Trabajo Práctico Especial

Estructura de Datos y Algoritmos

Hipergrafo camino mínimo

Integrantes: Martinez, Felipe (51224)

Scigliano, Agustín (51277)

Rivas, Matias (51274)

Año: 2013

1. Introduccíon

El trabajo práctico especial consiste en desarrollar una estructura que represente un hipergrafo, con sus nodos e hiperarcos, del cual debía buscarse el camino mínimo, es decir, el de menor peso. De todo los nodos, dos son los que se distinguen, que llamaremos *origen* y *destino*. La representación gráfica de la solución (camino mínimo) se realizó mediante la herramienta GraphViz.

Para la búsqueda de la solución, se utilizaron dos métodos: **Fuerza Bruta** y **Hill-Climbing**. Ambos serán explicados en este informe.

2. Definición

Un hipergrafo H es una tupla <V, E, w> donde V es un conjunto finito de nodos, E es un conjunto de hiperarcos y w es una función que le asigna a cada hiperarco un vector de pesos numéricos, a efectos de este TPE podemos asumir que cada hiperarco tiene un único peso numérico no negativo.

Un hipergrafo no tiene ciclos.

Un hiperarco es un par e = <T(e), h(e)>, donde T(e) ⊂ V es la cola del hiperarco, h(e) ⊂ VT(e), es llamado la cabeza del hiperarco.

Cada nodo representa un "asset". Un "asset" puede representar cualquier tarea que sea necesaria; es conocimiento que ya se obtuvo, relativo a un objeto real o propiedad de la realidad que se está modelando.

3. Implementación

Para este trabajo de desarrollaron dos métodos para determinar el camino mínimo del hipergrafo. Ambos son distintos entre sí, pero emplean la misma estructura de datos que hemos definido para el hipergrado.

El primer método utiliza lo que se llama **Fuerza Bruta**. No es eficiente en cuanto al tiempo (puede tardar varios minutos u horas si es necesario), pero es capaz de encontrar el camino que se desea. En caso que el hipergrafo sea muy grande, este método no cumple con el objetivo dado que en medio del proceso puede agotarse la RAM asignada al programa y abortar la búsqueda de la solución.

Es necesario destacar que este método se basó en la idea de invertir todo el hipergrafo y comenzar a buscar un camino desde el nodo destino, con la ventaja de saber cuáles son los nodos que necesitan ser visitados para poder acceder a un hiperarco. Esto nos ha permitido reducir la dificultad de encontrar el camino mínimo.

El Segundo método implementado hace uso de **Hill-Climbing**. La primera versión de este método se logró en un tiempo menor en comparación al primero por lo que hemos podido realizar ciertas mejoras en su lógica y así obtener resultados más precisos.

4. Seudocódigo

En esta sección se muestran los seudocódigos de los métodos de **Fuerza Bruta** y **Hill-Climbing Modificado**, junto a métodos complementario que permiten su uso.

Hill Climbing Modificado

```
While (tiempo no se acabó) {
        Random = Agarro una solución al azar*
        If (bestSolution es peor que Random) {
                Do {
                        Para cada uno de los vecinos de Random*{
                                If (vecino mejor que Random) {
                                        Random = vecino;
                                        Se encontró mejor solución = True;
                                }
                        }
                }while (se encontró mejor solución)
        }
        bestSolution = Random;
}
```

*Solución al Azar:

Arranca desde el nodo "Final" del hipergrafo (Se agrega a la cola de nodos usados).

```
While (primer nodo de la cola es distinto al nodo "Inicial") {
        Se elige un hiperarco entrante del primer nodo de la cola al azar.
        Se agrega el hiperarco a la solución.
        Se encola todos los nodos entrantes del hiperarco elegido.
        Se remueve el primer nodo de la cola.
}
```

*Vecinos:

Un vecino de una solución sería remover un hiperarco al azar de la solución y verificar si sigue siendo una solución válida. Si se encuentra una solución válida, se busca vecinos de esa nueva solución encontrada y así iterativamente hasta encontrar la mejor solución posible por esa rama del hipergrafo.

