Algoritmo Hungaro

para resolver el problema de minimizar la suma

de los pesos en un grafo bipartito $G = (X \cup Y, E)$, con $n = |X| = |Y|$.

1: Introducción

Recordemos que estamos suponiendo que el grafo viene dado por una matriz de pesos.

La clase pasada probamos que se puede restar (o sumar) una constante a cualquier fila o columna, y, si bien el valor del matching cambia, el matching que minimiza la suma es el mismo en la matriz antes de hacer el cambio que en la siguiente.

Con lo cual, mostramos el siguiente algoritmo: restar el minimo de cada fila y cada columna, para tener un cero en cada fila y cada columna. Luego, buscar un matching de ceros usando el algoritmo de matcheo que ya conocemos. Si se lo encuentra, como todos los valores de la matriz son no negativos, es un matching minimal. Si no, el algoritmo de matcheo debe dar un S y un $\Gamma(S)$ con $|S| > |\Gamma(S)|$. Como vimos en el ejemplo de clase, le sumamos al final uno a $\Gamma(S)$ y le restamos uno a S. Eso es lo que haremos siempre de ahora en mas, aunque no necesariamente uno, sino alguna constante m. Ahora bien, ¿que constante m restamos y sumamos? Como lo que queremos es crear al menos un nuevo cero, y en $S \times \overline{\Gamma(S)}$ no hay ceros, lo logico es tomar m el minimo en esa zona. Es decir, el algoritmo queda:

Parte 1) *Restar Minimo de cada fila.*

Parte 2) *Restar Minimo de cada columna.*

Parte 3) *Construir un matching inicial usando Greedy.*

Parte 4) *Extender el matching de 3) usando el algoritmo de matcheo (de ceros) de "*`escanear filas-etiquetando columnas`*" y "*`escanear columnas-etiqueteando filas`*", hasta encontrar una columna no matcheada y extendiendo el matching en consecuencia. Si en algun momento se traba, y no se llega a una columna no matcheada, el algoritmo nos da un S con $|S| > |\Gamma(S)|$ y entonces tomamos el minimo de $S \times \overline{\Gamma(S)}$, se lo restamos a S y sumamos a $\Gamma(S)$ para obtener ceros nuevos y poder continuar la extension del matching.*

Observemos que restarle m a S y sumarselo a $\Gamma(S)$ hace lo siguiente en estas 4 zonas:

.- En $S \times \Gamma(S)$: resto por S, sumo por $\Gamma(S)$, el resultado queda igual.

.- En $S \times \overline{\Gamma(S)}$: resto por S, no sumo nada, el resultado es restar m.

.- En $\overline{S} \times \Gamma(S)$: no resto nada pero sumo por $\Gamma(S)$: el resultado es sumar m.

.- En $\overline{S} \times \overline{\Gamma(S)}$: ni sumo ni resto nada, el resultado queda igual.

Visualmente:

$$\begin{array}{c} \\ \overline{S} \\ S \end{array} \begin{array}{c} \begin{array}{cc} \Gamma(S) & \overline{\Gamma(S)} \end{array} \\ \begin{pmatrix} +m & 0 \\ 0 & -m \end{pmatrix} \end{array}$$

Cuando lo hacemos "a mano" es mejor hacer eso, para ahorrarnos trabajo, pero cuando lo implementemos no haremos eso.

2: Complejidad

Recordemos primero cúal era la complejidad del algoritmo cuando no habia pesos: la complejidad de cada escanear columna era $O(1)$, pero la de los EscanearFilas era $O(n)$. Si llamamos "paso" a una iteración del algorimo que aumenta el matching en uno, tenemos que habra a lo sumo n pasos. En cada paso, cada fila se escaneaba una sola vez. Por lo tanto, hacemos, en cada paso, a lo sumo n escaneos de filas, asi la complejidad de cada paso es $O(n^2)$ y como hay a lo sumo n pasos, la complejidad del algoritmo completo es de $O(n^3)$. (cuando no hay pesos)

Como el algoritmo húngaro añade al algoritmo básico de busqueda de matchings diversas cosas como busqueda de minimos, cambios de matrices, etc, pareciera que debemos tener mas complejidad.

De ehcho, depende mucho como se lo programe. Una implementación "super-naive" del paso 4) seria:

Buscar el mayor matching.

Cambiar la matriz.

Buscar el mayor matching.

Cambiar la matriz

etc,

hasta extender el matching que se tenia en una fila. Luego, repetir.

