

Matchings Pesados

Daniel Penazzi

16 de mayo de 2021

Tabla de Contenidos

1 Matchings pesados

- Grafos bipartitos con pesos en los lados
- Algoritmo de minimizar el máximo (de Gross)

2 Complejidades

- Complejidad del algoritmo para encontrar matching maximal
- Complejidad del algoritmo de minimizar máximo

3 Ejemplo de minimizar el máximo

Problemas de optimización

- El algoritmo para encontrar un matching maximal permite una óptima utilización de recursos....cuando todas las condiciones son iguales.
- Pero podría ser que ese no fuese el caso.
- Por ejemplo los trabajadores podrían ser capaz de hacer tanto la tarea A como la B, pero con distintos costos pej por motivos de transporte u otras causas.
- Entonces no se quiere solamente cubrir todas las tareas pej, sino hacerlo con el menor costo.
- La situación entonces será que tendremos un grafo bipartito con pesos en los lados.

Grafos con pesos

- Para evitar complejidades innecesarias, supondremos que las dos partes del grafo tienen la misma cantidad de vértices, y que existe al menos un matching perfecto en el grafo.
- Si se interpretan los pesos como costos, el problema a resolver es, de entre todos los matchings perfectos que existan, elegir aquel que tenga el “menor” costo.
- Similarmente, los pesos pueden representar ganancias en vez de costos, y en ese caso se quiere elegir aquel matching perfecto que maximize la ganancia.
- Un problema es dual del otro (simplemente multiplicando los numeros por -1 y sumandole un número lo suficientemente grande pasamos de uno a otro) así que sólo analizaremos el problema de minimizar costos.

Significado de minimizar

- Queda definir qué significa “menor costo”.
- Tomemos como ejemplo el costo igual a la cantidad de tiempo que un trabajador tarda en terminar una tarea.
- Supongamos que las tareas A y B pueden ser realizados tanto por α como por β .
- Supongamos que α puede terminar A en siete horas y B en ocho horas, mientras que β puede terminar A en cinco horas y B en siete horas.
- ¿Cual asignamiento es “mejor” en términos del costo en tiempo?

Problemas de optimización

- Asignar $\alpha A, \beta B$ hace que el costo total en tiempo sea 14 horas, mientras que asignar $\alpha B, \beta A$ hace que el costo total sea 13 horas y conviene la segunda opción.
- Ahora bien, esto tiene sentido si los trabajos deben realizarse uno a continuación de otro (pej, que A deba estar terminado antes de poder empezar B).
- Pero si los trabajos pueden hacerse en paralelo, entonces la suma de los tiempos no tiene sentido, sino el máximo tiempo.
- En ese caso, $\alpha A, \beta B$ permite completar todo en $\max\{7, 7\} = 7$ horas mientras que $\alpha B, \beta A$ lo hace en $\max\{8, 5\} = 8$ horas y en este caso conviene el primer asignamiento.

Problemas de optimización

- Otro ejemplo donde se ve la diferencia de criterio es el siguiente:
- Supongamos que asignar tareas a trabajadores tiene un costo en dinero.
- Si los costos los absorbe la empresa que contrata a los trabajadores, a la empresa le interesa minimizar el costo total, es decir, la suma de todos los costos.
- Por otro lado, si los costos los absorbe cada trabajador individualmente, entonces lo que se quiere es que el trabajador que mas deba pagar, pague lo menos posible.
- En este caso, lo que se quiere es minimizar al mayor costo.

Problemas de optimización

- Así que tenemos al menos dos criterios para “minimizar costos” (hay varios mas, pero veremos principalmente estos dos):
 - 1 Minimizar la **suma** de los costos.
 - 2 Minimizar el **mayor** costo.
- En ambos casos, dado que un matching perfecto es en este caso una biyección entre X e Y , podríamos simplemente hacer una lista de todos ellos, y quedarnos con el mejor.
- Pero esto tiene complejidad $O(n!)$ (!)
- Los algoritmos para resolver estos dos problemas en tiempo **polinomial** son completamente distintos.
- Veremos primero el segundo problema porque el algoritmo es mucho mas fácil, pero el problema que **realmente** nos interesa es el primero.

