PROYECTO 2: BÚSQUEDAS CON INFORMACIÓN

RANDOM RESTART HILL CLIMBING SEARCH
LOCAL BEAM SEARCH

IAN SAUCY
CÉSAR ARÉVALO

UNIVERSIDAD DE DEUSTO
FACULTAD DE INGENIERÍA
SISTEMAS INTELIGENTES
BILBAO, ESPAÑA
2020

Índice

INTRODUCCIÓN

Para el presente proyecto se llevó a cabo la ejecución del problema MORE PIZZA, el cual es un ejercicio utilizado en una competición hashcode de Google, esto se realizó bajo la solución de los algoritmos Random Restart Hill Climbing Y Local Beam Search, partiendo del algoritmo Hill Climbing Básico suministrado, el cual se modificó de manera respectiva para obtener los resultados esperados.

LA FUNCIÓN PARA GENERAR LOS DATOS

```
analyze.execs <- function(res, v, nombre, param1, param2){
  df <- data.frame(matrix(unlist(res), nrow=length(res), byrow=T))
  colnames(df) <- c("time", "cost", "count","slices")
  before <- nrow(df)
  df <- df[!(df$cost > df$slices[1]),]
  after <- nrow(df)
  if(nombre == "Local Beam Search"){
    cat(paste0(nombre, " ran ", v, " times with ", param1," beams. \nRemoved ", before-after, " invalid results from analysis \n"))
  }else{
    cat(paste0(nombre, " ran ", v, " times with ", param1, " restarts and ", param2, " max iterations. \nRemoved ", before-after, " invalid results from analysis \n"))
  }
  cat(paste("Mean Time:", round(mean(df$time),4),  " Time Standard Dev:", round(sd(df$time), 3), sep="\t"), "\n")
  cat(paste("Mean Slices:", round(mean(df$cost), 2), " Slices Standard Dev:", round(sd(df$cost), 3), sep="\t"),"\n")
  cat(paste("Mean Iterr:", round(mean(df$count), 4),  " Itrr Standard Dev:", round(sd(df$count), 3), sep="\t"),"\n")
}
```

La función acumula y realiza las operaciones respectivas para generar el promedio ponderado, la desviación típica, sobre los resultados obtenidos tras cada una de las ejecuciones, se encuentra dentro del fichero more-pizza-main.R

RANDOM RESTART HILL CLIMBING

Resultados:

```
Random Restart Hill Climbing ran 50 times with 500 restarts and 500 max iterations.
Removed 0 invalid results from analysis
Mean Time:      0.1528   Time Standard Dev:     0.121
Mean Slices:    4500     Slices Standard Dev:   0
Mean Iterr:     4.48     Itrr Standard Dev:     5.304

>  execute.random.restart("../data/2-medium.txt", 10, 500, 50)
Random Restart Hill Climbing ran 50 times with 10 restarts and 500 max iterations.
Removed 2 invalid results from analysis
Mean Time:      0.1171   Time Standard Dev:     0.069
Mean Slices:    4499.94  Slices Standard Dev:   0.245
Mean Iterr:     3.5208   Itrr Standard Dev:     3.275

>  execute.random.restart("../data/2-medium.txt", 500, 10, 50)
Random Restart Hill Climbing ran 50 times with 500 restarts and 10 max iterations.
Removed 14 invalid results from analysis
Mean Time:      4.5034   Time Standard Dev:     1.908
Mean Slices:    4499.17  Slices Standard Dev:   1.134
Mean Iterr:     376.1389         Itrr Standard Dev:     161.206
```

Análisis:

Según la teoría sabemos que el Random Restart Hill Climbing (RRHC) tiene una probabilidad casi del 100% de llegar a la solución más óptima si se ejecuta la suficiente cantidad de reinicios.

El primer resultado muestra que el RRHC es completo, había suficientes reinicios para llegar al final a la solución más óptima. Como la media de los trozos es cero, significa que cada vez que se ejecuta el RRHC se encuentra la solución óptima.

En cambio, la segunda ejecución muestra la posibilidad de que no hay suficientes reinicios para tener esta probabilidad que se acerca a 100%. Diez reinicios no fueron suficientes para llegar al solución final siempre, por eso vemos una media de 4499.94 trozos y un desviación estándar de 0.245. Estos datos insinúan que aunque la mayoría del tiempo llega la solución más óptima no siempre lo logra.

