Práctica de asignación óptima de pilotos a vuelos
Curso 2017-2018 Q1

Algoritmia - FIB

Documentación de la realización de la práctica

Sebastián Sánchez Menéndez
Daniel Martínez Bordes
Ferran Martínez Felipe

Contenido

1. Introducción

Uno de los sectores que necesita un mayor número de técnicas de optimización para la gestión de recursos es el transporte aéreo. Ésto es debido a que a diferencia de en otros sectores, en éste cada recurso tiene un coste muy elevado, y por tanto obtener rentabilidad pasa por optimizar al máximo posible cada uno de los recursos disponibles.

Un caso concreto de este hecho consiste en la generación de los horarios de los diferentes vuelos que salen de distintos aeropuertos cada día para los diferentes pilotos de los que dispone la compañía. La idea tras la optimización aplicada en este caso consiste en distribuir a los pilotos de la forma más eficiente posible, a efectos de minimizar el número de pilotos necesario para cumplir con un horario establecido.

Para realizar la asignación y los horarios de los vuelos una de las técnicas más usada consiste en transformar el problema en un problema de flujos, para que a partir del flujo y el grafo de flujos resultante seamos capaces de saber cuantos pilotos necesitaremos y que horario va a seguir cada piloto.

Y dicho ésto toca centrarse en la práctica realizada. La idea detrás de la práctica consiste en implementar un algoritmo tal que sea capaz de, dada una lista de vuelos, generar un horario que minimice el número de pilotos necesarios (respetando unas restricciones). Se ha pedido que se realicen diferentes versiones del algoritmo que cumplan distintos objetivos y respeten distintas restricciones (cada versión está explicada en la Sección 2) a partir de ciertos algoritmos de Max-Flow (explicados en la versión 3).

2. Explicación de las diferentes versiones del problema

2.1 Versión con Demands y Lower Bounds

Esta versión es la explicada en el libro de Kleinberg & Tardos, y se basa en reducir el problema de los horarios para pilotos en un problema de circulacion con demandas y límites inferiores.
La idea para solucionar el problema de optimización es realizar una búsqueda binaria entre 0 y N, donde N es el número de vuelos, buscando el valor mínimo de pilotos para el cual es posible la circulación en la red.
Para buscar una circulación posible con K pilotos generamos la red de flujo de la forma G = {s U t U B} donde B es un grafo bipartido {A U B}, en el cual por cada vuelo tenemos un nodo perteneciente a A y otro perteneciente a B, y los arcos son los siguientes:

- Por cada nodo a perteneciente a A, un arco (s, a) con capacidad 1 y límite inferior 0.
- Por cada nodo b perteneciente a B, un arco (b, t) con capacidad 1 y límite inferior 0.

- Por cada nodo a y su correspondiente nodo b, un arco (a,b) con capacidad 1 y límite inferior 1.
- Por cada par de nodos b perteneciente a B y a perteneciente a A tal que se puede pilotar el vuelo correspondiente al nodo b después del vuelo correspondiente al nodo a, un arco con capacidad 1 y límite inferior 0.
- Un arco (s,t) con capacidad K.

Además, a los nodos s y t hay que añadirles una demanda de -K y K respectivamente.

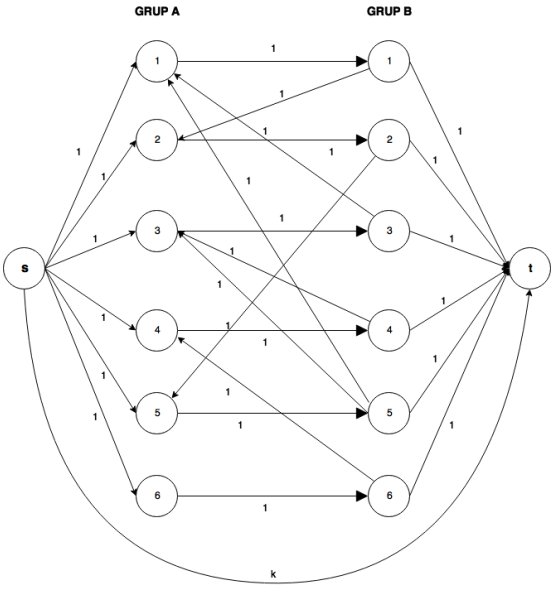

Figura 1.*Grafo generado usando los vuelos de ejemplo antes de aplicar las transformaciones de circulaciones con demandas.*

Para codificar la red utilizamos dos matrices, una para las capacidades y otra para los flujos, de tamaño 2*n+4.
2*n de estos nodos corresponden al grafo bipartido de los vuelos, y los otros cuatro nodos son s, t y dos nuevos nodos s' y t' necesarios para poder plantear el problema de circulaciones con demandas.