Minimizar máximo

- Saber si existe o no uno es muy fácil: simplemente eliminamos todos los lados con peso mayor o igual que ω y corremos el algoritmo que busca matching maximal en el grafo que queda.
- Si el matching maximal que encuentra es perfecto, tenemos un mejor matching, si no, no.
- Si tenemos un mejor matching, seguimos buscando, eliminando cada vez mas lados.
- En algún momento tendremos que para algún μ , existe un matching perfecto con mayor peso μ pero no existe un matching perfecto con mayor peso $\mu - 1$.

Minimizar máximo

- Hay un par de detalles para mejorar la eficiencia.
- Primero, no conviene ir bajando “de a uno” el umbral a partir del cual eliminamos lado.
- Porque esto daría que en el peor de los casos tendríamos que hacer $O(n^2)$ búsquedas de matchings maximales.
- Lo que conviene es hacer **busqueda binaria**.
- Es decir, seteamos un umbral μ a un número que esté a la mitad de todos los $O(n^2)$ pesos.

Minimizar máximo

- Testeamos en el grafo que tiene solamente los lados de pesos menor o igual que μ a ver si tiene un matching perfecto. Luego:
 - 1 Si no tiene un matching perfecto, hay que tomar un umbral mas grande. Lo lógico es tomar el nuevo umbral a la mitad entre el μ que acabamos de testear y la cota superior.
 - 2 Si tiene un matching perfecto, hay que tomar un umbral mas chico. Lo lógico seria tomar el nuevo umbral a la mitad entre el μ que acabamos de testear y la cota inferior.
 - Excepto que podemos mejorar un poco esto. Lo explicamos en la pagina siguiente.
- Repetimos 1 y 2 hasta llegar a un μ para el cual exista matching perfecto pero tal que no exista matching perfecto para $\mu - 1$.

Minimizar máximo

- Como mejorar un poco el paso 2):
- Cuando borramos todos los lados con peso mayor que μ , nos aseguramos que si encontramos un matching perfecto este tendrá mayor peso menor o igual que μ .
- Pero podría suceder que el peso fuese efectivamente **menor** que μ .
- Entonces en vez de tomar el nuevo umbral a la mitad entre μ y la cota inferior, lo hacemos a la mitad entre el mayor peso que encontramos y la cota inferior.
- Esta mejora no afecta la complejidad de peor caso, pero en la práctica puede hacer un speed-up del algoritmo en algunos casos.

Complejidad de encontrar matching maximal

- Empezaremos con el algoritmo básico de encontrar un matching maximal.
- Sabemos que Edmonds-Karp es $O(nm^2)$ y Dinic es $O(n^2m)$
- Pero eso es en networks generales. En el caso particular de networks asociados a grafos bipartitos los algoritmos tienen mejor complejidad
- Para empezar todas las capacidades son iguales a uno.

Complejidad de encontrar matching maximal

- Recordemos brevemente como probabamos que Edmonds-Karp tenia complejidad $O(nm^2)$:
 - La complejidad de encontrar UN camino aumentante (y por lo tanto poder aumentar el flujo) era $O(m)$.
 - El $O(nm)$ restante viene de que probamos que podia haber a lo sumo esa cantidad de caminos aumentantes.
- Pero si todas las capacidades son iguales a uno, entonces cada vez que aumentamos el flujo lo hacemos en una unidad.
- Por lo tanto la cantidad de caminos aumentantes esta acotada superiormente por la cantidad de vecinos de s , en particular, por n .
- Asi que la complejidad de Edmonds-Karp en este tipo de networks es $O(mn)$.

Complejidad de encontrar matching maximal

- Para Dinic, se puede probar que si la capacidad de todos los lados es 1, entonces la complejidad de correr Dinic es $O(mn^{\frac{2}{3}})$, lo cual es un poco mejor.
- Pero, el network correspondiente al grafo bipartito tiene una propiedad adicional:
 - Todo vértice distinto de s, t tiene grado de entrada uno o grado de salida 1.
 - (Pues $\Gamma^-(x) = \{s\} \forall x \in X$ y $\Gamma^+(y) = \{t\} \forall y \in Y$).
- Se puede probar que en networks con capacidades todas uno y con esa propiedad adicional, Dinic corre en tiempo $O(m\sqrt{n})$.

