Práctica 8

*

December 23, 2015

1 Introducción:

1.1 Exposición de la práctica:

En la práctica presentada se nos pide implementar los algoritmos de optimización denominados búsqueda local(para una sola solución) y algoritmos genéticos(varias soluciones) para el conjunto de datos dantzig42. Para ello hemos desarrollado nuestro código en el lenguaje de programación Java y después hemos creado nuestro ejecutador .jar. Además hemos desarrollado este documento para plasmar las partes más importantes de la práctica.

1.2 Objetivos:

Los objetivos son claros; conseguir un algoritmo eficiente y efectivo para optimizar una solción o un conjuntos de soluciones de un conjunto de datos. El problema planteado tiene nombre propio TSP. El código que hemos desarrollado no es dependiente a un solo conjunto de datos,ya que es posible utilizarlo en más de un conjunto de datos(menos la lectura).

Como hemos recibido el archivo dantzig42 para desarrollar la práctica, vamos a analizar el archivo para ver su contenido. En el archivo de datos podemos encontrar un conjunto de valores numéricos separados por ceros. Estos ceros indican un salto de linea. Si procesamos estos datos, conseguimos la mitad una matriz de una dimensión de 49x49. Como tenemos 49 ciudades, el punto en común de los pares de ciudades nos indica cual es el peso de una ciudad a otra.

Para optimizar nuestras ciudades vamos a aplicar distintos criterios en el proceso. Empezaremos utilizando el criterio Best First y luego utilizaremos el Greedy, para la búsqueda local. Una vez conseguidos estos procesos, haremos el calculo de el algoritmo genético.

2 Explicación y Pseudocódigo del algoritmo:

3. Experimentación. (Las distintas configuraciones de parámetros que se han utilizado, y los resultados obtenidos. Cuantas repeticiones...)

4. Explicar las conclusiones obtenidas, haciendo especial énfasis en el mejor diseño conseguido. ¿Cual de los dos algoritmos funciona mejor sobre la instancia proporcionada? ¿Podría decirse que uno de los algoritmos es significativamente mejor que otro?

P.D. Siguiendo a la buena costumbre de las practicas. Quiero que me entregueis un JAR con el codigo (o dos JARs, uno por cada algoritmo). Cuando se ejecute el JAR, este repetira la optimizacion 10 veces, y devolvera la lista de resultados obtenidos, asi como la mejor solucion y el fitness medio de todas las soluciones obtenidas.

En el algoritmo de búsqueda local se recibe una solución la cual se pretende optimizar. La solución recibida se utilizará para calcular una combinatoria de todos los swaps posibles entre dos de las ciudades del camino. Una vez calculados estos nuevos caminos, se buscará el mejor camino de todos, o el primero mejor que encontremos(Greedy,BestFirst) comparándolos con nuestra solución de entrada. Si encontramos un camino mejor que el de nuestra entrada, el camino, pasará a ser la nueva entrada, y así, seguiremos hasta que ningún vecino supere a nuestra solución de entrada.

*

entrada: Listaciudades
salida : ListaCiudadesOptima
while $Valor(ListaCiudades) >= Valor(ListaCiudadesOptima)$ & $!TiempoComputacionalAcabado$ & $!Numerodeiteracionesmáximo$ **do**
  $Combinatoria = CrearCombinatoriaDeSwaps()$
  $ListaCiudadesOptima = BusquedaMejorLista(Combinatoria)$
  if $ListaCiudadesOptima <= ListaCiudades$ **then**
    return $ListaCiudades$
  else
    $ListaCiudades = ListaCiudadesOptima$
  end
end
return $ListaCiudades$

Algorithm 1: Búsqueda Local

En el algoritmo genético se recibe una población de soluciones para optimizar. Estas soluciones se ordenarán en base a su valor de salida(adaptación). Una vez obtenida la lista ordenada, se procederá a recoger la mitad de las muestras en sentido de valor de salida descendente. De estas muestras conseguidas, se harán cruces. Estos cruces crearán el mismo número de hijos que padres hemos cogido.