Una vez leída la entrada y declaradas las matrices, la red de flujo final tendrá los siguientes arcos:

- Por cada nodo a perteneciente a A, un arco (a, t') con capacidad 1.
- Por cada nodo b perteneciente a B, un arco (s', b) con capacidad 1.
- Por cada par de nodos b perteneciente a B y a perteneciente a A tal que se puede pilotar el vuelo correspondiente al nodo b después del vuelo correspondiente al nodo a, un arco con capacidad 1.
- Un arco (s', s) con capacidad K.
- Un arco (t, t') con capacidad K.

- Un arco (s, t) con capacidad K.

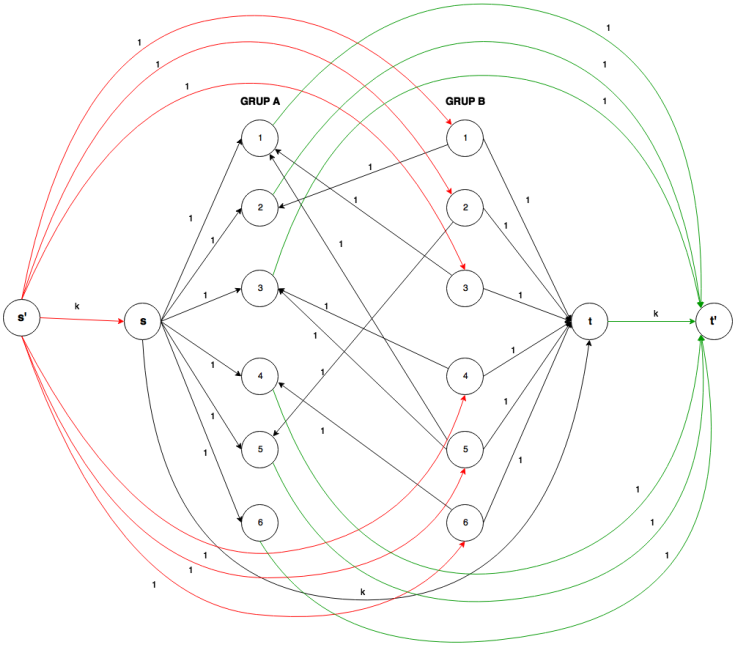

Figura 2.*Grafo generado usando los vuelos de ejemplo sobre el que calcularemos el buscaremos una circulación factible con K pilotos.*

La diferencia sustancial entre ambas versiones es la forma de buscar qué vuelos son posibles pilotar desde otro vuelo. En la primera versión basta con realizar un bucle que busque, para cada vuelo, qué otros vuelos tienen ciudad de origen igual a su ciudad de destino y mantienen el margen de 15 minutos entre sus tiempos de salida.
En la versión 2, en cambio, si un piloto es capaz de, después de pilotar un cierto vuelo u, ser pasajero de otro vuelo v de forma que llegue a pilotar un tercer vuelo w, entonces hemos de añadir el arco (u,w). Esto conlleva que tenemos que realizar una búsqueda a través de los diferentes vuelos que se pueden realizar después de un vuelo concreto (o lo que es lo mismo, que para cada nodo del grupo de nodos conectados a s tendremos que realizar una búsqueda en profundidad, ya que si de un nodo i podemos llegar a un nodo j y del nodo j podemos llegar a un nodo k significa que podemos llegar de i a k.). Ésta búsqueda, costosa en cuestión de tiempo (debemos recorrernos varias veces un grafo con cientos de nodos) ha sido optimizada mediante el uso de programación dinámica, guardando las conexiones de un nodo con todos sus niveles inferiores la primera vez que lo visitamos y copiando sus capacidades las siguientes veces que se visita.

