Práctica 7

*

December 22, 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 Pseudocódigo del algoritmo:

1. Introducción.

2. Algoritmo implementados: busqueda local y algoritmo genetico. Ojo, los pseudocódigos se utilizan para ilustrar las explicaciones dadas en el texto. Por si solos, no explican nada.

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.

*

entrada:
salida : ListaOrdenadaCiudades
while $D_{Pos} = D_{Grow-Pos} \cup D_{Prune-Pos} \neq \emptyset$ **do**
    ; // Construir una nueva regla
    Dividir D en $(D_{Grow-Pos} \cup D_{Grow-Neg}) \cup (D_{Prune-Pos} \cup D_{Prune-Neg})$
    Rule $:=$ GrowRule$(D_{Grow-Pos} \cup D_{Grow-Neg})$
    Rule $:=$ PruneRule$(D_{Prune-Pos} \cup D_{Prune-Neg})$
    if *la tasa de error de Rule en* $(D_{Prune-Pos} \cup D_{Prune-Neg}) > 50\%$ **then**
        return RuleSet
    else
        Añadir Rule a RuleSet
        Borrar ejemplos cubiertos por Rule de D
    end
end
return RuleSet

Algorithm 1: Búsqueda Local

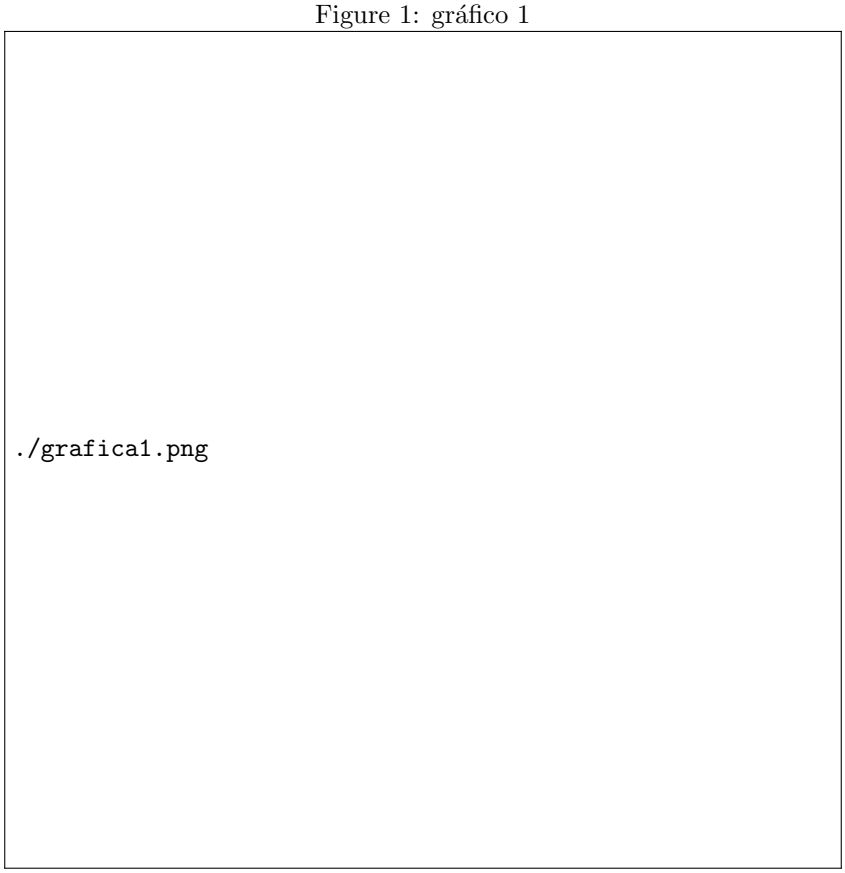
entrada: Secuencia$_1$, ..., secuencia$_n$.
salida : RuleSet
while $D_{Pos} = D_{Grow-Pos} \cup D_{Prune-Pos} \neq \emptyset$ **do**
    ; // Construir una nueva regla
    Dividir D en $(D_{Grow-Pos} \cup D_{Grow-Neg}) \cup (D_{Prune-Pos} \cup D_{Prune-Neg})$
    Rule $:=$ GrowRule$(D_{Grow-Pos} \cup D_{Grow-Neg})$
    Rule $:=$ PruneRule$(D_{Prune-Pos} \cup D_{Prune-Neg})$
    if *la tasa de error de Rule en* $(D_{Prune-Pos} \cup D_{Prune-Neg}) > 50\%$ **then**
        return RuleSet
    else
        Añadir Rule a RuleSet
        Borrar ejemplos cubiertos por Rule de D
    end
end
return RuleSet

Algorithm 2: Busqueda Local

3 Experimentación y Resultados:

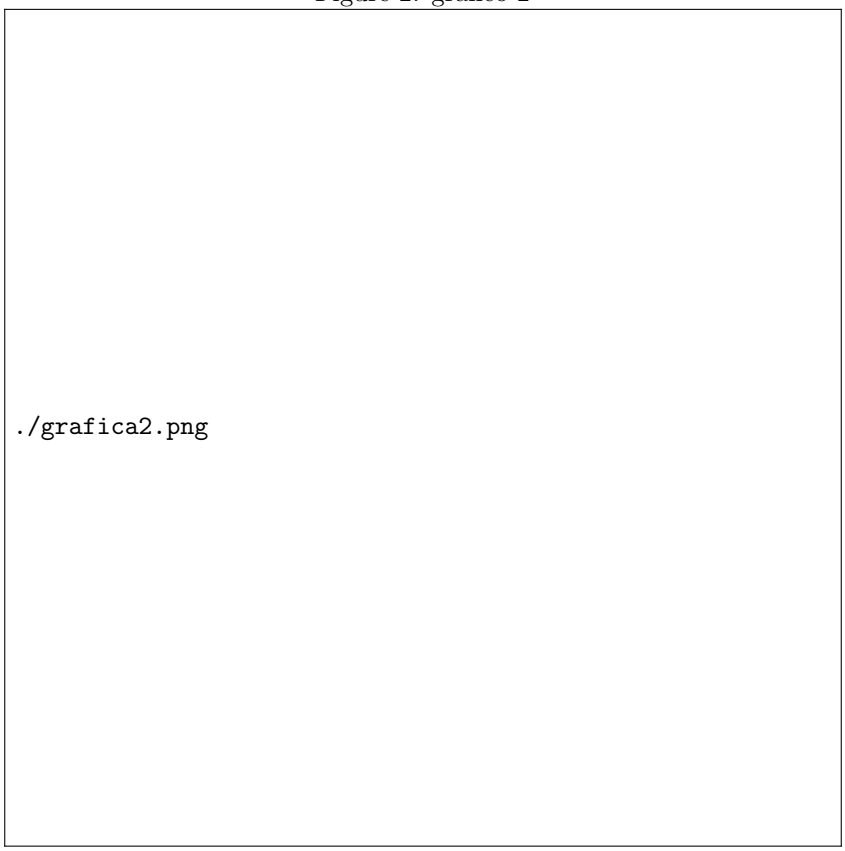
Hemos ejecutado el programa y nos han salido estos resultados:

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:

El algoritmo K-Means tiene una gran limitación, y es que es muy dependiente del conjunto de datos. Es un algoritmo sencillo de implementar pero pese a su simplicidad es bastante eficiente. Los algoritmos de evaluación interna SSE(p) y Silhouette son bastante completos, aunque a nuestro parecer el Silhouette es el mejor de los dos.

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/$Curso_AG_08_Clase_5.pdf$