# Proyecto de Algorísmia

Víctor Giménez, Guillem Ferrer y Jordi Armengol

Enero 2018

## 1 Breve descripción de la implementación

Hemos considerado el enunciado como un problema de asignación en el cual tendremos flujos representando a pilotos. Para trazar la red, basándonos en el material recomendado de [Kleinberg and Tardos, 2009], tendremos una red de flujos con demandas y lower bounds. En concreto, tendremos un 'source' S que proporciona un número k de pilotos y un 'sink' T que los absorbe, conectados por un arco de peso k (el flujo que pase por dicho arco no habrá sido usado para solucionar el problema). Además, para cada vuelo, tendremos dos vertices (salida y llegada), conectados por arcos respectivamente a S y a T, con capacidad infinita (en realidad es sencillamente el doble del numero de vuelos pues k $\leq$ numero de vuelos y hay maxflow de m+k). Podría ser capacidad de 1 pero eso solo se aplica en la primera versión del problema. Por último, conectaremos estos dos vertices por un arco con capacidad mínima de 1 y máxima de 1.

Además, si este vuelo es accesible desde otro anterior, conectaremos la llegada del previo a nuestra salida con otro arco de capacidad infinita (de nuevo, $2 \times m$). De esta manera, si el grafo se puede satisfacer (is feasible), entonces significa que: usamos k pilotos y todas los arcos con lowerbound (los vuelos en sí) están cubiertas. Esto es equivalente a decir que se pueden cubrir todos los vuelos con K pilotos. Ahora, sencillamente tendremos que probar con una K más baja. En concreto, nosotros bajamos la k según el numero de pilotos no usados (por el arco de S a T), y usamos una búsqueda dicotómica para determinar la K mínima. Como último punto, a la hora de implementar dicho grafo, lo transformamos en un grafo de capacidades normales, con dos vertices extra para suplir las ofertas de S y las que se forman en eliminar los lowerbounds, y tragarse las demandas de T y las formadas en eliminar

los lowerbounds. Por último, hemos implementado tres algoritmos: Ford-Fulkerson, Edmonds-Karp y Dinic, para obtener las redes de flujos.

# 2 Complejidad temporal

Usaremos: n: numero de ciudades; m: numero de vuelos.

Transformar al grafo de capacidades desde el input es aproximadamente O($(m \times m$ )/n). $\frac{m}{n}$ es la carga esperada por casilla del vector Arrivals. Para calcular el resultado final, llamaremos a la función de maxflow (depende de qué algoritmo, tendrá un coste diferente) un máximo de log m veces, ya que realizamos una búsqueda dicotómica para encontrar la k. Realmente el coste es muchísimo menor que log(m) en la mayoría de casos, pues el problema no se reduce a la mitad sino a la mitad menos los pilotos no usados (arco s-t) en caso de que sea 'feasible'. Experimentalmente, vemos que en caso de m con valores de 250, esperamos que el problema se reduzca a menos de 70 en la primera llamada a la función.

Por último, la transformación a resultado (los pilotos) es O($k \times (|V|+|E|)$): para cada piloto, recorrer el grafo de S a T en un DFS, pues $|V|$ es O(m) y $|E|$ es O($m^2$).

Como k es también O(m) estamos hablando de un coste O($m^3$).

A continuación, viene el coste detallado para cada algoritmo y versión, teniendo en cuenta lo mencionado préviamente.

- Ford-Fulkerson 1:
  - transformación O($m \times m$/n)
  - numero de arcos en el grafo: O(3+4m($m \times m$) = O($m \times m$) = $|E|$
  - numero de vertices: O(4+2m) = O(m) = $|V|$
  - Capacidad máxima: $2 \times m$ = O(m) = C
  - maxflow: O($C \times |E|$) = O($m^3$)
  - maxflow se ejecuta O(log(m)) veces
  - Coste de computar solución = O($m^3$)
  - coste temporal total = O($m \times m/ + (log(m)) \times *(m^3) + m^3$) = O((log(m))*($m^3$))
- Ford-Fulkerson 2:

  - transformación O($m \times m$/n)
  - numero de arcos en el grafo: O(3+5m($m \times m$) = O($m \times m$) = $|E|$
  - numero de vertices: O(4+2m) = O(m) = $|V|$
  - Capacidad máxima: $2 \times m$ = O(m) = C
  - maxflow: O($C \times |E|$) = O($m^3$)
  - maxflow se ejecuta O(log(m)) veces
  - Coste de computar solución = O($m^3$)
  - coste temporal total = O($m \times m/$ + ($log(m)$) $\times *(m^3)$ + $m^3$) = O((log(m))*($m^3$))