Después de generar esta red final, lanzamos un algoritmo de Max-Flow, y si es posible la circulación guardamos el número de pilotos y la matriz de flujo correspondiente.
Al terminar la búsqueda binaria, tenemos guardado el número óptimo de pilotos y la red de flujo correspondiente de la última circulación factible.

A partir de la red de flujo se puede generar el horario de los pilotos. La forma en que lo hacemos es buscando un nodo perteneciente a A tal que no exista flujo desde ningún nodo de B hacia él, lo cual implica que dicho nodo representa el inicio del turno de un piloto. Una vez encontrado este nodo, hemos de buscar para su nodo correspondiente en B un flujo de salida hacia otro nodo de A y añadimos los vuelos correspondiente a su horario. Una vez encontremos un vuelo cuyo nodo de B no tiene flujo dirigido a ningún nodo de A, sabemos que el piloto ha acabado su turno y devolvemos su horario.
Para optimizar esta función, cada vez que añadimos un vuelo al horario de un piloto lo marcamos como visitado en un vector, y solo comprobamos si es un nuevo piloto, o si el vuelo pertenece al horario que se está generando si el vuelo no ha sido visitado todavía.

El coste temporal es la suma de los costes de:

- Generar el grafo de entrada: en la versión 1 realizamos dos bucles con coste O(n) y un bucle con coste $O(n^2)$, por lo que el coste es $O(n^2)$. En la versión 2, sin embargo, el coste pasa a ser $O(n^3)$, debido a la búsqueda en profundidad que se realiza para generar la red de flujo
- El coste de la búsqueda binaria es log(n), y en cada iteración lanzamos un algoritmo de resolución de Max-Flow. Por tanto, este coste puede ser $O(\log(n)NE^2)$ en el caso de Edmonds-Karp y $O(\log(n)EV^2)$ en el caso de Dinic.
- Por último, para generar la salida, dado que solo comprobamos los nodos que todavía no han sido utilizados, el coste es también de $O(n^2)$.

Por lo tanto, el coste final del algoritmo es $O(\log(n)NE^2)$ en el caso de Edmonds-Karp o $O(\log(n)EV^2)$ en el caso de Dinic, ya que es el coste asintótico dominante.

2.2 Versión con Maximum Bipartite Matching

La idea tras esta implementación consiste en convertir el problema en un problema de flujos tal que el Maximum Bipartite Matching represente el horario que han de seguir los vuelos (ej.: si el vuelo 2 está conectado con el vuelo 3 significa que el vuelo 3 puede hacerse después del vuelo 2 por el mismo piloto).

Para ello, generamos una red de flujos tal que los nodos de la red quedan repartidos entre dos grupos, a los que a efectos prácticos llamaremos grupo A y grupo B, además de unos nodos extra que serán nuestros nodos s y t. Por cada vuelo, generaremos un nodo en cada uno de los grupos, tales que s estará conectado a todos los nodos del grupo A, todos los nodos del grupo B están conectados a t y habrá una conexión entre el nodo i del grupo A y el nodo j del grupo B si después del vuelo i puede realizarse el vuelo j. Todas las aristas tendrán capacidad 1.