Complejidad de Minimizar máximo

- ¿Cual es la complejidad de este algoritmo?
- El peor caso posible seria que hubiera lados entre todos los vértices de las dos partes (n^2 lados) y que todos los pesos de esos lados fuesen distintos.
- En ese caso, la búsqueda binaria involucraria $\lg(n^2) = 2\lg(n) = O(\lg(n))$ llamadas al algoritmo que tiene que buscar matching maximal.
- Ese algoritmo vimos que tiene complejidad $O(m\sqrt{n}) = O(n^{\frac{5}{2}})$ con nuestra suposición de que $m = n^2$ si se programa con Dinic, y complejidad $O(mn) = O(n^3)$ si se lo hace con Edmonds-Karp.
- Asi que la complejidad total, digamos con Dinic, seria $O(n^{\frac{5}{2}}\lg(n))$excepto que hay algo que todavia no hemos calculado

Lo que falta

- Ese algo es que, para poder hacer búsqueda binaria mediante umbrales, tenemos que ordenar los n^2 pesos primero.
- Usando un orden por comparación óptimo, esto lleva $O(\# \text{términos para ordenar} \cdot \lg \# \text{términos para ordenar}) = O(n^2 \lg(n^2)) = O(n^2 \lg(n))$.
- De todos modos esto es menor que $O(n^{\frac{5}{2}} \lg(n))$ así que es “absorbido” por esa complejidad.

Ejemplo

- Supongamos que la matriz de la página siguiente es una matriz de tiempos para que trabajadores A, B, \dots realicen los trabajos i, ii, \dots , que se van a hacer todos los trabajos en paralelo y queremos minimizar el tiempo para que todos los trabajos esten terminados.
- Queremos entonces minimizar el máximo costo (es decir, el máximo tiempo, en este caso).

Ejemplo

	<i>i</i>	<i>ii</i>	<i>iii</i>	<i>iv</i>	<i>v</i>	<i>vi</i>	<i>vii</i>	<i>viii</i>
<i>A</i>	5	9	9	8	7	10	4	15
<i>B</i>	8	10	9	10	6	7	1	15
<i>C</i>	8	8	9	8	7	7	5	1
<i>D</i>	9	1	1	5	9	8	4	4
<i>E</i>	1	1	4	1	4	5	6	7
<i>F</i>	8	5	9	8	6	10	8	10
<i>G</i>	5	7	5	4	1	1	7	7
<i>H</i>	9	1	9	10	3	9	9	9

- Los números que aparecen son 1, 4, 5, 6, 7, 8, 9, 10, 15, son 9. A la mitad sería el número en la quinta posición, en este caso 7. Así que tomaremos ese como umbral.

Ejemplo

umbral 7.

	<i>i</i>	<i>ii</i>	<i>iii</i>	<i>iv</i>	<i>v</i>	<i>vi</i>	<i>vii</i>	<i>viii</i>
<i>A</i>	5	9	9	8	7	10	4	15
<i>B</i>	8	10	9	10	6	7	1	15
<i>C</i>	8	8	9	8	7	7	5	1
<i>D</i>	9	1	1	5	9	8	4	4
<i>E</i>	1	1	4	1	4	5	6	7
<i>F</i>	8	5	9	8	6	10	8	10
<i>G</i>	5	7	5	4	1	1	7	7
<i>H</i>	9	1	9	10	4	9	9	9

Ejemplo

umbral 7. Matching es perfecto

	<i>i</i>	<i>ii</i>	<i>iii</i>	<i>iv</i>	<i>v</i>	<i>vi</i>	<i>vii</i>	<i>viii</i>		
<i>A</i>	[1]	0	0	0	1	0	1	0		<i>Ai</i> : 5
<i>B</i>	0	0	0	0	1	1	[1]	0		<i>Bvii</i> : 1
<i>C</i>	0	0	0	0	1	[1]	1	1		<i>Cvi</i> : 7
<i>D</i>	0	1	1	1	0	0	1	[1]	<i>ii</i>	<i>Dviii</i> : 4
<i>E</i>	1	1	[1]	1	1	1	1	1		<i>Eiii</i> : 4
<i>F</i>	0	1	0	0	[1]	0	0	0	<i>v</i>	<i>Fv</i> : 6
<i>G</i>	1	1	1	[1]	1	1	1	1		<i>Giv</i> : 4
<i>H</i>	0	[1]	0	0	1	0	0	0	<i>s</i>	<i>Hii</i> : 1
		<i>H</i>			<i>H</i>			<i>D</i>		<i>max</i> : 7

Ejemplo

Como en 7 hay matching, debemos bajar el umbral.

Ejemplo

Como en 7 hay matching, debemos bajar el umbral. Los números que han quedado son 1,4,5,6,7.

