Memoria mini proyecto PD 21-22

Álvaro Maya cano - alvmaycan - varo9747@gmail.com
Félix González Marín - felgonmar - felixgm.1997@gmail.com

Propuesta de trabajo

Nuestra propuesta se basa en investigar y aplicar en haskell algoritmos previamente vistos en otras asignaturas. Para este mini proyecto, nos hemos decantado por los algoritmos de enfriamiento simulado y algoritmo genético.

Para aplicar estos algoritmos, hemos recreado algunos de los problemas más conocidos. Estos son el problema de la mochila para el algoritmo de enfriamiento simulado y el de las n reinas para el algoritmo genético.

El problema clásico de la mochila consiste en la optimización de una lista de la compra, en la que nos darán una lista de objetos que deseamos comprar y una mochila que soporta una determinada carga o peso.

Estos objetos tienen los atributos de valor, peso y nombre para una mejor identificación. En el código se creó un tipo nuevo; `type Objeto = (Int,Int,String)` para la resolución del problema.

Los objetos se pueden crear por pantalla con la ejecución del programa o con una llamada que enseñaremos en el apartado de código donde se explican las distintas funciones creadas.

Para la resolución de este problema optamos por el uso del algoritmo de enfriamiento simulado que hemos visto en otras asignaturas y del que también investigamos por nuestra cuenta.

Lo que hará nuestro algoritmo será mediante la selección aleatoria de unos llamados vecinos, comprobar si estos aportan una solución mejor de la que teníamos previamente. Para ello se hace uso de los números aleatorios en haskell. También tenemos una condición o penalización que debe ser superada, que en nuestro caso es que la lista de vecinos no supere el peso de la mochila, si esa condición no es superada los vecinos no serán valorados, aunque en nuestro caso, el algoritmo siempre escogerá vecinos que sean válidos ya que creamos una función que aunque escoge aleatoriamente los objetos dentro de la lista ya dada, estos solo podrán pasar a la lista final si es que entran dentro de los límites del peso de la mochila.

El único inconveniente del algoritmo es que tiene también cierto porcentaje en el que se aceptan ciertos resultados que pueden ser peores que los que teníamos anteriormente, ya que el algoritmo es así por definición. Sin embargo, esta probabilidad de aceptación de

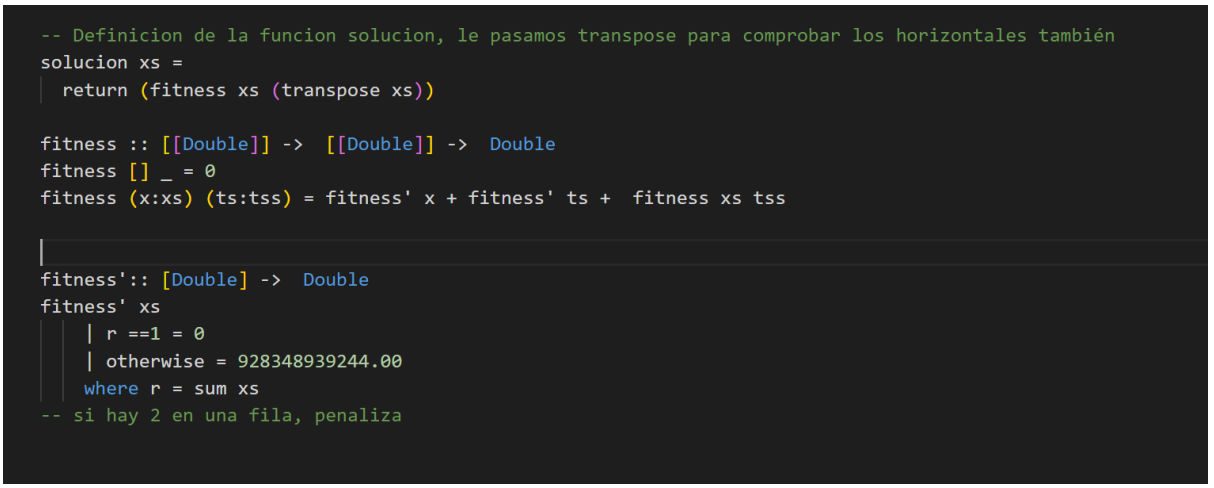
resultados peores será menor a medida que vayamos avanzando en las iteraciones para que así al final quede la mejor solución.

Código

Para el acortamiento del texto pondremos el código recortado de las funciones donde se usen las funciones que se pedían como objetivo mínimo y solo un ejemplo por cada parte aunque haya más dentro del código.