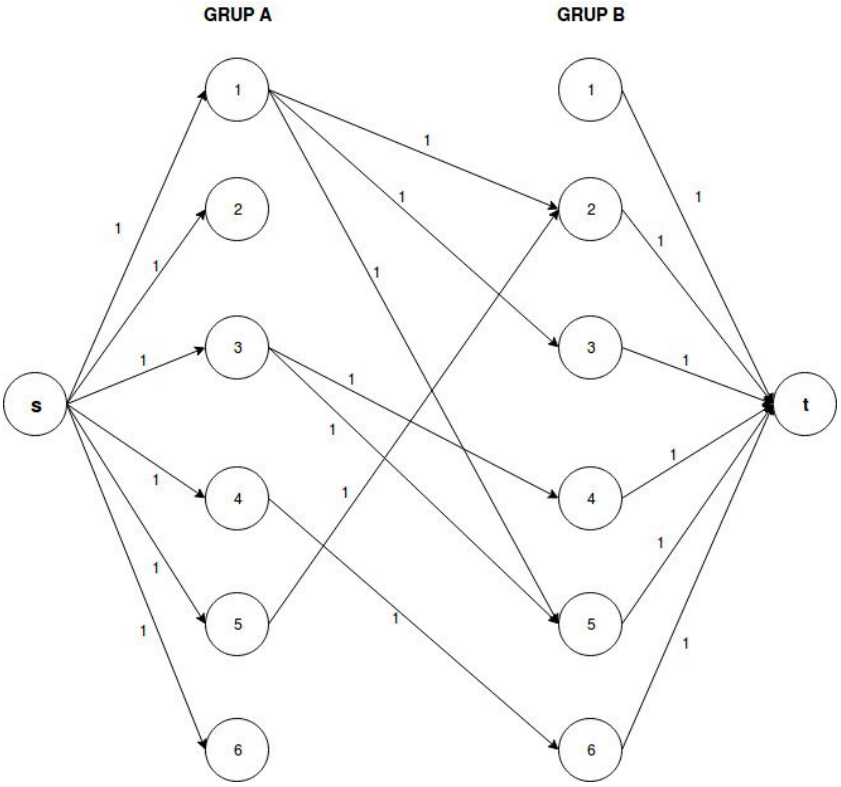

Figura 3.*Grafo generado usando los vuelos de ejemplo sobre el que calcularemos el Maximum Bipartite Matching.*

Para codificar el grafo utilizamos dos matrices, una para las capacidades y otra para los flujos de tamaño 2*n+4, tal que 2*n de estos nodos corresponden al grafo bipartido de los vuelos y los otros cuatro nodos son s, t.

Tras generar el grafo, basta con aplicar cualquier algoritmo que encuentre el Max-Flow para encontrar el Maximum Bipartite Matching del grafo entre el grupo A y el grupo B. Este Maximum Bipartite Matching representará el schedule de los vuelos que hemos de seguir, de tal forma que si un nodo del grupo A está conectado a uno del grupo B significa que en el horario tendrá al nodo del grupo B después del del grupo A.

La red de flujo resultante tendrá los siguientes arcos:

- Por cada nodo a perteneciente a A, un arco (s, a) con capacidad 1.
- Por cada nodo b perteneciente a B, un arco (b, t) con capacidad 1.
- Por cada par de nodos b perteneciente a B y a perteneciente a A tal que se puede pilotar el vuelo correspondiente al nodo b después del vuelo correspondiente al nodo a, un arco con capacidad 1.

Cabe recalcar que el horario generado a partir de este grafo será aquel que requiera del número mínimo de pilotos, ya que por las propiedades de un Maximum Bipartite Matching podemos asegurar que el grafo de flujos resultante será aquel con el máximo número de

conexiones entre A y B tal que no se repitan nodos (y a más concatenaciones de vuelos, menor número de pilotos requerido ya que un único piloto podrá hacer más vuelos consecutivos).

Para generar los horarios a partir de la red de flujos, basta con recorrerse los nodos del grupo B. Cada nodo que no tenga conexión con t será forzosamente el primer vuelo de un horario, ya que de no ser así iría precedido de uno del grupo A, y por tanto tendríamos una arista de un nodo de A al nodo de B y flujo del nodo de B a t. A partir de este nodo, podemos lanzar un DFS para generar el horario que parte de ese nodo.

Así, y una vez recorridos todos los nodos del grupo B tales que no tienen flujo con t, tendremos el horario para todos los pilotos. Para saber el número de pilotos a partir de los horarios, basta con contar los horarios que hemos obtenido.

Dicho lo anterior, pasemos a comparar las dos versiones pedidas en la práctica.

En la primera versión se nos pide que un piloto solo puede hacer un vuelo i después de un nodo j si la ciudad de destino de i y la ciudad de origen de j coinciden, y el vuelo j sale mínimo 15 minutos después del nodo i.

Para hacer ésto, basta con recorrerse todos los vuelos, y para cada vuelo realizar una conexión los otros vuelos que puedan realizarse desde el primero.

