Práctica 8

*

December 27, 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:

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 Order 1 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 en hijo
  for $Paracadaelementofueradelsegmento, empezandodesdeelfinaldelsegmento$ **do**
    if $elelementoseleccionadodeP2noestáenelsegmentoseleccionadodeP1$ **then**
      colocarlo en orden después del segmento en P3(si lista está llena se empezará desde el principio hasta el principio del segmento)
    end
  end
end
return $ListaCiudadeshijo1, ListaCiudadesHijo2$

Algorithm 2: Order 1 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:

Comprobaremos si es mejor empezar con un aleatorio uniforme o sesgada:

Aleatorio uniforme:
Distancia Media: 3112.9
Tiempo de ejecución Medio: 0

Aleatorio sesgado:
Distancia Media: 2450.0
Tiempo de ejecución Medio: 31

Para la selección de tipo de aleatoriedad nos basaremos en la distancia.
Elegimos aleatoriedad sesgada que tiene distancia más baja.

Analizaremos como funciona el Greedy: (10 vueltas)

Cada vuelta:

1 [20,27,3,40,41,0,1,2,7,8,25,26,6,5,34,33,32,31,29,30,4,39,38,37,36,35,28,21,12,13,14,15,22,23,24,9,11,10,16,17,18,19] - 998.0

2 [35,32,31,9,10,11,12,17,18,19,20,27,28,29,25,26,23,24,6,5,3,4,38,39,40,41,0,1,2,13,14,15,16,22,21,8,7,30,33,34,37,36] - 1005.0

3 [10,11,9,5,4,38,39,3,6,26,25,24,8,7,2,40,41,0,1,23,15,14,13,12,22,21,27,29,30,34,37,36,35,33,31,32,28,20,19,18,17,16] - 1005.0

4 [5,4,38,37,36,35,34,33,23,10,9,8,7,24,26,25,39,40,0,1,41,29,30,6,3,2,11,12,16,20,21,22,15,13,14,17,18,19,27,28,32,31] - 1020.0

5 [22,24,31,32,5,6,2,3,4,33,34,35,36,37,38,39,1,0,41,40,11,15,17,14,13,12,10,23,26,25,27,28,29,30,7,8,9,16,18,19,20,21] - 1018.0

6 [40,41,0,1,39,34,29,30,33,35,36,37,38,4,5,6,21,22,15,14,13,12,16,20,27,28,32,31,7,8,9,24,25,26,23,10,11,17,18,19,2,3] - 1018.0

7 [23,24,9,4,38,39,22,21,19,20,27,28,29,31,32,33,30,5,6,7,8,2,1,0,41,40,3,26,18,17,14,13,15,16,12,11,10,34,35,36,37,25] - 1115.0

8 [8,9,10,11,12,13,14,15,2,1,0,41,40,39,33,30,29,31,32,28,27,21,22,23,25,26,20,19,18,17,16,24,3,4,38,37,36,35,34,5,6,7] - 986.0

9 [35,36,1,0,41,40,33,34,37,38,39,3,2,8,9,11,10,23,29,31,30,25,26,20,19,18,17,14,13,12,7,4,5,6,24,22,15,16,21,27,28,32] - 1046.0

10 [13,12,9,8,7,3,40,41,0,1,2,24,23,21,22,11,10,25,30,33,34,5,6,4,39,38,37,36,35,32,19,20,27,28,31,29,26,16,18,17,15,14] - 1027.0

Se tarda una media de 52ms para ejecutarse cada greedy.
La media de distancia: 1023.8

La mejor opción salida:

,10,11,12,13,14,15,2,1,0,41,40,39,33,30,29,31,32,28,27,21,22,23,25,26,20,19,18,17,16,24,3,4,38,37,36,35,34,5,6,7
- 986.0

Analizaremos como funciona el BestFirst: (10 vueltas)

Cada vuelta:

1 [14,15,16,22,21,27,28,26,25,29,30,5,6,7,8,9,23,24,2,3,39,38,1,41,40,0,17,18,19,20,32,31,33,34,35,36,37,4,10,11,12,13] - 1004.0

2 [25,24,1,0,41,40,3,23,22,15,17,14,13,12,21,28,32,31,29,30,35,36,37,38,39,4,34,33,5,6,2,7,8,9,11,10,16,18,19,20,27,26] - 998.0

3 [35,39,40,41,0,1,4,6,5,33,30,27,20,21,22,10,11,9,8,7,2,3,38,37,36,34,31,28,16,12,13,14,15,17,18,19,25,26,23,24,29,32] - 943.0

4 [24,9,10,11,12,13,14,17,15,16,32,35,36,37,4,38,39,40,41,1,0,3,2,34,33,31,28,27,21,22,23,29,30,8,7,6,5,25,26,20,18,19] - 1030.0

5 [17,16,4,39,38,35,32,31,34,36,37,1,41,40,0,15,14,13,12,11,10,8,7,2,3,6,5,33,30,29,24,9,23,22,21,26,25,27,28,20,19,18] - 1074.0

6 [12,13,14,15,17,18,19,32,35,36,37,38,39,40,41,0,1,2,8,9,11,10,24,7,6,5,34,33,31,30,29,28,27,25,26,23,22,21,16,20,4,3] - 993.0