Para hacer los cruces hemos utilizado el algoritmo PMX (Partially Mapped Crossover). En este algoritmo un segmento de uno de los padres pasará en orden al hijo. La información restante ocupará los trozos de información vacíos siempre que la información no este ya incluida en el primer segmento. He aquí el algoritmo.

entrada: ListaCiudades1(P1),ListaCiudades2(P2)
salida : ListaCiudadesHijo1,ListaCiudadesHijo2
while $Numerohijos < 2$ **do**
  Elegir segmento de azar y copiarlo de P1
  Comenzando desde el primer punto de cruce buscar elementos en ese segmento de P2 que no han sido copiadas
  for $Paracadaelemento$ **do**
    buscar en la descendencia para ver qué elemento j ha sido copiado en su lugar de P1
    Coloque i en la posición ocupada por j en P2, ya que sabemos que no tendremos que poner j allí
    if $ellugarocupadoporjenP2yahasidorellenadoenladescendenciak$ **then**
      poner i en la posición ocupada por k en P2
    end
    Después de haber tratado con los elementos del segmento crossover, el resto de la descendencia se puede llenar desde P2
  end
end
return $ListaCiudadeshijo1, ListaCiudadesHijo2$

Algorithm 2: PMX: Partially Mapped Crossover

Los hijos sufrirán mutaciones en el 30 por ciento de los casos(criterio establecido por nosotros) y se hará un nuevo conjunto de datos entre los padres y los hijos mutados y no mutados. De esta forma se obtendrá una población del mismo número de la que teníamos de entrada.

El algoritmo puede converger en estos tres casos:

- Tiempo computacional.
- Numero de iteraciones.
- Población de rasgos muy similares.

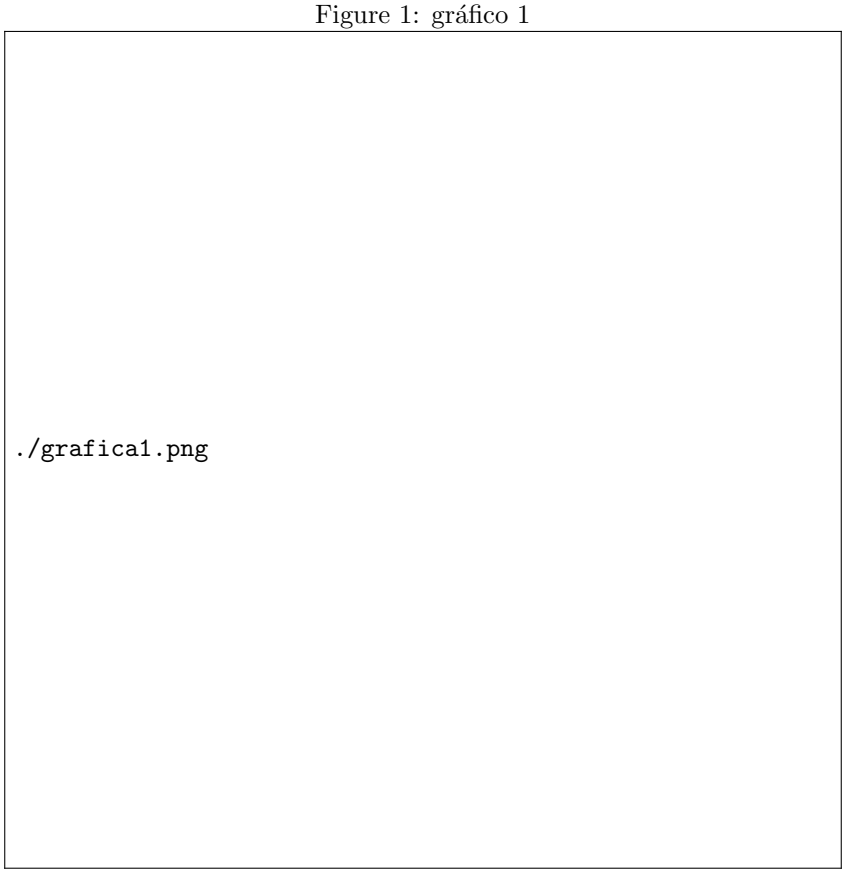
entrada: PoblacionListaCiudades
salida : PoblacionListaCiudadesMejoradas
while $!TiempoComputacionalAcabado$ & $!Numerodeiteracionesmáximo$ **do**
  $PoblacionOrdenada = OrdenarPoblacionListaCiudades()$
  $Cruces = CrearCruces(PoblacionOrdenada/2)$
  $mutaciones = CrearMutaciones(Cruces)$
  $PoblaciónListaCiudadesMejoradas =$
  $ReemplazarMutacionesPorMuestrasDeBajoNivel(PoblacionOrdenada, Mutaciones)$
end
return $PoblaciónListaCiudadesMejoradas$