Posteriormente, y tras añadir un nodo s y un t, puede aplicarse Max-Flow para obtener el grafo de flujos a procesar. Una vez generado el grafo de flujos la salida se procesa tal y como se ha comentado anteriormente en este apartado.

En la segunda versión se nos pide una ampliación de la primera versión, en la que ahora un piloto puede realizar un vuelo w después de un vuelo i si entre los dos vuelos existe un vuelo j que va de la ciudad de destino de i a la ciudad de origen de w. Evidentemente en esta versión se han de respetar también la diferencia de 15 minutos entre un vuelo y el siguiente, aunque un piloto lo realice de pasajero.

Para realizar esta versión, basta con realizar un DFS desde cada uno de los vuelos. A partir de cada vuelo inicial, iremos recorriendo todos los vuelos que sean accesibles desde el primero, respetando las restricciones comentadas anteriormente. Es decir, que si se puede conectar un nodo i a un nodo j, y este nodo j a un nodo k, debemos conectar i con k. Evidentemente, para realizar este procedimiento podemos utilizar programación dinámica, tal que no tengamos que recorrernos todo el grafo para cada vuelo (solo habrá que hacerlo una vez).

El coste temporal es la suma de los costes de:

- Generar el grafo de entrada: en la versión 1 realizamos dos bucles con coste O(n) y un bucle con coste $O(n^2)$, por lo que el coste es $O(n^2)$. En la versión 2, sin embargo, el coste es la suma de dos bucles de coste O(n) más la suma de cada uno de los DFS (coste $O(n^3)$).

- Aplicar Max-Flow: En el caso de utilizar Edmonds-Karp el coste es $O(m^2n)$. En el caso de usar Dinic el coste es $O(mn^2)$
- Generar la salida: Para generar la salida se aplica un DFS sobre cada uno de los nodos del grupo B, lo cual tiene coste $O(n^3)$.

3. Explicación de las diferentes versiones del algoritmo de Max-Flow

3.1 Edmonds-Karp

El algoritmo calcula el flujo máximo, basándose en encontrar un camino posible en el grafo residual, en cada iteración, con un BFS. El algoritmo busca un camino de aumento de flujo y cuando lo encuentra, calcula su bottleneck flow, actualiza el flujo máximo del grafo y el flujo que pasa por arista del augmenting path modificando el grafo residual. En realidad el grafo residual no está creado, sinó

3.2 Dinic

Este algoritmo busca un flujo máximo al igual que el algoritmo de Edmonds-Karp, pero en lugar de buscar un camino de aumento en el grafo residual, se basa en bloquear el flujo en el grafo de niveles. El grafo de niveles se define como el grafo en que solo existen los arcos (u,v) tal que el L(u) + 1 = L(v). L(u) es el nivel de el nodo u, y tiene el valor de la distancia mínima desde el nodo fuente de la red. Para bloquear el flujo en dicho grafo, se generan flujos de forma que se saturen los caminos existentes, desde s hasta t, y una vez no existe ningún camino de s a t se vuelve a calcular el grafo de niveles, hasta que en el grafo de niveles no exista ningún camino hasta t.

4. Resultados experimentales

Resultados media de tiempo versión 1, Edmonds Karp, Maximum Bipartite Matching:

Grupo de problemas	Tiempo (ms)
instance_100_2_*.air	365.333
instance_100_3_*.air	327.882
instance_100_4_*.air	290.158
instance_100_5_*.air	279.249
instance_100_6_*.air	296.692

instance_100_7_*.air	285.918
instance_100_8_*.air	221.918
instance_100_9_*.air	245.178
instance_100_10_*.air	275.59
instance_100_11_*.air	252.313
instance_100_12_*.air	268.622
instance_100_13_*.air	281.413
instance_100_14_*.air	276.116
instance_100_15_*.air	252.712
instance_100_16_*.air	226.931
instance_100_17_*.air	281.13
instance_100_18_*.air	264.163
instance_100_19_*.air	276.16
instance_100_20_*.air	232.431
instance_100_21_*.air	209.636
instance_100_22_*.air	205.393
instance_100_23_*.air	242.201
instance_100_24_*.air	208.757
instance_100_25_*.air	190.684
instance_100_26_*.air	254.721
instance_100_27_*.air	219.506
instance_100_28_*.air	229.667
instance_100_29_*.air	199.403
instance_100_30_*.air	212.981

