Algoritmia para problemas difíciles

Práctica 1

Informe de resultados

Andrés Gavín Murillo 716358

Darío Ferrer Chueca 721132

7 de enero de 2020

Introducción

El objetivo de la práctica consiste en desarrollar el algoritmo de Karger para realizar una aproximación del algoritmo de corte mínimo en un grafo de n productos.

La práctica se ha desarrollado en C++, donde se explican todos los pasos realizados, y se compila y ejecuta de la siguiente manera (realizado para Hendrix):

make

Generar datos de prueba: **./practica1 <directorioDondeGenerar> <maxJuntos>**
<maxJuntos> = <natural> => Número máximo de veces que dos productos se han podido comprar juntos.
Resolver productos: **./practica1 <ficheroProductos> <ficheroCompradosJuntos> <T>**
<T> = max => Iteraciones para alcanzar óptimo con alta probabilidad.
<T> = <natural> => Iteraciones introducidas por el usuario.

Detalles de implementación

Cada conjunto de productos se almacena en una estructura de datos formada por un vector de productos (Producto.hpp) que guarda la información de cada uno, y una matriz de adyacencia que permite almacenar el grafo. Como la matriz es cuadrada, simétrica y la diagonal está formada por 1s, sólo se almacena la parte triangular superior (como se indica en el código, ConjuntoProductos.hpp).

El problema a resolver se ha identificado como un problema de grafos, el cual se corresponde con encontrar el corte mínimo, que consiste en realizar la partición de un grafo en dos conjuntos disjuntos, siendo el número de aristas entre ambos trozos mínimo.

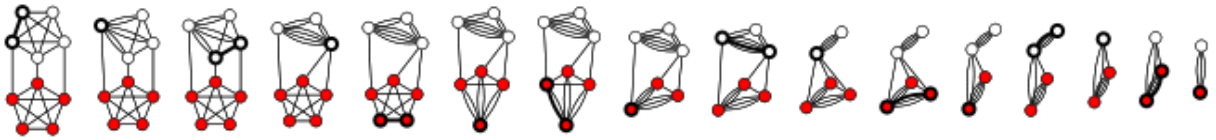
Para realizar la partición se ha optado por la implementación del algoritmo de Karger, un algoritmo aleatorio cuyo pseudocódigo es el siguiente:

```
Let G = (V, E)
While |V| > 2:
        Pick an edge e from E uniformly at random
        Merge or contract e in a single vertex
        Remove self-loops
Return the only cut left with 2 final vertices.
```

Este algoritmo tiene como entrada un grafo con V vértices y E aristas, y se basa en la operación de contracción. Mientras quede un número de vértices mayor que dos, se

selecciona de manera aleatoria una de las aristas del grafo y se hace la contracción de la misma, juntando los dos vértices que une en un único vértice nuevo.

De esta manera, se reduce en cada iteración el número de vértices en 1, hasta que finalmente, se termina cuando sólo quedan dos, que representan los dos subconjuntos de vértices resultantes. Finalmente, las aristas que quedan uniendo ambos vértices representan el corte alcanzado.



IMPLEMENTACIÓN CONTRACCIÓN

Para la implementación del algoritmo se pensaron diferentes alternativas, pero finalmente se utilizó una estructura basada en el número de vértices, ya que al tener una matriz de adyacencia para representar el grafo resulta lo más sencillo. La interpretación de elegir una arista al azar se realizó eligiendo dos nodos diferentes al azar. Así, el nodo generado en la unión tiene la información de los nodos originales que ha contraído. El coste de la contracción es O(n²) como se detalla en el código (main.cpp).

ANÁLISIS DEL ALGORITMO

En primer lugar, se elige un corte mínimo C, con tamaño k. Para producir C, en cada iteración, el algoritmo no debe contraer ninguna de las aristas en C. Por lo tanto, para conseguir ese corte mínimo, se busca la probabilidad de que, tras n-2 iteraciones(n vértices), no se dé ninguna de estas contracciones:

$$P(\mathcal{E}) = P\left(\bigcap_{i=1}^{n-2} \mathcal{E}_i\right)$$

Sea m el número total de aristas en el grafo, la probabilidad de contraer una arista que está en C es **k/m**. Debido a que C es el corte mínimo, el grado de todos los vértices debe ser mayor o igual que C, y la suma de los grados de los vértices debe ser 2*nºaristas. De lo cual se puede obtener **m >= nk/2**. De esta manera, se puede llegar al siguiente razonamiento:

$$P(\bar{\mathcal{E}}_1) = \frac{k}{m} \leq \frac{k}{nk/2} = \frac{2}{n}$$

Así pues, para el conjunto del algoritmo se debe satisfacer la siguiente probabilidad:

