Diseño y análisis de algoritmos:

Algoritmos constructivos y búsquedas por entornos.

Ángel Alberto Hamilton López
alu0100888102
alu0100888102@ull.edu.es

Introducción.

En esta práctica se pide que implementa distintos algoritmos para resolver el Max-Mean Dispersion Problem. Los algoritmos a implementar son los siguientes:

- Algoritmo voraz.
- Una variante del algoritmo voraz desarrollada por el alumno.
- Algoritmo GRASP.
- Algoritmo Multiarranque.
- Algoritmo de búsqueda por entornos.

A demás de desarrollar e implementar estos algoritmos, también se pide hacer un estudio del comportamiento de estos con los distintos casos de ejemplo que se proporcionan. Las tablas de resultados obtenidos durante el análisis se encuentran en archivos adjuntos, sin embargo las conclusiones de estas se encuentran en este informe.

Estructura del programa.

Para implementar los algoritmos se utiliza una clase Algoritmos, la cual contiene métodos para cada algoritmo y es la que realiza las operaciones. Esta clase contiene un array con los nodos que se han añadido a la solución, un Grafo problema y un Grafo solución.

La clase Grafo nos sirve para representar un grafo completo no lineal. Esto lo hace a través de un doble array que forma una matriz con un tamaño que se se pasa como parámetro. Cada fila de la matriz representa un nodo y cada columna la distancia para llegar hasta ese nodo. Así, si se quiere obtener la distancia desde el nodo X al nodo Y, habría que conseguir en la matriz el valor M[X,Y]. En este caso, al ser un grafo completo y no dirigido, la matriz es simétrica por lo que M[X,Y] y M[Y,X] tienen el mismo valor.

Finalmente, la clase Main es la que recibe la entrada e imprime los resultados por pantalla.

Algoritmo voraz proporcionado en el enunciado.

El pseudocódigo del algoritmo proporcionado es el siguiente:

1. Busca la arista de mayor valor y añade los dos nodos que une a la solución.
2. Por cada nodo que no esté en la solucion:
    1. Combrobar si añadirlo a la solución aumentaría el valor de esta.
    2. Si es así lo añadimos.
    3. Repetir.
3. Si ninguno de los nodos aumenta el valor del grafo, concluye el algoritmo.

Este algoritmo tan simple consigue una solución bastante buena, aunque al tener un componente aleatorio no existe posibilidad de conseguir otra solución mejor sin cambiar la implementación. Observando el tiempo de ejecución, comprobamos que el algoritmo tiene complejidad exponencial, lo que no lo hace el más apropiado para conjuntos muy grandes.

Algoritmo voraz desarrollado por el alumno.

El algoritmo implementado es idéntico al proporcionado en el enunciado salvo por el conjunto de solución inicial. En lugar de encontrar la arista de mayor valor y añadir sus nodos, se busca el nodo cuya suma total de sus aristas sea el más alto y se añade a la solución como conjunto inicial.

Tiene el mismo comportamiento, ventajas e inconvenientes que el otro algoritmo voraz ya explicado. Sin embargo, parece que la solución que proporciona tiene mayor o igual valor que la otra variante, haciendo que esta versión sea mejor. Sin embargo, esto parece ser una mera casualidad.

Algoritmo GRASP.

Los algoritmos GRASP se basan en encontrar una solución factible aleatoria y luego modificarla hasta que no haya una mejora posible. A estas dos etapas se les llama “fase de construcción” y “fase de búsqueda local” El pseudocódigo para el algoritmo GRASP implementado en este caso es el siguiente:

Fase de construcción:

1. Elegir un nodo al azar y añadirlo a la solución.
2. Repetir:
    1. Buscar todos los nodos que aumentaría en valor de la solución.
    2. Elegir uno al azar y añadirlo a la solución.
    3. Repetir hasta que no haya nodos que mejoren el valor de la solución.

Fase de búsqueda local:

1. Repetir:
    1. Repetir para cada nodo que no esté en la solución:
        1. Crear un grafo temporal igual a la solución.
        2. Buscar todos los nodos dentro del temporal que tengan enlaces negativos con este.
        3. Añadir el nodo al temporal.
    2. Elegimos el mayor entre todos los grafos temporales.
    3. Si este grafo es mejor que la solución actual, lo convertimos en la solución actual
2. Repetir hasta que el mayor grafo temporal no mejore la solución.

Este es capaz de generar soluciones aleatorias por lo que puede llegar a encontrar la solución optima a un problema y no se limita a una local. Este algoritmo en concreto tiene una complejidad exponencial, por lo que tampoco es indicado para grupos muy grandes. Sin embargo, parece (a juzgar por los tiempos de las pruebas) que no se incrementa tanto como los voraces simples.

Algoritmo multiarranque.

El algoritmo multiarranque tiene una premisa simple: Generar muchas soluciones con un algoritmo que tenga un factor de aleatoriedad y escoger la mejor solución. En este caso se utiliza el algoritmo GRASP ya explicado para generar las soluciones.

1. Repetir N veces:
    1. Obtener una solución con el algoritmo GRASP.
    2. Si esta solución es mejor que la máxima, establecerla como máxima.
2. Devolver la solución máxima.

Este algoritmo, al emplear GRASP para su desarrollo, tiene el problema de ser demasiado lento en grupos muy grandes. Esto se podría arreglar utilizando otro algoritmo más eficiente. Lo que distingue a este algoritmo es que consigue una solución óptima o muy cercana a esta. Esto se ve claramente en los resultados de los análisis. Con N= 100, llegamos al que podemos suponer es el resultado óptimo en todos los tests, con algunas excepciones en los ejemplos más grandes, donde hay mayor variedad de posibles soluciones. Esto, hace de este algoritmo el mejor cuando se necesita la mejor solución posible y el tiempo y la eficiencia no son una gran preocupación.

Algoritmo de búsqueda por entorno variable.

Esta familia de algoritmo parten de una solución inicial y luego se desplazan por las proximidades de esta solución buscando una mejor. En este caso se utiliza una primera solución aleatoria y las “proximidades” de esta solución, son otros grafos iguales a los que se les ha añadido o sustraído un nodo.

1. Genera un conjunto de nodos aleatorios que será la solución inicial.
2. Repetir:
    1. Por cada nodo en la solución, generar un grafo temporal donde este es eliminado.
    2. Por cada nodo no en la solución, generar un grafo temporal donde este está inlcluido.
    3. Elegir el máximo grafo temporal.
3. Si el grafo temporal máximo es mejor que la solución actual, convertirlo en la solución y repetir. En otro caso terminamos.

Este algoritmo consigue una gama más amplia de soluciones locales que otros algoritmos aleatorios, pero su complejidad también es exponencial. Teniendo uno de los tiempos más altos de los algoritmos se llega a la conclusión de que, por lo menos esta variante, no es una buena solución para este problema.