El ejemplo final muestra la conexión entre los parámetros del Hill Climbing(HC) normal y el RRHC. Aunque habían suficientes reinicios no habían suficientes iteraciones en el HC para que cada iteración del HC dentro de el RRHC pueda tener la posibilidad de llegar a un solución óptima. Por eso, la media de los iteraciones del RRHC (la cantidad de veces que reiniciaron el HC) es mucho más alto que el primero ejemplo. Eso fue porque el algoritmo tenía que reiniciar más veces para llegar a una solución, que tampoco fue lo óptima.

En relación al tema de los tiempos de ejecución los dos primeros son similares debido a que la cantidad de iteraciones alcanzadas en ambos ejemplos oscila sobre el número 5, esto infiere una ejecución de tiempo similar. Obviamente, el final ejemplo como tiene una media de reinicios mucha más alta infiere una ejecución de tiempo más largo.

LOCAL BEAM SEARCH

Resultados:

```
> execute.local.beam("../data/2-medium.txt", 1, 5000)
Local Beam Search ran 5000 times with 1 beams.
Removed 1859 invalid results from analysis
Mean Time:      0.0137   Time Standard Dev:     0.006
Mean Slices:    4498.24  Slices Standard Dev:   1.789
Mean Iterr:     18.908   Itrr Standard Dev:     3.235

> execute.local.beam("../data/2-medium.txt", 3, 5000)
Local Beam Search ran 5000 times with 3 beams.
Removed 1332 invalid results from analysis
Mean Time:      0.047    Time Standard Dev:     0.012
Mean Slices:    4499.43  Slices Standard Dev:   0.782
Mean Iterr:     24.0338  Itrr Standard Dev:     3.256

> execute.local.beam("../data/2-medium.txt", 5, 5000)
Local Beam Search ran 5000 times with 5 beams.
Removed 947 invalid results from analysis
Mean Time:      0.0875   Time Standard Dev:     0.019
Mean Slices:    4499.71  Slices Standard Dev:   0.533
Mean Iterr:     30.0851  Itrr Standard Dev:     6.845

> execute.local.beam("../data/2-medium.txt", 10, 5000)
Local Beam Search ran 5000 times with 10 beams.
Removed 354 invalid results from analysis
Mean Time:      0.2419   Time Standard Dev:     0.073
Mean Slices:    4499.91  Slices Standard Dev:   0.29
Mean Iterr:     50.6847  Itrr Standard Dev:     17.429
```

Análisis:

Local Beam Search(LBS) viene en muchas variedades, lo que presentamos aquí es una variante que mantiene K número de beams, para que en cada iteración del algoritmo expanda cada beam y seleccione el mejor estado(entre los nuevos sucesores y el estado actual). El algoritmo mantiene cada beam separado con sus propios sucesores y estados actuales.

Según la teoría, beam search disminuye la probabilidad de que la solución está situada en un máximo o mínimo local. Trivialmente este es una característica de los beams, como cada beam empieza en un estado aleatorio hay más oportunidades de encontrar una solución óptima. Por eso, debemos ver un tiempo de ejecución más largo en comparación con el HC normal y algo más con un LBS que usa más beams. Esta amplificación de tiempo tiene relación no solo en el proceso de expandir y mantener más nodos sucesores, estados actuales etc sino también en el número de iteraciones, por tanto la cantidad de veces que encuentra una solución óptima debería ser más alta, es decir que la media del costo debería ser más cerca a la cantidad requerida.

Adicionalmente, no es garantizado que LBS es completo, será completo en función del tamaño del problema, las iteraciones permitidas y la cantidad de beams. Cuanto más

iteraciones y beams se permite más se acerca a su completitud.(Pushpak Bhattacharyya; “Beam Search”)

En nuestros resultados la aplicación de la teoría es bastante visible, principalmente en la desviación estándar. Entre un beam y diez hay un diferencia de 1.5 trozos, aunque no es mucho en este caso, muestra el aumento de la probabilidad con más cantidad de beams. También se puede evidenciar la reducción de la desviación estándar en todos los pasos entre uno y diez beams.

Además, mientras aumenta los beams, también aumenta el tiempo de ejecución y el medio de iteraciones requeridas para encontrar una solución. Este relacion esta basada en la teoría porque si hay diez beams en lugar de un que hay que seguir expandiendo hasta que encuentra una solución óptima(en cualquier beam) o cuando todos los beams llegan a una máxima local o mínima. Trivialmente, este require màs iteraciones cuando hay más beams y va a tardar más tiempo en ejecución.

