
        padres <- sample(1:tam, 2, replace=FALSE)
        hijos <- reproduccion(p[padres[1],], p[padres[2],], n)
        p <- rbind(p, hijos[1:n]) # primer hijo
        p <- rbind(p, hijos[(n+1):(2*n)]) # segundo hijo
      }
```

Al momento de realizar la paralelización se varió la población inicial y los resultados se pueden observar en la figura 1, en donde se puede observar que el paralelismo realmente disminuyo los tiempos los cuales se realiza el análisis.

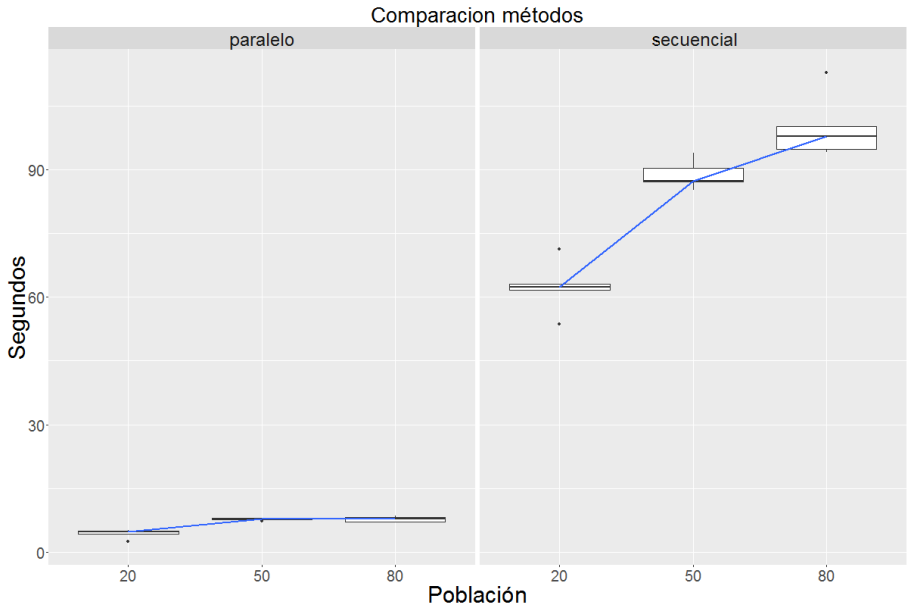

Figura 1: Comparación de los tiempos entre los métodos secuencial y paralelo. Línea azul une las medias de las poblaciones.

4. Conclusiones

Se logró una optimización en los tiempos del análisis de los logaritmos, lo cuál se demuestra la importancia que tiene este método para el análisis de datos.

Referencias

[1] Satu Elisa Schaeffer. Práctica 10: algoritmo genético, 2019. URL `https://elisa.dyndns-web.com/teaching/comp/par/p10.html`.