Estructura de Datos y Algoritmos

CAMINO MÍNIMO EN HIPERGRAFOS DIRIGIDOS

7 de junio de 2013

Tomas Lori - Legajo 52558
Santiago Ramírez Ayuso - Legajo 52561
Jorge Gomez - Legajo 52055
Instituto Tecnológico de Buenos Aires

Índice

1. Introduccíon

El objetivo de este informe es describir el Trabajo Práctico especial de la materia, Estructura de Datos y Algoritmos. Como requerimientos, se pide implementar un programa realizado en Java en el cual, dado un HyperGrafo en el formato .hg ingresado por línea de comandos, se encuentre el camino mínimo para llegar desde el origen del mismo hasta el destino. Para ello es necesario la implementación de un algoritmo que encuentre una solución exacta y un algoritmo que encuentre una solución aproximada en un tiempo elegido, utilizando el método de local search. // Una vez encontrado el camino, se deberá generar 3 archivos adicionales: //Un archivo .hg conteniendo solo el camino mínimo.

- Un archivo .dot con el grafo completo.
- Un archivo .dot con el grafo completo y con el camino mínimo marcado.
- Los archivos .dot serán abiertos con la herramienta gráfica GraphViz para poder visualizar los resultados obtenidos.

2. Marco teorico

Un HiperGrafo es un conjunto de nodos e hiperarcos relacionados entre sí donde cada nodo tiene una cabeza y una cola de aristas, y cada arista una cabeza y una cola de nodos. Cada nodo se podría representar como una condición y cada arista como una tarea que contiene un peso. Para poder realizar una tarea, es necesario tener todas las condiciones de la cola habilitadas, y la realización de la misma habilitara todas las condiciones de la cabeza.

3. Estructuras

Elegimos modelar la estructura del Hypergrafo como un conjunto de aristas, uno de nodos, un nodo fuente y un nodo sumidero.
Ademá elegimos en particular modelar los conjuntos de nodos y aristas como mapas que utilizan como clave el nombre del nodo o arista. Esta decisión surgio de la necesidad de poder encontrarlos utilizando su nombre a la hora de parsear el archivo de entrada ya que es la unica manera de asegurar que un nodo referenciado por una arista no fue creado ya al ser referenciado por otra. Consideramos que existiendo la necesidad de crear estas estructuras desecharlas y crear nuevas para utilizar en el resto del programa era innecesario.
Los nodos y las aristas por otro lado decidimos modelarlos como conjuntos de colas y cabezas, teniendo estos una lista para cada categoria, debido a que era necesario acceder a un conjunto o a otro dependiendo del algoritmo que se quisiera utilizar. Además los nodos y aristas poseen como variables su nombre, peso (en caso de las aristas) y tanto marcas como contadores, usando las primeras para determinar pertenencia a un conjunto

y los contadores como forma de llevar registro de cantidad de colas o cabezas que lo alcanzaron durante el proceso de un determinado algoritmo.

4. Algoritmos

Presentamos a continuación los algoritmos utilizados para calcular caminos mínimos de hypergrafos dirigidos.

4.1. Algoritmo para generación de soluciones iniciales

Este primer algoritmo es utilizado para generar una solución inicial en el algoritmo exacto.
Debido a que este algoritmo genera una solución aproximada de manera muy rápida se utiliza en el algoritmo exacto para poder tener un valor inicial que ayude a la poda del arbol de soluciones, lo que reduce significativamente el tiempo necesario para encontrar la solución.

```
queue = {raiz}
while Hay nodos en la cola do
    currentNode ← Siguiente elemento de la cola
    for all Aristas cabeza de currentNode do
        if Se desencolaron todas los nodos cola de la arista then
            predecesores de la arista ← aristas predecesorasde todos los nodos
            suma hasta la arista ← peso de todas las aristas predecesoras
        end if
        for all Nodos cabeza de la arista do
            if peso del nodo > suma hasta la arista + peso de la arista then
                Encolo los nodos
                peso del nodo ← suma hasta la arista + peso de la arista
                predecesores del node ← predecesores de la arista + la arista
            end if
        end for
    end for
end while
```

4.2. Algoritmo exacto

Este algoritmo recorre el hipergrafo comenzando por el destino, verificando todos los caminos posibles hacia el mismo y quedándose con el menor. Una vez encontrado retorna el peso mínimo del camino seleccionado y los conjuntos de nodos e hiperarcos utilizados en el mismo.
Inicialmente utiliza el algoritmo de generación de soluciones iniciales y encuentra el peso de un camino posible hasta el destino. Este peso es utilizado para realizar una poda de caminos, y así recorrer y calcular la menor cantidad de caminos posibles, aumentando considerablemente la eficiencia del algoritmo.

```
solution ← initialSolution()
cola = {raiz}
nodeRecursive()
```

```
nodeRecursive()

currentNode ← primerelementodelacola
if peso del camino actual > peso del camino minimo then return
else if currentNode == raiz then
    if peso del camino actual < peso del camino minimo then
        peso del camino minimo ← peso del camino actual
        camino minimo ← camino actual
    end if
end if
for all Aristas cola de currentNode do
    Agrego la arista al camino actual
    nodeRecursive()
    Remuevo la arista del camino actual
end for
```