La complejidad de buscar el mayor matching es $O(n^3)$, y la complejidad naive de cambiar la matriz es de $O(n^2)$, pues hay que buscar un minimo en $S \times \overline{\Gamma(S)}$, que tiene $O(n^2)$ elementos, y luego sumar y restar ese minimo en diversos lugares, que tambien tiene complejidad $O(n^2)$. Por lo tanto, si llamamos "paso" a una iteración en la cual se aumenta el matching en una fila, una codificación "super-naive" del Hungaro tiene complejidad en cada paso de:

$(O(n^3) + O(n^2)).\#$ (veces que se cambia una matriz en un paso).

Veremos mas abajo que una matriz se cambia a lo sumo n veces en un paso, por lo que tendriamos una complejidad en cada paso de $O(n^3)n$ y como hay a lo sumo n pasos, una complejidad total de $O(n^5)$.

Veamos com reducir esto. Primero lo reduciremos a $O(n^4)$: Para empezar, en el ejemplo que dimos en clase no empezamos a buscar el matching desde cero cuando cambiamos la matriz, sino que continuamos con el matching que teniamos. ¿Es esto posible siempre?

Lema Cuando se cambia la matriz, no se pierde ningun cero del matching

Prueba: Como vimos, cuando se cambia la matriz solo se cambian las regiones $S \times \overline{\Gamma(S)}$ y $\overline{S} \times \Gamma(S)$. Pero en la primera no hay ceros de ningun tipo, y en la segunda, si bien puede haber ceros, no puede haber ceros del matching, porque cuando escaneamos las columnas de $\Gamma(S)$, al encontrar un cero del matching, la fila correspondiente es agregada a S, y por lo tanto, si hubiera un cero del mathcing parcial en $\overline{S} \times \Gamma(S)$, la fila en donde esta deberia estar en S, no en $\overline{S}$.

QED

Lema: *En cada paso, hay a lo sumo n cambios de matrices.*

Prueba: Observemos que luego de un cambio de matriz, se agregan ceros nuevos en $S \times \overline{\Gamma(S)}$. Estos ceros nuevos haran que el $\Gamma(S)$ "cresca", es decir, tendremos nuevas columnas para escanear. Cuando las escaneamos, puede pasar una de dos cosas:

1) *Una nueva columna esta sin matchear*: en ese caso, el matching se extiende y se acaba el paso.

2) *Las nuevas columnas estan matcheadas*: en ese caso, como las nuevas columnas estan en el viejo $\overline{\Gamma(S)}$, es decir, son columnas que no tenian ceros en S, el matching de la columna debe necesariamente ser con una fila de $\overline{S}$. Pero entonces al escanear la columna, se agrega una fila NUEVA al S, es decir, crece el S.

Por lo tanto, vemos que luego de un cambio de matriz, o crece el matching (y por lo tanto termina el paso) o bien crece el S. Como el S solo puede crecer a lo sumo n veces, tenemos que puede haber a lo sumo n cambios de matrices en cada paso.

QED

Ahora esta claro tambien que la complejidad de extender el matching no es $O(n^3)$ en cada paso, sino solo $O(n^2)$, pues el $O(n^3)$ es si empezamos con un matching desde cero, pero como empezamos con un matching ya construido, la complejidad de extenderlo en uno es $O(n^2)$, si lo hacemos, se acaba el paso, si no lo logramos, se cambia la matriz, por lo tanto, la complejidad de una implementación meramente "naive" es:

$$n.(O(n^2) + O(n^2)), n = O(n^4)$$

(el primer n es el número de pasos, el último n es el numero de veces que cambio la matriz, el primer $O(n^2)$ es el tiempo en extender el matching en 1 o bien construir el S, y el segundo $O(n^2)$ es el cambio de matriz).

Sin embargo, podemos hacer incluso mejor:

Teorema: *La complejidad del Algoritmo Húngaro es* $O(n^3)$.

Prueba:

Primero, las partes 1),2) y 3) se hacen una sola vez y tienen complejidad $O(n^2)$ cada una. Como la complejidad de la parte 4) es mayor que esto, podemos ignorarlas.

La complejidad del paso 4) es como la analizamos arriba:

$$O(n.n.(\text{Compl(Ext.Match en 1 ohallar } S) + \text{Compl(cambiarMatriz)}$$

(el primer n es el número de pasos, el segundo el número de cambios de matriz en un paso, donde "paso" como siempre es extende el matching en uno).

En la implementación Naive, las dos complejidades Compl(Ext.Match en 1 ohallar S) y Compl(cambiarMatriz son $O(n^2)$. Veamos como se pueden reducir a $O(n)$ cada una.

