Explicación de los algoritmos

Christofides

1. Obtener el árbol recubridor mínimo T de G.
2. Sea O el conjunto de vértices de grado impar en T, hallar un apareamiento perfecto M de mínimo peso en el grafo completo sobre los vértices de O.
3. Combinar las aristas de M y T para crear el multigrafo H.
4. Obtener un ciclo euleriano en H (H se considera "euleriano" si es conexo y solo presenta vértices de grado par).
5. Obtener un ciclo hamiltoniano a partir del ciclo euleriano anterior, descartando los nodos visitados (*shortcutting*).

1. Obtener el árbol recubridor mínimo T de G. (Kruskal)

Funciona de la siguiente manera:

- Se crea un bosque B (un conjunto de árboles), donde cada vértice del grafo es un árbol separado
- Se crea un conjunto C que contenga a todas las aristas del grafo
- Mientras C es no vacío
    - eliminar una arista de peso mínimo de C
    - si esa arista conecta dos árboles diferentes se añade al bosque, combinando los dos árboles en un solo árbol
    - en caso contrario, se desecha la arista
- Al acabar el algoritmo, el bosque tiene un solo componente, el cual forma un árbol de expansión mínimo del grafo.

2. Hallar un apareamiento perfecto M de mínimo peso en el grafo completo.

1. Obtenemos el Grafo O que contiene los vértices de grado impar de M y sus aristas de G que tengan como extremo vértices de grado impar.
2. Aplicar backtracking, una “Etapa” es una llamada recursiva, los candidatos son las ramificaciones disponibles:
    a. Comprobamos en la Etapa si es una posible solución.
        i. Comprobamos si es mejor a la que teníamos, si es así la guardamos.
    b. Creamos una nueva etapa y obtenemos los candidatos de la nueva Etapa.
    c. Seleccionamos un candidato para la siguiente Etapa.
    d. Comprobamos si el candidato seleccionado es un camino óptimo a recorrer.
        i. si es óptimo vamos a la siguiente etapa.
        ii. Si no es óptimo seleccionamos el siguiente candidato de la Etapa.

3.Combinar las aristas de M y T para crear el multigrafo H.

1. Creamos un multigrafo con los vertices y aristas de M
2. Insertamos las aristas T en H.

4. Obtener un ciclo euleriano (*Hierholzer*)

1. Creamos un ciclo C(Lista de Aristas)
   - 1.1. Seleccionamos el vértice(nodo) de origen.
   - 1.2. Se ejecuta un DFS hasta encontrarnos de nuevo con el vértice origen.
2. Mientras C no contenga todos los vértices de G (El grafo pasado por parámetro).
   - 2.1. Seleccionamos una arista del ciclo y averiguamos que vértice es su vértice vecino.
   - 2.2. Eliminamos las aristas y lo vértices ya visitados en dicho ciclo, para evitar pasar por ellos nuevamente.
   - 2.3. Ahora creamos un nuevo ciclo C2 que contenga los vértices de C y el seleccionado en el paso 2.1.
   - 2.4. Ahora C, pasa a ser la combinación de C2 y C2.
   - 2.5. Volvemos al paso 2.

5. Obtener un ciclo hamiltoniano (*shortcutting*)

1. A partir de la lista de Aristas obtenida por parámetro generamos una lista de vértices en el orden que hemos ido visitando.
2. Copiamos la lista de vértices generada en el paso anterior de modo que vamos sacando los nodos que están repetidos.
3. Generamos una nueva lista de aristas a partir de la lista de vértices del paso anterior obteniendo así el camino Hamiltoniano.
4. Añadimos la última arista para así generar el ciclo Hamiltoniano.

Nearest Neighbour algorithm

1. Empezar en un vértice arbitrario y cogerlo como actual.
2. Encontrar el vértice adyacente al actual cuya arista tenga el peso más bajos de todas las aristas adyacentes al vértice actual.
3. Marcar el vértice encontrado en el paso 2 como actual.
4. Marcar el vértice encontrado en el paso 2 como visitado
5. Si todos los vértices ya han sido visitadas, terminamos.
6. Si no, vamos al paso 2.

2-opt

Pasos a realizar:

1. Cojemos un tour inicial, generado con alguno que nos de una primera solución
2. Para todos los pares de aristas no incidentes en el mismo vértice hacemos:
    - Las invertimos.
    - Si el nuevo tour es mejor que el actual, lo definimos como actual.
3. Si el tour ha sido mejorado vamos al paso 2.
4. Fin

3-opt

Pasos a realizar:

5. Cojemos un tour inicial, generado con alguno que nos de una primera solución
6. Para conjuntos de tres aristas no incidentes en el mismo vértice hacemos:
    - Para todas las permutaciones posibles de nuevos tour con las tres aristas escogidas.
        - Si el nuevo tour es mejor que el actual, lo definimos como actual.
7. Si el tour ha sido mejorado vamos al paso 2.
8. Fin

Branch and Bound

Aplicar backtracking, una “Etapa” es una llamada recursiva, los candidatos son las ramificaciones disponibles:

1. Comprobamos en la Etapa si es una posible solución.
    a. Comprobamos si es mejor a la que teníamos, si es así la guardamos.
2. Creamos una nueva etapa y obtenemos los candidatos de la nueva Etapa.
3. Seleccionamos un candidato para la siguiente Etapa.
4. Comprobamos si el candidato seleccionado es un camino óptimo a recorrer.
    a. si es óptimo vamos a la siguiente etapa.
    b. Si no es óptimo seleccionamos el siguiente candidato de la Etapa.

Tabla de Clases

Clase	Alumno
Tarea	Alejandro Rosas
Relacion	Alex Peregrina
Solucion	Marc Vila
ControladorTareas	Alex Peregrina
ControladorSoluciones	Marc Vila
ControladorDatosTareas	Alejandro Rosas
ControladorDatosSoluciones	Marc Vila
ControladorDatosRelaciones	Alex Peregrina
ControladorTSP	David Gracia
NearestNeighbour	Alejandro Rosas
BranchAndBound	Alex Peregrina
MultiGrafo	Alex Peregrina
Grafo	David Gracia
Arista	David Gracia
Vértice	David Gracia
ParOrdenado	David Gracia
Christofides	Marc Vila
Kruskal	David Gracia
ApareamientoPerfectoMinimo	Alex Peregrina
Hierholzer	Marc Vila
CicloHamiltoniano	Alejandro Rosas
DosOpt	Alejandro Rosas
TresOpt	Alejandro Rosas