Nota:

Mientras que estuvimos trabajando en los algoritmos, descubrimos algo en el código de ‘local.expand.node’ que resulta un poco extraño. El costo puesto por el algoritmo en el nodo final no estaba de acuerdo con el costo calculado manualmente usando el estado final del nodo final.

Por eso, hemos cambiado esta línea de código para que use el estado posterior aplicando la acción para calcular costo. Los resultados con este cambio parecen más lógicos en relación a la teoría. Abajo incluimos los resultados sin este cambio, con el código original para dar un fondo mejor en caso de equivocamos con esto cambio. Aun así, los observaciones que hicimos en este documento no se cambia fuera de los números concretos.

```
successor              <- list()
successor$parent       <- current_node
successor$state        <- effect(current_node$state, action)
successor$actions      <- rbind(current_node$actions, action)
successor$depth        <- current_node$depth + 1
#original                                 ▼▼▼▼▼▼▼▼▼
#successor$cost          <- get.cost(action, current_node$state, problem)
successor$cost         <- get.cost(action, successor$state, problem)
successor$evaluation   <- get.evaluation(successor$state, problem)
```

```
> execute.local.beam("../data/2-medium.txt", 1, 1000)
Local Beam Search ran 1000 times with 1 beams.
Removed 112 invalid results from analysis
Mean Time:      0.014    Time Standard Dev:     0.006
Mean Slices:    4469.62  Slices Standard Dev:   19.972
Mean Iterr:     18.8119  Itrr Standard Dev:     3.176

> execute.local.beam("../data/2-medium.txt", 3, 1000)
Local Beam Search ran 1000 times with 3 beams.
Removed 107 invalid results from analysis
Mean Time:      0.0455   Time Standard Dev:     0.011
Mean Slices:    4471.74  Slices Standard Dev:   18.358
Mean Iterr:     23.6226  Itrr Standard Dev:     3.105

> execute.local.beam("../data/2-medium.txt", 5, 1000)
Local Beam Search ran 1000 times with 5 beams.
Removed 101 invalid results from analysis
Mean Time:      0.0877   Time Standard Dev:     0.02
Mean Slices:    4469.62  Slices Standard Dev:   18.829
Mean Iterr:     29.1958  Itrr Standard Dev:     6.415

> execute.local.beam("../data/2-medium.txt", 10, 1000)
Local Beam Search ran 1000 times with 10 beams.
Removed 103 invalid results from analysis
Mean Time:      0.2466   Time Standard Dev:     0.077
Mean Slices:    4471.25  Slices Standard Dev:   17.998
Mean Iterr:     50.0914  Itrr Standard Dev:     18.02
```

```
Random Restart Hill Climbing ran 50 times with 500 restarts and 500 max iterations.
Removed 2 invalid results from analysis
Mean Time:      0.2193   Time Standard Dev:     0.188
Mean Slices:    4472.6   Slices Standard Dev:   17.189
Mean Iterr:     7.0417   Itrr Standard Dev:     7.86

>  execute.random.restart("../data/2-medium.txt", 10, 500, 50)
Random Restart Hill Climbing ran 50 times with 10 restarts and 500 max iterations.
Removed 4 invalid results from analysis
Mean Time:      0.1526   Time Standard Dev:     0.079
Mean Slices:    4475.76  Slices Standard Dev:   17.075
Mean Iterr:     4.5435   Itrr Standard Dev:     3.365

>  execute.random.restart("../data/2-medium.txt", 500, 10, 50)
Random Restart Hill Climbing ran 50 times with 500 restarts and 10 max iterations.
Removed 3 invalid results from analysis
Mean Time:      4.7082   Time Standard Dev:     1.866
Mean Slices:    4457.26  Slices Standard Dev:   22.199
Mean Iterr:     401.5532         Itrr Standard Dev:     159.88
```

Bibliografía

“Beam Search.” *Wikipedia*, 24 Feb. 2020. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Beam_search&oldid=942391463.

Pushpak Bhattacharyya. *Beam Search*. Indian Institute of Technology Bombay, https://www.cse.iitb.ac.in/~cs344/2011/slides/cs344-beam-search-2feb11.pptx.