- Edmonds-Karp 1:
  - transformación O($m \times m$/n)
  - numero de arcos en el grafo: O(3+4m($m \times m$) = O($m \times m$) = $|E|$
  - numero de vertices: O(4+2m) = O(m) = $|V|$
  - maxflow: O($|V| * |E|^2$) = O($m^5$) o bien O($C \times |E|$) = O($m^3$). El coste menor (O($m^3$)) es el que nos quedaremos, ya que está más ajustado
  - maxflow se ejecuta O(log m) veces
  - Coste de computar solución = O($m^3$)
- coste temporal total = O(m*m/n + (log m)*(m$^3$) + $m^3$) = $O((logm) * (m^3)$)

- Edmonds-Karp 2:
  - transformación O($m \times m$/n)
  - numero de arcos en el grafo: O(3+5m($m \times m$) = O($m \times m$) = $|E|$
  - numero de vertices: O(4+2m) = O(m) = $|V|$
  - maxflow: O($N \times M \times M$) = O($m^5$) o bien O($C \times |E|$) = O($m^3$). El coste menor (O($m^3$)) es el que nos quedaremos, ya que está más ajustado
  - maxflow se ejecuta O(log(m)) veces
  - Coste de computar solución = O($m^3$)

- coste temporal total = O($m \times m$/n + ($log(m)$) $\times$ ($m^3$) + $m^3$) = O(($log(m)$) $\times$ ($m^3$))

- Dinic 1:
  - transformación O($m \times m$/n)
  - numero de arcos en el grafo: O(3+4m+$m \times m$) = $O$(m$\times m$) =—E—$numerodevertices$ : $O$(4+2$m$) = $O(m)$ =—V—$axflow : levels \rightarrow bfs \rightarrow O(m)blockingflowinlevelledgraph$ $dfs$(m$\times n$) bucle $\rightarrow O(n) \rightarrow maxflow : O$(m$\times n^2$)
  - maxflow: O($M \times N^2$) = O($m^2 \times m^2$) = O($m^4$)
  - maxflow se ejecuta O(log(m)) veces coste temporal total: O($m \times$ $m$/n + ($log(m)$) $\times$ ($m^4$) + $m^3$) = O(($log(m)$) $\times$ ($m^4$))
  - Sin embargo, como se puede leer en la última referencia ("On the practical efficiency of various maximum flow algorithms"), en la práctica Dinic tiene una tiempo medio de O($|V|^2$) para redes aleatorias, incluso densas. En este caso, implicaría un coste total de: O(($log(m)$) $\times m^2$), siendo el más rápido.

- Dinic 2:
  - transformación O($m \times m$/n)
  - numero de arcos en el grafo: O(3+5m+$m \times m$) = $O$(m$\times m$) =—E—$numerodevertices$ : $O$(4+2$m$) = $O(m)$ =—V—$axflow : levels \rightarrow bfs \rightarrow O(m)blockingflowinlevelledgraph$ $dfs$(m$\times n$) bucle $\rightarrow O(n) \rightarrow maxflow : O$(m$\times n^2$)
  - maxflow: O($M \times N^2$) = O($m^2 \times m^2$) = O($m^4$)
  - maxflow se ejecuta O(log(m)) veces coste temporal total: O($m \times$ $m$/n + ($log(m)$) $\times$ ($m^4$) + $m^3$) = O(($log(m)$) $\times$ ($m^4$))
  - Sin embargo, como se puede leer en la última referencia ("On the practical efficiency of various maximum flow algorithms"), en la práctica Dinic tiene una tiempo medio de O($|V|^2$) para redes aleatorias, incluso densas. En este caso, implicaría un coste total de: O(($log(m)$) $\times m^2$), siendo el más rápido.

# 3 Diferencia entre la versión 1 y la 2

Sencillamente subiremos la capacidad del arco que conecta la salida y la llegada de un mismo vuelo, permitiendo a pilotos 'pasar' por este arco sin

ser piloto, o mejor dicho, siendo pasajeros. Además, si no hubieramos puesto capacidad 'infinita' entre vuelos, S y T, ahora tendríamos que hacerlo ahora para dejar que hubiera flujo de pilotos entre un vuelo y otro no limitado a uno.

Un ejemplo de la razón por la cuál tenemos que poner dicha capacidad es el caso siguiente: cien vuelos salen a primera hora desde cien ciudades distintas para ir a la misma. De ahí, tras un buen rato, sale un vuelo a otra ciudad. Por último, de esta última ciudad, salen cien vuelos a las cien ciudades del principio. Si no pusieramos capacidad infinita en los arcos mencionados, solo podrían viajar dos pilotos en el vuelo intermedio.