La complejidad de Compl(Ext.Match en 1 ohallar S) es $O(n^2)$ porque debemos re-escanear todas las filas y columnas. Pero, primero, recordemos que no hara falta escanear las columnas: basta con leer el registro donde se indica cual es el matching de esa columna. En cuantro a las filas,

lo que haremos es lo siguiente: el re-escaneo de las filas de S lo haremos MIENTRAS cambiamos la matriz. Por lo tanto, en realidad ahora solo debemos empezar a escaner NUEVAS filas que se agrewgan al S luego de cambiar la matriz.

Esto producirá que, en cada paso, una fila solo sera escaneada a lo sumo una vez por EscanearFilas, independientemente de cuantos cambios de matrices haya.

Asi, tendremos un total de a lo sumo n EscanearFilas en cada paso, y la complejidad total sera entonces $O(n^2)$ en cada paso (comparada con $O(n^2)$ en cada cambio de matriz).

Asi, la complejidad en realidad quedaria:

$$O(n.(O(n^2) + n.\text{Compl(cambiarMatriz)}$$

Veremos que se puede reducir la complejidad del cambio de matriz a $O(n)$.

Entonces, en cada paso tendremos a lo sumo n EscanearFilas c/u con $O(n)$, y a lo sumo n cambios de matrices, c/u $O(n)$, por lo tanto la complejidad de un paso será de a lo sumo $nO(n) + nO(n) = O(n^2)$ y por lo tanto la del algoritmo sera $O(n^3)$.

Veamos entonces como probar que la complejidad de cambiar la matriz es $O(n)$:

En vez de cambiar toda la matriz, llevaremos un contador en cada fila y en cada columna de cuanto restarle o sumarle a esa fila o columna: en vez de cambiar la matriz cambiamos estos contadores, los cuales son $2n$ por lo tanto la complejidad es $O(n)$.

Para concretar, como restamos de las filas y sumamos a las columnas, tendremos un contador RF digamos que nos dice cuanto restarle a la fila i por ejemplo, y un contador SC que nos dira cuanto sumarle a la columna j.

El problema que nos quedaria seria que como no cambiamos realmente la matriz, ¿como buscamos ceros ahora? Lo que hacemos es que, en vez de buscar matchings de ceros en la matriz, lo que hacemos es buscar matchings de entradas que den cero cuando se les suma SC y se les resta RF. Es decir, en vez de testear si $A_{i,j} = 0$, testeamos si $A_{i,j} - RF(i) + SC(j) = 0$. Para hacerlo a mano seria muy tedioso, y es mas facil visualmente chequear por ceros que andar haciendo calculos. Pero para la computadora ambas son operaciones $O(1)$.

Ademas, esta el problema de calcular el minimo. Para no perder tiempo $O(n^2)$, se usa la tecnica del algoritmo de Dijkstra de ir guardando minimos parciales en registros: en nuestro caso, guardaremos el minimo de la interseccion de cada columna con el S parcial en un registro, digamos $m(j)$, lo cual podremos actualizar en tiempo $O(1)$, y cuando haya que calcular el minimo de $S \times \overline{\Gamma(S)}$, simplemente sera necesario calcular el minimo de n registros, por lo que sera $O(n)$. Ademas, el re-escaneo de las filas que debemos hacer ahora en el cambio de matriz, en realidad no lo hacemos, porque lo unico que nos interesa es saber donde estan los nuevos ceros en $S \times \overline{\Gamma(S)}$, pero eso es mas facil hacerlo, no ree-scaneando las filas de S, sino mirando las columnas de $\overline{\Gamma(S)}$: observar que los nuevos ceros solo estaran en aquellos lugares donde este el minimo m de $S \times \overline{\Gamma(S)}$. Pero el minimo de cada columna en su intersección con S ya lo venimos guardando en $m(j)$, asi que solo hay que chequear cuales columnas tienen $m(y) = m$: esas columnas tendran los nuevos

ceros, y para etiquetearlas solo hay que tener la precaución, cuando calculamos $m(j)$, de guardar en otro registro DONDE se produce el minimo.

Un ultimo detalle: ¿como sabemos cuales columnas estan en $\Gamma(S)$? Cone l regiustro $m(j)$, ahora es facil: $\Gamma(S)$ son precisamente aquellas columnas que tienen $m(j) = 0$. Asi que podemos revisar todas las columnas: aquellas con $m(j) = 0$, le sumamos m al contador $SC(j)$, aquellas con $m(j) > 0$, quequeamos si es iguala m, y ademas actualizamos el minimo.

En conclusión, se puede hacer todo en tiempo $O(n)$.

QED