7 [19,20,30,4,38,37,36,35,34,33,5,6,7,8,2,3,39,40,41,0,1,9,11,10,22,21,23,24,29,31,32,28,27,25,26,16,12,13,14,15,17,18] - 903.0

8 [15,17,14,13,12,9,8,7,6,5,37,36,35,34,33,31,2,3,40,41,0,1,39,38,4,24,26,25,29,30,23,10,11,22,21,20,18,19,32,28,27,16] - 1012.0

9 [7,2,1,0,41,40,39,38,37,4,3,22,21,20,16,10,11,9,8,36,35,34,25,26,19,18,17,15,14,13,12,6,5,33,30,29,31,32,28,27,23,24] - 1029.0

10 [15,17,14,13,10,23,6,5,4,3,40,41,0,1,2,22,21,16,18,19,20,28,29,31,32,35,36,37,39,38,34,33,30,27,26,25,24,7,8,9,11,12] - 983.0

Se tarda una media de 49ms para ejecutarse cada BestFirst.
La media de distancia: 996.9

La mejor opción salida:

,30,4,38,37,36,35,34,33,5,6,7,8,2,3,39,40,41,0,1,9,11,10,22,21,23,24,29,31,32,28,27,25,26,16,12,13,14,15,17,18
- 903.0

Analizaremos como funciona el GeneticAlgorithm: (10 vueltas)

Cada vuelta:

1 [3,2,7,8,6,5,30,31,29,28,27,25,26,23,24,9,11,10,22,21,20,16,12,13,14,15,17,18,19,1,0,41,40,39,38,37,34,33,32,35,36,4] - 856.0

2 [30,29,25,27,28,32,31,33,34,35,36,37,38,4,5,6,7,8,24,26,21,22,23,2,3,9,11,12,13,14,15,17,18,19,20,16,10,1,0,41,40,39] - 972.0

3 [19,18,17,14,13,15,16,12,11,9,10,22,21,23,26,5,33,31,32,28,27,25,24,8,7,6,2,3,4,38,39,40,41,0,1,37,36,35,34,30,29,20] - 844.0

4 [4,3,2,6,5,30,22,21,20,27,26,25,24,23,10,11,9,8,7,1,0,41,40,39,12,16,15,13,14,17,18,19,28,29,31,32,33,34,35,36,37,38] - 925.0

5 [7,8,9,11,10,23,24,25,26,22,21,20,27,28,29,30,31,32,33,34,35,40,41,0,1,2,12,13,14,15,17,18,19,16,3,4,39,38,37,36,5,6] - 948.0

6 [30,33,34,36,35,32,31,28,27,23,24,9,8,7,3,4,37,38,39,40,41,0,1,2,6,5,19,18,17,16,21,22,10,11,12,13,14,15,20,26,25,29] - 897.0

7 [26,25,8,7,30,31,32,33,6,5,34,35,36,37,38,39,4,3,2,1,0,40,41,29,28,19,18,14,13,12,15,17,16,22,23,24,9,11,10,21,20,27] - 939.0

8 [36,37,4,3,2,7,8,6,5,30,29,28,27,21,16,18,19,20,26,25,24,9,10,11,12,13,14,17,15,22,23,1,0,41,40,39,38,35,32,31,33,34] - 903.0

9 [28,27,6,2,3,4,37,36,34,33,30,29,25,26,23,24,9,10,22,21,20,19,18,17,16,15,14,13,12,11,8,7,5,39,40,41,0,1,38,35,32,31] - 841.0

10 [23,30,31,32,33,34,35,36,37,38,39,4,6,7,8,9,24,25,26,27,28,29,5,2,1,0,41,40,3,21,20,19,18,10,11,12,13,14,17,15,16,22] - 937.0

Se tarda una media de 1614ms para ejecutarse cada geneticAlgorithm.
La media de distancia: 906.2

La mejor opción de salida:

,6,2,3,4,37,36,34,33,30,29,25,26,23,24,9,10,22,21,20,19,18,17,16,15,14,13,12,11,8,7,5,39,40,41,0,1,38,35,32,31
- 841.0

Aquí escribimos conclusiones

4 Conclusiones:

Implementar los algoritmos nos ha llevado a sacar varias conclusiones. El algoritmo de búsqueda local usando Greedy o BestFirst como criterio es mucho más fácil de implementar que el algoritmo genético. Planificar los cruces ha sido la parte mas costosa del problema en los algoritmos genéticos. Una vez que se sabe como funciona bien el algoritmo es sencillo implementarlo.
El algoritmo genético ofrece mejores resultados que el de búsqueda local. Según el numero de población va aumentando los resultados de los algoritmos genéticos aumenta, al igual que su costo computacional. La búsqueda local ofrece un resultado bueno sin mucho costo computacional. Si les diéramos el mismo tiempo de ejecución a los dos algoritmos sin duda el algoritmo de búsqueda local daría mejores resultados. Respondiendo a la pregunta de que algoritmo funciona mejor con la solución base, la respuesta es el algoritmo genético aunque no ofrece unos resultados tan diferentes como esperábamos.
En nuestra opinión no es un algoritmo que sea mejor, simplemente que con un mayor número de iteraciones y mayor tiempo computacional ofrece mejores resultados que la búsqueda local, ya que, este algoritmo no mejora tanto aunque el tiempo computacional aumente considerablemente.

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