4.3. Algoritmo aproximado usando Local Search

En el caso del algoritmo aproximado de tipo local search optamos por generar soluciones vecinas tomando como solución un conjunto de aristas por las cuales se puede recorrer el hypergrafo de un extremo al otro, es decir es un camino, y como sos vecinos a aquellos conjuntos de aristas que pueden generarse removiendo una arista del conjunto actual que a su vez tambien sean solución. La valuacion de las soluciones se hace sumando el peso de todas sus aristas y se considera una mejor solucion a aquella de peso menor.// Elegimos como algoritmo local search Hill Climbing luego de considerar tanto la precisión de este como su velocidad de ejecución. Como solución inicial optamos por elegir el conjunto que agrupa la totalidad de las aristas del grafo de manera tal de no limitar las posibilidades de elección del algoritmo.

```
currentSolution ← startSolution;
while No se agoto el tiempo asignado al algoritmo do
    neighbors ← neighbors(currentSolution)
    nextEval ← −INF
    nextSolution ← NULL
    for all neighbors do
        if value(x) > nextEval then
            nextSolution ← x
            nextEval ← value(x)
        end if
        if nextEval <= value(currentSolution) then return currentSolution
        end if
        SolutionNode ← nextSolution
    end for
```

end while
return currentSolution

4.4. Tablas de comparación

En las siguientes tablas mostramos como a pesar de que la elección de una solución inicial cercana al valor exacto hace que la velocidad del algoritmo exacto sea muy pequea para la mayoria de lo hypergrafos existen algunos para los cuales es inviable conseguir una solución exacta y por ende es muy necesario el algoritmo aproximado.
La siguiente tabla muestra la comparación entre los métos para tiempos de 5, 15 y 30 segundos.

Grafo	Inicial	Exacto	Aproximado 5s	Aproximado 15s	Aproximado 30s
A	211 / 97ms	196 / 380ms	988	197	197
B	518 / 106ms	491 / 20932ms	680	516	516
C	438 / 79ms	390 / 4454ms	420	420	420
D	155 / 89ms	-	788	123	123
Aproximado 1	1170 /149ms	-	5996	5170	1136
Aproximado 2	652 / 145ms	-	5997	2717	716

5. Decisiones

A continuación detallaremos las dificultades encontradas en ambos algoritmos y las desiciones que se tomaron.

5.1. Algoritmo Exacto

Inicialmente se comenzó con la implementación de un algoritmo el cual no contemplaba todos los casos de posibles soluciones. Inmediatamente detectado este problema, se procedió a la implementación de un algoritmo que trabajaba sobre los conjuntos de aristas.
Este algoritmo calculaba todos los subconjuntos posibles del conjunto de aristas, sin guardarlos en memoria, ya que de lo contrario no alcanzaba el heap. Se llego a la conclusión, utilizando una clase que iteraba sobre los subconjuntos de aristas sin guardarlos, de que el algoritmo por si solo tenía una complejidad exponencial a la cantidad de aristas. No demoraba en realizar los calculos para cada subconjunto, si no en recorrer todos. Por eso se llego a la decisión de podarlo, esto reducía el número de iteraciones necesarias de manera polinomica. Sin embargo la complejidad total seguía siendo exponencial y el grafo seguía siendo irresoluble para tiempos razonables.
Otra posible solucion que se encontró fue la utilización de árboles para la realización de las podas, pero el nuevo problema yacía, en el límite de memoria.
La siguiente propuesta para la solución del problema consistió en un algoritmo asemejado al algoritmo de Dijkstra, que evaluava para ver si era solución un conjunto de aristas una vez que este era el de peso mínimo ponderado, en caso de no ser solución, se agregaban al conjunto de soluciones posibles subconjuntos que contenían todas las aristas de este y

una arista extra. Pero nuevamente, la restriccón de nuestro algoritmo era la memoria. La propuesta final que pudo lograr resolver los problemas presentados anteriormente, como ya fue explicado anteriormente, consiste en la utilización de conjuntos de nodos incluidos en el recorrido del grafo.

6. Conclusiones

Luego de la realización del Trabajo se pudo concluir que no es eficiente la creación de estructuras y su guardado en la memoria, si se quiere resolver problemas de gran tamanio como en los ejercicios B y C. Por eso es conveniente la utilización de caminos y recorridos a través del grafo para poder realizar así podas.
Para grafos mucho mas grandes, como lo son D, aproximado1 y aproximado2, es conveniente utilizar una heurística ya que un algoritmo podría horas para resolverlo.