Algorithm 3: Algoritmo Genético

3 Experimentación y Resultados:

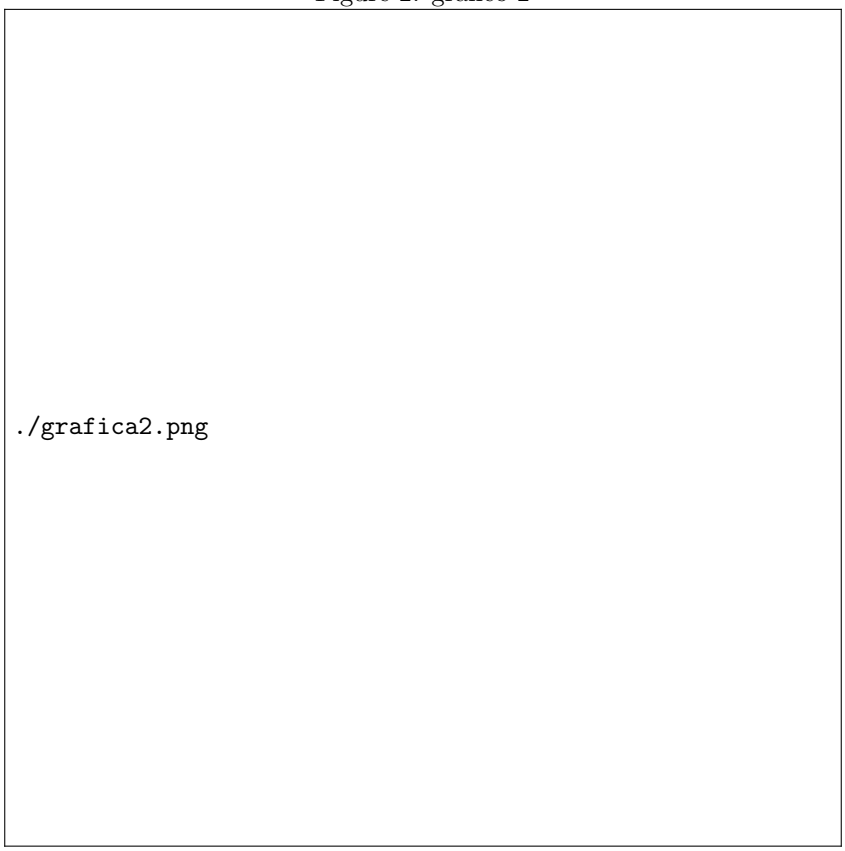
Hemos ejecutado el programa y nos han salido estos resultados:

Figure 1: gráfico 1



Aquí escribimos conclusiones

Figure 2: gráfico 2



En cuanto al gráfico del tiempo vemos que cuanto más grande es el parámetro k el tiempo de ejecución para cada partición es mayor. Esto quiere decir, que cuantos más clusters existan, más distancias se deben de calcular entre los centroides y las instancias, por lo tanto, un mayor tiempo de ejecución.

4 Conclusiones:

5 Valoración Subjetiva:

Ieltzu: Ha sido una práctica en la que no hemos invertido mucho tiempo. Los tres teníamos que haber estudiado los dos algoritmos, por lo tanto, no ha sido muy complicado implementarlo. Práctica interesante, pero en la época del año que nos ha pillado, para mi, sobraba.
Mikel:
Maria:

Bibliografia

- http://www.herrera.unt.edu.ar/gapia/Curso$_A$G/$Curso_AG_08_Clase_5.pdf$
$http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t2geneticos.pdf$