Fuerza Bruta

El orden de complejidad de este algoritmo seria de N * M (siendo N la cantidad de nodos y M la cantidad de hiperarcos). Esto es asi, ya que para cada hiperarco estaríamos visitando todos sus nodos. A pesar de que en la realidad, esto cambiaria debido a la poda que utilizamos y que se daría en la mayoría d los casos.

```
Add (Nodo ORIGEN)

exactSolution (Solution sol, Node end, List<Node> nodes, Solution bestSolution)

If (primer elemento de la lista es Nodo DESTINO) {
        If (no hay mejor solución) {
                bestSolution = sol ;
        }
        Else If (comparo si la solución que me llego es mejor que la que tenía) {
                bestSolution= sol ;
        }
 return;
```

}

aux = tomo el primer Nodo de la lista;

forEACH (hiperarco que le llega al Nodo aux que estamos analizando){

solution =creo posible solución con hiperarcos que forman parte de ella hasta el momento;

auxNodes= creo lista auxiliar para manejar los distintos nodos padres de cada hiperarco;

auxNodes.addAll (nodes);

auxNodes.remove (aux);

forEACH (nodo padre del hiperarco que estamos analizando){

auxNodes.add (nodo);

}

Solution.addToSolution (hiperarco actual);

(Aplico una poda para ver si la posible solución que estoy armando ya es mayor a la mejor que tengo hasta el momento)

exactSolution (solution, end, auxNodes, bestSolution);

5. Comparación de Tiempos

Usando el algoritmo exacto sacamos los siguientes tiempos:

Archivo	A.hg	B.hg	C.hg	Enunciado.hg
Tiempo	27 s	35 s	43 s	<1 s

Usando el algoritmo aproximado sacamos los siguientes resultados dándole 2 minutos para calcular cada grafo:

Archivo	A.hg	B.hg	C.hg	D.hg	Aproximado1.hg	Aproximado2.hg	Enunciado.hg
Peso Minimo	196	491	398	161	1301	659	7

6. Generación de hipergrafos

La generación de los hipergrafos estuvo a cargo de la clase *Parser*. Mediante el método *readFiles* se lee el archivo que contiene al hipergrafo, se lo lee por línea buscado el nodo origen y destino, en primer lugar, para luego tomar los datos de los hiperarcos que son guardados en una instancia de *Hypergraph* y ,al mismo tiempo, se escribe el archivo .dot del .hg en cuestión.

A los nombres de los nodos se les ha agregado "node_" al inicio dado que la herramienta GraphViz daba warnings por cada nodo. Al final del archivo .dot los nombres eran cambiados por los originales para que en la imagen generada se mostraran correctamente.

Luego, cuando se obtenía una solución exacta o aproximada del camino mínimo, el *parser* volvía a leer el archivo .hg pero teniendo en cuenta que se posee una List<Hyperarc>. Se preguntaba si el hiperarco o nodo pertenecía al camino mínimo. Si pertenecía, se muestra

de color rojo. De lo contrario, negro. Si un hiperarco y un nodo, ambos pertenecientes al camino mínimo, estaban conectados, la arista se la muestra de color rojo. De lo contrario, negro.

Para finalizar, el archivo .hg con el camino mínimo se crea tomando los datos de la lista de hiperacos. Se lee los nodos de entrada y salida necesarios y se escribe la línea con el mismo formato del .hg de entrada.

7. Conclusión

Este trabajo planteo un problema NP, es decir, no polinomial al cual intentamos darle respuesta aplicando heurísticas que permitieran aproximarnos a una solución correcta.

EL armado de la clase *HyperGraph,* que fue lo primero que se diseñó y se le dedico un tiempo considerable, fue importante dado que este no solo fue utilizado por los algoritmos de búsqueda del camino mínimo, sino también por el *Parser*, el cual tomaba los datos de los archivos .hg y también permitía devolver los archivos min.dot y min.hg.