Práctica 10: algoritmo genético

17 de octubre de 2017

1. Introducción

En esta tarea se aborda uno de los problemas más populares y trabajados en la rama de la optimización, se trata de el problema de la mochila, el cual consiste en decidir que objetos colocas o seleccionas para llevar en una mochila, considerando que tiene cierta capacidad de objetos y cuyo objetivo es obtener el máximo beneficio. La tarea diez consiste en paralelizar el código dado, y argumentar estadisticamente que la paralelización efectivamente entrega una solución en menor tiempo que de la forma secuencial.

2. Parámetros de trabajo

La experiemtnación se realizó en un HP Z230 Tower Workstation con procesador Intel(R) Xenon(R) CPU E3-1240 v3 y 3.40 GHz de memoria ram 16 GB y un sistema operativo de 64 bits con Windows 7 Home Premium.

La población inicial a lo largo de la práctica varía entre $n \in \{20, 50, 100, 200, 350, 400\}$, seleccionando estos valores tras un diseño de experimentos siendo los más convenientes en tiempos, como para esta parte solo nos interesa comparar los tiempos y es de suponerse que el tiempo que tardo en realizar una generación es similar, el número de generaciones establecido para esta primer fase es de $g = \{5\}$.

3. Modificaciones del código

Se agregaron los comandos necesarios para la paralelización, para efecto práctico se mostrará el modo de paralelizar solo una de las funciones, la función `mutar`, de forma similar se realizó para el resto de las mismas, tomando en cuenta que no toda función que se establece en el código base se paralelizó ya que algunas por su modo de estructura es simplemente ilógico.

```
library(parallel)
cluster <- makeCluster(detectCores() - 1)
  clusterExport(cluster, "pm")
  clusterExport(cluster, "p")
  clusterExport(cluster, "mutacion")
  clusterExport(cluster, "n")
  clusterExport(cluster, "tam")
  clusterExport(cluster, "fun.mutar")
  ...
  vec.mutados <- parSapply(cluster, 1:tam, fun.mutar)
  vec.mutados <- unlist(vec.mutados)
  ...
  stopCluster(cluster)
```

Así mismo se incluyeron los comandos de medición de tiempo y los ciclos `for` necesarios para la automatización del cambio de población inicial. De igual forma los resultados de los tiempos se mandan guardar a un archivo tipo `csv`, los cuales en un inicio están almacenados en una variable tipo `data.frame`, esto con el fin de ir salvando los tiempos por ciclo de población inicial.

```
tiemposec <- data.frame()
...
valss<-c(20,50,100,200,350,400)
for(i in 1:(length(valss))){
init <- valss[i]
...
tmax <- 5
....
for (iter in 1:tmax) {
c <- Sys.time()
...
 d <- Sys.time()
 ti <- c(c,d)
 tie <- diff(ti,units="secs")
 tiemposec <- rbind(tiemposec,c(init,tie))
 } }
 ...
 write.csv(tiemposec, file="TiemposP.csv")
```

4. Resultados y Conclusión

En la figura 1 podemos observar de forma clara la diferencia de tiempos de ejecución. Las instrucciones de la práctica especifican que se requiere una prueba estadística para la justificación de la paralelización, pero al notar la gran diferencia de tiempos se cree innecesario realizarla, es por esto que se cambiaron los tamaños de la población con el fin de observar si al ser una muestra más pequeña de población en algún punto era mejor la forma secuencial del código, pero ni de esta manera los tiempos llegan a aproximarse, es sin duda la forma paralelizada la que requiere menos de una tercera parte del tiempo en forma secuencial.

5. Reto 1

El primer reto consiste en modificar la función de selección de padres de tal forma que el algoritmo tome los padres en base a una probabilidad establecida en proporción con su aportación al valor objetivo. En este caso la finalidad no es ver el cambio en los tiempos de ejecución sino la mejora en la solución final, es decir en mi diferencia con el óptimo del problema.

```
Shapiro-Wilk normality test

data:  ambos$Diferencia
W = 0.9887, p-value = 0.7114
```

```
Wilcoxon rank sum test

data:  ambos$Diferencia[ambos$Tipo == "Probabilidad"] and
ambos$Diferencia[ambos$Tipo == "Normal"]
W = 383, p-value = 3.795e-05
alternative hypothesis: true location shift is not equal to 0

> median(ambos$Diferencia[ambos$Tipo=="Probabilidad"])
[1] 0.09074176
> median(ambos$Diferencia[ambos$Tipo=="Normal"])
[1] 0.1116319
```

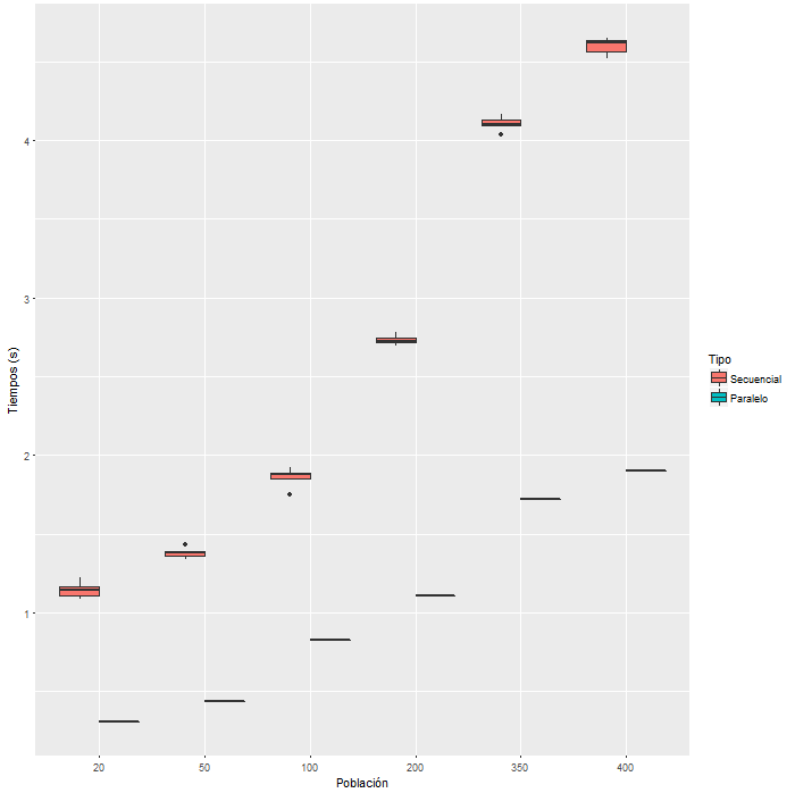

Figura 1: Tiempos de ejecución por tamaño de población para el código secuencial, así como para el paralelizado.