Algoritmo genético

- Fitness



```haskell
-- Definicion de la funcion solucion, le pasamos transpose para comprobar los horizontales también
solucion xs =
  return (fitness xs (transpose xs))

fitness :: [[Double]] ->  [[Double]] ->  Double
fitness [] _ = 0
fitness (x:xs) (ts:tss) = fitness' x + fitness' ts +  fitness xs tss

fitness':: [Double] ->  Double
fitness' xs
    | r ==1 = 0
    | otherwise = 928348939244.00
    where r = sum xs
-- si hay 2 en una fila, penaliza
```

En esta función definimos cómo de válido es nuestro gen. Si es = 0, entonces es válido.

- Genera población (función principal)

```haskell

-- FUNCIONES PRINCIPALES
generaPoblacion :: Int -> IO ()
generaPoblacion n = do
    individuo <- generaIndividuo (n*n)
    putStrLn ("gen: " ++ show ((parteLista individuo n)))
    s <- solucion (parteLista individuo n)
    putStrLn ("solucion: ")
    print s
    if(s > 0)
      then do
        putStrLn "NO ES SOLUCION. Empezando mutaciones: "
        mutacion (parteLista individuo n)
    else
      do
        putStrLn "SOLUCION CORRECTA"

    return ()



-- Funcion si fallamos el primer gen
mutacion :: [[Double]] -> IO()
mutacion xs = do
    s <- solucion (muta xs)
    putStrLn "MUTACION COMPLETA"
    putStrLn "SOLUCION NUEVA: "
    print s


    return ()
```

En esta función, será la que llamaremos en main cuando nos pasen el número de reinas que desean en el tablero. Crearemos una posible solución y después comprobaremos si es correcta.

- Funciones auxiliares

```haskell
-- FUNCIONES AUXILIARES
generaIndividuo :: Int -> IO [Double]
generaIndividuo n = do
  gen <- newStdGen
  let xs = randomRs (0,1) gen
  return (transforma (take n xs) probabilidad)

transforma :: [Double] ->Double -> [Double]
transforma xs n = [if (x > n) then 1 else 0 | x <- xs]

parteLista :: [a] -> Int -> [[a]]
parteLista [] _ = []
parteLista xs n = (take n xs) : parteLista (drop n xs) n
```

Aquí le daremos forma al individuo, que será nuestra posible solución y más tarde veremos si es valido.

- muta gen

```haskell
muta::[[Double]]->[[Double]]
muta [] = []
muta (x:xs) = (mutaAux x):(muta xs)

mutaAux:: [Double] -> [Double]
mutaAux xs
  | (sum xs) == 1 =  xs
  | otherwise = transforma (take (length xs) xs) 0.7
```

Función para mutar los posibles genes

Enfriamiento simulado

- Creación de módulos

```haskell
module EnfriamientoSim
  (enfriamiento_simulado,
   funcion_valor,
     sacar_valor,
     sacar_valor',
     sacar_peso,
     sacar_peso',
     caben_objetos,
```

```
    cabe_objeto
  )where
```

- Tipos algebraicos

```
type Temperatura = Double
```

- Uso de plegados

```
funcion_valorFL ::[Objeto]-> Int
funcion_valorFL xs = foldr (\x acc -> acc + (sacar_valor' x)  ) 0
xs
```

- Uso de guardas

```
cabe_objeto pesoMoch objeto
    |pesoMoch < (sacar_peso' objeto) = False
    |pesoMoch >= (sacar_peso' objeto) = True
```

- Uso de recursión

```
caben_objetos pesoMoch lista
    | pesoMoch < pesoObjeto = caben_objetos pesoMoch (tail lista)
```

- Uso de case of

```
case o of "1" -> crear_obj []
             "2" -> return ()
             _ -> comienzo
```

Compilación del programa

Para el uso del programa basta con compilar el archivo de Principal.hs y seguir las instrucciones que se muestran por pantalla.

En el caso del ejercicio de enfriamiento simulado hay algunas pruebas de algunas funciones en el archivo test_funciones.txt. Para usarlas habría que compilar el archivo de EnfriamientoSim.hs y ejecutar los ejemplos desde el ghci. El módulo de EnfriamientoSim es el que realiza todo el algoritmo y por eso mismo es donde se pueden comprobar los ejemplos.

En el caso del ejercicio de n reinas, diremos el tamaño del tablero y nos propondrá una solución que se ejecutará en el archivo Genetico.hs.

Métodos usados

Como hemos podido ver en el código, hemos usado gran cantidad de listas, recursividad, orden superior, case of… ya que al ser problemas de algoritmos no nos ha resultado difícil implementar estos casos. Lo único que no hemos podido usar ha sido los datos de tipo abstracto.