Resultados media de tiempo versión 1, Alg. Dinic, Maximum Bipartite Matching:

Grupo de problemas	Tiempo (ms)
instance_100_2_*.air	643.252
instance_100_3_*.air	257.06
instance_100_4_*.air	192.4
instance_100_5_*.air	139.73
instance_100_6_*.air	138.602
instance_100_7_*.air	126.726
instance_100_8_*.air	936.566
instance_100_9_*.air	890.816
instance_100_10_*.air	971.119
instance_100_11_*.air	781.289
instance_100_12_*.air	974.598
instance_100_13_*.air	865.561
instance_100_14_*.air	84.493
instance_100_15_*.air	743.869
instance_100_16_*.air	602.677
instance_100_17_*.air	725.585
instance_100_18_*.air	721.567
instance_100_19_*.air	712.978
instance_100_20_*.air	586.338
instance_100_21_*.air	552.039
instance_100_22_*.air	545.731
instance_100_23_*.air	670.545
instance_100_24_*.air	575.453
instance_100_25_*.air	513.666
instance_100_26_*.air	659.463

instance_100_27_*.air	55.389
instance_100_28_*.air	529.049
instance_100_29_*.air	553.024
instance_100_30_*.air	526.492

Resultados media de tiempo versión 2, Edmonds Karp, Maximum Bipartite Matching:

Grupo de problemas	Tiempo (ms)
instance_100_2_*.air	801.989
instance_100_3_*.air	649.174
instance_100_4_*.air	520.301
instance_100_5_*.air	501.054
instance_100_6_*.air	456.489
instance_100_7_*.air	449.572
instance_100_8_*.air	348.767
instance_100_9_*.air	376.68
instance_100_10_*.air	428.424
instance_100_11_*.air	376.221
instance_100_12_*.air	387.184
instance_100_13_*.air	407.755
instance_100_14_*.air	403.537
instance_100_15_*.air	359.102
instance_100_16_*.air	338.261
instance_100_17_*.air	381.474
instance_100_18_*.air	361.546
instance_100_19_*.air	379.113
instance_100_20_*.air	303.946
instance_100_21_*.air	275.668

instance_100_22_*.air	258.349
instance_100_23_*.air	343.099
instance_100_24_*.air	280.432
instance_100_25_*.air	252.554
instance_100_26_*.air	314.128
instance_100_27_*.air	284.034
instance_100_28_*.air	283.374
instance_100_29_*.air	255.619
instance_100_30_*.air	278.271

Resultados media de tiempo versión 2, Dinic, Maximum Bipartite Matching:

Grupo de problemas	Tiempo (ms)
instance_100_2_*.air	1021.03
instance_100_3_*.air	697.405
instance_100_4_*.air	474.723
instance_100_5_*.air	400.422
instance_100_6_*.air	427.015
instance_100_7_*.air	327.909
instance_100_8_*.air	289.782
instance_100_9_*.air	292.755
instance_100_10_*.air	321.651
instance_100_11_*.air	251.893
instance_100_12_*.air	258.57
instance_100_13_*.air	281.223
instance_100_14_*.air	266.735
instance_100_15_*.air	196.224
instance_100_16_*.air	191.763

instance_100_17_*.air	229.413
instance_100_18_*.air	220.369
instance_100_19_*.air	206.026
instance_100_20_*.air	162.208
instance_100_21_*.air	142.169
instance_100_22_*.air	127.702
instance_100_23_*.air	163.004
instance_100_24_*.air	132.5
instance_100_25_*.air	121.263
instance_100_26_*.air	158.722
instance_100_27_*.air	135.477
instance_100_28_*.air	135.033
instance_100_29_*.air	109.341
instance_100_30_*.air	124.757

5. Bibliografia

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein(2001). "26. Maximum Flow". Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill.
- Kleinberg, J. and Tardos, E. (2009). Algorithm Design.
- Edmonds, J. and Karp, R.M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM 19, 2 (1972), 248–264.