Asignación Òptima de Pilotos a Vuelos en una Compañía Aérea*

Grau A Curs 2017-2018

Departament de Ciències de la Computació
alg@cs.upc.edu

Resum

Con este proyecto se pretende que experimentéis el uso de técnicas algorítmicas vistas en clase en problemas reales. En esta ocasión nos centraremos en algoritmos de max-flow para resolver problemas de asignación de recursos. En particular, la asignación de pilotos a vuelos de una compañía aérea. La práctica se hará en grupos de 3 persones (excepcionalmente 2, bajo autorización expresa). La ***composición de los grupos*** *se deberá comunicar a alg@cs.upc.edu antes del* ***19 de diciembre de 2017****.*

La entrega de la práctica se hará on-line a través del racó. Tendréis tiempo hasta las 23:59 horas del día ***12 de enero de2017****. Algunos grupos pueden ser convocados para una entrevista personal, fecha a decidir entre el 24 y el 26 de Enero, con prueba interactiva.* ***Es obligatorio que en la entrevista (si hace falta) estén presentes todos los miembros del grupo****.*

I. Descripción del problema

El transporte aéreo es uno de los sectores que hace un uso más intensivo de técnicas avanzadas de optimización para la gestión eficiente de sus recursos. Esto no es de extrañar si pensamos en el elevado coste que tienen sus recursos. En este proyecto nos vamos a centrar en uno de los muchos problemas del sector: la asignación de pilotos a los vuelos de una compañía aérea. Además, vamos a hacer una serie de simplificaciones para que el volumen de trabajo y el tipo de técnicas necesarias se ajusten a los contenidos y objetivos de la asignatura. Consideremos los vuelos de una compañía a lo largo de un día. Cada avión de la compañía va haciendo trayectos. Entre cada par de trayectos consecutivos hay una pausa mínima de 15 minutos, pero que en algunos casos puede ser de varias horas. A efectos de este proyecto, asumiremos que los trayectos y los horarios de cada uno de los aviones ya se han fijado y lo que nos queda es asignar un piloto a cada trayecto. Nuestro objetivo sera establecer el numero mínimo de pilotos que nos hace falta para pilotar cada uno de los trayectos (véase [Kleinberg and Tardos, 2005], pagina 387). La asignación de pilotos a trayectos tiene que cumplir:

- Cada vuelo tiene exactamente un piloto asignado, que es el que lo pilota.
- Cada piloto puede pilotar los vuelos que le han sido asignados. En la vida real, esto requiere cumplir numerosas condiciones. Algunas son obvias como que los vuelos asignados a un mismo piloto no se pueden solapar. Otras son mas complejas y tienen que ver con reglas sobre seguridad (e.g. descanso mínimo entre vuelos), jornada laboral (e.g. horas máximas trabajadas), etc. En este proyecto nos conformaremos con dos versiones muy simplificadas. Nos referiremos a ellas como la *versión 1* y la *versión 2*.

Formalicemos un poco mas el problema. Tenemos n trayectos $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$. Cada trayecto es una tupla $T_i = (o_i, d_i, h1_i, h2_i)$ donde o_i y d_i son las ciudades origen y destino del trayecto, y $h1_i$ y $h2_i$ son las horas de salida y llegada del trayecto. El tiempo lo mediremos en

*La versión más actualizada de este documento, así como cualquier material adicional relacionado se publicará en el Racó.

minutos. Como nos restringimos a un día, todos los tiempos serán enteros del rango $0, \ldots, (24 \times 60)$. Por ejemplo, con 3 ciudades (Barcelona (BCN), París (CDG), Londres (LGW)) podríamos tener el siguiente conjunto de 6 trayectos,

$$T_1 = (BCN, CDG, 0, 100), T_2 = (CDG, BCN, 450, 550), T3 = (CDG, LGW, 150, 250),$$
$$T4 = (LGW, CDG, 600, 700), T5 = (LGX, CDG, 300, 400), T6 = (CDG, BCN, 750, 850).$$

Decimos que k pilotos son suficientes para $\mathcal{T}$ si se puede asignar un piloto a cada trayecto de manera factible. Esto significa que a ningún piloto se le asignen vuelos que se solapen en el tiempo. Además, si miramos la secuencia (ordenada por tiempo) de vuelos asignados a un piloto, tiene que ser posible que el piloto haga la transición entre vuelos consecutivos:

- En la versión 1, decimos que la transición entre T_i y T_j es posible si el destino de T_i y el origen de T_j coinciden (es decir, $d_i = o_j$) y hay un margen de como mínimo 15 minutos entre la hora de llegada de T_i y la hora de salida de T_j (es decir, $h1_j - h2_i \geq 15$).