$$p_n \geq \left(1 - \frac{2}{n}\right) p_{n-1}$$

Esto se puede reformular como $\binom{n}{2}^{-1}$, lo cual es equivalente a $\frac{2}{n(n-1)}$. Esto es una probabilidad muy baja, por lo cual no podría considerarse un buen algoritmo, sin embargo,

para aumentar las posibilidades, se realiza éste un número de ejecuciones k y se escoge la mejor opción posible. De esta forma, la probabilidad de no encontrar un corte mínimo tras k ejecuciones del algoritmo sería:

$$\left(1-\frac{2}{n(n-1)}\right)^k$$

Si el algoritmo se ejecuta (n (n – 1) / 2) ln(n) veces, la probabilidad de no encontrar el corte mínimo es la siguiente:

$$\left(1-\frac{2}{n(n-1)}\right)^{(\frac{n(n-1)}{2})\ln n} \leq \left(\frac{1}{e}\right)^{\ln n} = \frac{1}{n}$$

Por ejemplo en caso de que el grafo cuente con 1000 nodos, la probabilidad de no encontrar el corte mínimo es 0,001, lo cual prácticamente asegura que el resultado obtenido es el correcto.

El coste en tiempo total del algoritmo es $O(n^2)$ para una ejecución, por lo que para T ejecuciones, el tiempo es: $O(T\ n^2)$.

MATRIZ DE ENTEROS

Para el caso en el que la matriz de datos no solo dé la información de si dos productos han sido comprados juntos, si no el número de veces, se ha adaptado el algoritmo, de manera que, al añadir las aristas entre los grafos, se añaden tantas aristas como veces los productos han sido comprado juntos. De esta manera, se hace que aquellos productos con un número mayor de aristas entre sí sean más difíciles de separar que aquellos que han sido comprados juntos pocas veces.

Pruebas realizadas

Se han realizado diferentes pruebas con distintos ficheros en Hendrix. Finalmente sólo se muestran pruebas con ficheros generados desde nuestra implementación, ya que la mayoría de los ficheros disponibles en la web no cumplen nuestros criterios (algunos ni los especificados en el guión de la práctica), por ejemplo, hay ficheros de matrices cuya diagonal no son 1s, ficheros de matrices donde no hay separación entre los dígitos, ficheros de matrices donde no se especifica el número de productos y ficheros de productos que contienen atributos diferentes a los de nuestra implementación.

Ficheros	Número de iteraciones (T)	Tiempo en realizar el corte	Nodos del conjunto 1	Nodos del conjunto 2	Aristas que separan los conjuntos
f1_10.txt y f2_10.txt	103 (max)	0,004s	1	Resto	2
f1_100.txt y f2_100.txt	22795 (max)	22,43s	46	Resto	32
f1_100.txt y f2_100.txt	50	0,048s	Resto	52, 76, 79	127
f1_100.txt y f2_100.txt	1000	0,967s	Resto	72	37
f1_1000.txt y f2_1000_0.1txt	100	6,051s	Resto	79, 152, 482, 827	1567
f1_1000.txt y f2_1000_0.1txt	1000	1m 1,608s	Resto	945	382
f1_1000.txt y f2_1000_0.5txt	100	6,010s	25, 239, 243, 259, 358, 736, 797	Resto	4207
f1_1000.txt y f2_1000_0.5txt	1000	1m 1,241s	Resto	815	594
f1_1000.txt y f2_1000_0.9txt	100	6,057s	449, 511	Resto	1087
f1_1000.txt y f2_1000_0.9txt	1000	1m 2,382s	Resto	817	526
f1_1000.txt y f2_1000_0.0txt	1000	1m 0,963s	582 nodos	418 nodos	0
f1_1000.txt y f2_1000_1.0txt	100	6,140s	20 nodos	980 nodos	19600
f1_1000.txt y f2_1000_1.0txt	1000	1m 0,088s	999 nodos	1 nodos	999
f1_100.txt y f2_100_0.6.txt	22795 (max)	22,119s	Resto	77	243

En los ficheros de 1000 productos no se ha ejecutado el algoritmo de Krager el número máximo de veces ya que hay que repetirlo 3450423 veces para alcanzar el corte óptimo con alta probabilidad.

El fichero f2_100_0.6.txt contiene una matriz donde los productos han podido ser comprados juntos más de una vez.

Bibliografía

https://web.stanford.edu/class/archive/cs/cs161/cs161.1138/lectures/11/Small11.pdf

https://en.wikipedia.org/wiki/Karger%27s_algorithm