Ejemplo

Como en 7 hay matching, debemos bajar el umbral. Los números que han quedado son 1,4,5,6,7. Bajando a la mitad de esta lista de cinco elementos, sería el elemento numero 3, es decir el número 5

Ejemplo

umbral 5.

Ejemplo

umbral 5.

	<i>i</i>	<i>ii</i>	<i>iii</i>	<i>iv</i>	<i>v</i>	<i>vi</i>	<i>vii</i>	<i>viii</i>
<i>A</i>	5	9	9	8	7	10	4	15
<i>B</i>	8	10	9	10	6	7	1	15
<i>C</i>	8	8	9	8	7	7	5	1
<i>D</i>	9	1	1	5	9	8	4	4
<i>E</i>	1	1	4	1	4	5	6	7
<i>F</i>	8	5	9	8	6	10	8	10
<i>G</i>	5	7	5	4	1	1	7	7
<i>H</i>	9	1	9	10	4	9	9	9

Ejemplo

umbral 5.

	<i>i</i>	<i>ii</i>	<i>iii</i>	<i>iv</i>	<i>v</i>	<i>vi</i>	<i>vii</i>	<i>viii</i>		
<i>A</i>	[1]	0	0	0	0	0	1	0		<i>Ai</i> : 5
<i>B</i>	0	0	0	0	0	0	[1]	0	<i>vii</i>	<i>Bvii</i> : 1
<i>C</i>	0	0	0	0	0	0	1	[1]	<i>viii</i>	<i>Cviii</i> : 1
<i>D</i>	0	1	[1]	1	0	0	1	1	<i>ii</i>	<i>Diii</i> : 1
<i>E</i>	1	1	1	1	1	[1]	0	0	<i>iii</i>	<i>Evi</i> : 5
<i>F</i>	0	[1]	0	0	0	0	0	0	<i>s</i>	<i>Fii</i> : 5
<i>G</i>	1	0	1	[1]	1	1	0	0	<i>iv</i>	<i>Giv</i> : 4
<i>H</i>	0	1	0	0	[1]	0	0	0		<i>Hv</i> : 4
	<i>E</i>	<i>F</i>	<i>D</i>	<i>D</i>	<i>E</i>	<i>E</i>	<i>D</i>	<i>D</i>		<i>max</i> : 5

Ejemplo

Como con umbral 5 hay matching, debemos bajar el umbral.

Ejemplo

Como con umbral 5 hay matching, debemos bajar el umbral. Los números que han quedado son 1,4.

Ejemplo

umbral 4.

Ejemplo

umbral 4.

	<i>i</i>	<i>ii</i>	<i>iii</i>	<i>iv</i>	<i>v</i>	<i>vi</i>	<i>vii</i>	<i>viii</i>
<i>A</i>	5	9	9	8	7	10	4	15
<i>B</i>	8	10	9	10	6	7	1	15
<i>C</i>	8	8	9	8	7	7	5	1
<i>D</i>	9	1	1	5	9	8	4	4
<i>E</i>	1	1	4	1	4	5	6	7
<i>F</i>	8	5	9	8	6	10	8	10
<i>G</i>	5	7	5	4	1	1	7	7
<i>H</i>	9	1	9	10	4	9	9	9

Ejemplo

umbral 4. Hay una fila con todos ceros, no puede haber matching.

	<i>i</i>	<i>ii</i>	<i>iii</i>	<i>iv</i>	<i>v</i>	<i>vi</i>	<i>vii</i>	<i>viii</i>
<i>A</i>	0	0	0	0	0	0	1	0
<i>B</i>	0	0	0	0	0	0	1	0
<i>C</i>	0	0	0	0	0	0	0	1
<i>D</i>	0	1	1	0	0	0	1	1
<i>E</i>	1	1	1	1	1	0	0	0
<i>F</i>	0	0	0	0	0	0	0	0
<i>G</i>	0	0	0	1	1	1	0	0
<i>H</i>	0	1	0	0	1	0	0	0

Ejemplo

- Como para umbral=4 no hay matching y para umbral=5 sí, tenemos que el matching perfecto que dimos para umbral=5 minimiza el máximo.
- En gral puede no encontrarse matching porque haya una fila o columna de todos 0s, o bien todas las filas y columnas tienen al menos un 1, se corre el algoritmo y se encuentra como en la prueba de Hall un S con $|\Gamma(S)| < |S|$.