  Por ejemplo, el conjunto de trayectos anterior lo podrían hacer tres pilotos de la siguiente manera:

  1. (T_1, T_2)
  2. (T_3, T_4)
  3. (T_5, T_6)

  Pero esta asignación no es óptima, pues con solo dos pilotos es suficiente,

  1. (T_1, T_3, T_5, T_2)
  2. (T_4, T_6)

- En la versión 2, también permitimos que un piloto pilote un trayecto T_i y a continuación el trayecto T_j si $d_i \neq o_j$. Pero, en este caso, el piloto tiene que tener tiempo de desplazarse como pasajero (usando trayectos de la compañía y asumiendo un tiempo mínimo de 15 minutos para hacer cambios de avión). Por ejemplo, el conjunto de trayectos anterior lo podrían hacer dos pilotos de la siguiente manera:

  1. (T_1, T_5, T_2) (para poder hacer la transición $T1, T5$, el piloto se trasladara de París a Londres como pasajero en T_3)
  2. (T_3, T_4, T_6).

Es evidente que el numero de pilotos de la solución óptima en la versión 2 siempre va a ser menor o igual que en la versión 1.

II. En qué consiste el proyecto

Tenéis que hacer un programa en C++ que dada una instancia del problema calcule el numero de pilotos k mínimo necesario para pilotar los trayectos. Para cada piloto, se devolverá la lista de trayectos que tiene asignados.

Internamente, el programa calculara el óptimo usando una red de flujos, tal como se describe en [Kleinberg and Tardos, 2005] (pagina 387). El programa tiene que incluir como mínimo el *shortest augmenting path method* de Edmons-Karp (EK). Se valorara positivamente que se incluya algún otro algoritmo alternativo.

El programa tiene que funcionar con los formatos de entrada y salida que se describen a continuación. Para que podáis hacer experimentos, se os facilitara un conjunto de instancias en el formato apropiado.

No se puede utilizar ninguna librería que no sea estándar.

El nivel de sofisticación y esfuerzo dedicado al proyecto es opcional y se tendrá en cuenta en su evaluación. La versión más sencilla, suficiente para aprobar, contendría una implementación de EK resolviendo las versiones 1 y 2. Versiones más sofisticadas del proyecto incluirán la programación de otros algoritmos para resolver max-flow. En este caso se valorará también la inclusión de una comparación de la eficiencia de los algoritmos propuestos con relación a las dos versiones del problema. Tened en cuenta que para hacer comparaciones de eficiencia tendréis que medir el tiempo de los algoritmos y de otros parámetros que penséis que puedan ser útiles para cuantificar el tipo i la cantidad de trabajo que realiza el programa.

III. Formato de Entrada y de Salida

El programa tiene que leer instancias del problema dadas en ficheros de texto donde cada linea es un trayecto especificado con 4 enteros: los dos primeros indican las ciudades de origen y destino (asumimos que se identifican con un índice) y los dos segundos son las horas de salida y llegada (en minutos desde el inicio del día). Es decir, el ejemplo anterior se correspondería con el siguiente fichero:

```
0 1 0 100
1 0 450 550
1 2 150 250
2 1 600 700
2 1 300 400
1 0 750 850
```

La salida del programa tiene que ser un fichero de texto cuya primera linea es el numero de pilotos k óptimo. Seguirán k líneas que contendrán la secuencia de trayectos que pilotara cada piloto. Una salida posible para el ejemplo anterior seria:

```
2
1 3 5 2
4 6
```

IV. Qué hay que entregar

Tenéis que entregar una carpeta etiquetada con el identificador del grupo con lo siguiente:

- Un informe (en pdf) que incluya:
  - Breve descripción que lo que habéis implementado
  - Para cada versión del problema que hayáis considerado (como mínimo la versión 1) y para algoritmo que hayáis implementado (como mínimo Edmons-Karp), el coste temporal del algoritmo en función de los parámetros del problema (numero de vuelos, de ciudades, etc)
  - Si vuestro programa también funciona para la versión 2, discusión de las diferencias de la red de flujos en esta versión respecto a la de la versión 1
  - Tablas de resultados experimentales sobre las instancias que se os han facilitado para cada versión del problema que hayais considerado. Como veréis, los nombres de las instancias tienen tres parámetros. Cada configuración de los dos primeros define un tipo de problemas. El tercer parámetro indica la semilla aleatoria con la que se generó la instan-

cia. Para cada tipo de problemas se han generado 10 instancias. La tabla de resultados tiene que incluir, para cada tipo de problemas, y para cada algoritmo de max-flow que hayáis implementado, el tiempo medio (sobre las 10 instancias) de CPU necesario para resolverlos.

  - Lista de referencias bibliográficas y recursos on-line que hayáis consultado durante la elaboración de la práctica.

- Una carpeta con todas las fuentes necesarias para compilar y ejecutar vuestro(s) programa(s) en un entorno Linux. Se deben incluir instrucciones para la compilación y ejecución en entorno Linux.

- Un fichero de texto "Resultado1.txt" (y también "Resultado2.txt" si habéis considerado la versión 2) que contenga la lista de instancias en orden alfabético seguidas por el valor de la solución óptima. Junto a las instancias encontrareis un ejemplo que incluye el resultado de las 10 primeras. Tenéis que ser muy estrictos con el formato de este fichero, pues lo usaremos para comprobar la corrección de los resultados que obtienen vuestros programas haciendo un "diff" con nuestra solución.

V. Referencias

[Kleinberg and Tardos, 2005] Kleinberg, J. and Tardos, E. (2009). Algorithm Design.

[Edmons and Karp, 1972] 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.