# 4 Tabla de resultados experimentales

| | |
|---|---|
| instance 100 10 v1Dinic | 0.055 |
| instance 100 10 v2Dinic | 0.066 |
| instance 100 10 v1Edmond | 0.248 |
| instance 100 10 v2Edmond | 0.256 |
| instance 100 10 v1Ford | 0.403 |
| instance 100 10 v2Ford | 1.433 |
| instance 100 11 v1Dinic | 0.046 |
| instance 100 11 v2Dinic | 0.055 |
| instance 100 11 v1Edmond | 0.209 |
| instance 100 11 v2Edmond | 0.216 |
| instance 100 11 v1Ford | 0.342 |
| instance 100 11 v2Ford | 1.24 |
| instance 100 12 v1Dinic | 0.044 |
| instance 100 12 v2Dinic | 0.053 |
| instance 100 12 v1Edmond | 0.208 |
| instance 100 12 v2Edmond | 0.22 |
| instance 100 12 v1Ford | 0.351 |
| instance 100 12 v2Ford | 1.251 |
| instance 100 13 v1Dinic | 0.043 |
| instance 100 13 v2Dinic | 0.052 |
| instance 100 13 v1Edmond | 0.214 |
| instance 100 13 v2Edmond | 0.228 |
| instance 100 13 v1Ford | 0.363 |
| instance 100 13 v2Ford | 1.306 |
| instance 100 14 v1Dinic | 0.039 |
| instance 100 14 v2Dinic | 0.052 |
| instance 100 14 v1Edmond | 0.205 |
| instance 100 14 v2Edmond | 0.22 |
| instance 100 14 v1Ford | 0.347 |
| instance 100 14 v2Ford | 1.259 |
| instance 100 15 v1Dinic | 0.039 |
| instance 100 15 v2Dinic | 0.042 |
| instance 100 15 v1Edmond | 0.194 |
| instance 100 15 v2Edmond | 0.203 |
| instance 100 15 v1Ford | 0.327 |
| instance 100 15 v2Ford | 1.099 |

| | |
|---|---|
| instance 100 16 v1Dinic | 0.033 |
| instance 100 16 v2Dinic | 0.043 |
| instance 100 16 v1Edmond | 0.181 |
| instance 100 16 v2Edmond | 0.19 |
| instance 100 16 v1Ford | 0.297 |
| instance 100 16 v2Ford | 1.031 |
| instance 100 17 v1Dinic | 0.037 |
| instance 100 17 v2Dinic | 0.045 |
| instance 100 17 v1Edmond | 0.199 |
| instance 100 17 v2Edmond | 0.211 |
| instance 100 17 v1Ford | 0.347 |
| instance 100 17 v2Ford | 1.237 |
| instance 100 18 v1Dinic | 0.036 |
| instance 100 18 v2Dinic | 0.04 |
| instance 100 18 v1Edmond | 0.194 |
| instance 100 18 v2Edmond | 0.202 |
| instance 100 18 v1Ford | 0.324 |
| instance 100 18 v2Ford | 1.15 |
| instance 100 19 v1Dinic | 0.034 |
| instance 100 19 v2Dinic | 0.041 |
| instance 100 19 v1Edmond | 0.181 |
| instance 100 19 v2Edmond | 0.195 |
| instance 100 19 v1Ford | 0.292 |
| instance 100 19 v2Ford | 1.053 |
| instance 100 2 v1Dinic | 0.14 |
| instance 100 2 v2Dinic | 0.158 |
| instance 100 2 v1Edmond | 0.561 |
| instance 100 2 v2Edmond | 0.58 |
| instance 100 2 v1Ford | 1.106 |
| instance 100 2 v2Ford | 2.17 |
| instance 100 20 v1Dinic | 0.033 |
| instance 100 20 v2Dinic | 0.04 |
| instance 100 20 v1Edmond | 0.176 |
| instance 100 20 v2Edmond | 0.188 |
| instance 100 20 v1Ford | 0.302 |
| instance 100 20 v2Ford | 1.08 |

| | |
|---|---|
| instance 100 21 v1Dinic | 0.031 |
| instance 100 21 v2Dinic | 0.041 |
| instance 100 21 v1Edmond | 0.174 |
| instance 100 21 v2Edmond | 0.185 |
| instance 100 21 v1Ford | 0.304 |
| instance 100 21 v2Ford | 1.008 |
| instance 100 22 v1Dinic | 0.029 |
| instance 100 22 v2Dinic | 0.038 |
| instance 100 22 v1Edmond | 0.165 |
| instance 100 22 v2Edmond | 0.178 |
| instance 100 22 v1Ford | 0.288 |
| instance 100 22 v2Ford | 0.988 |
| instance 100 23 v1Dinic | 0.032 |
| instance 100 23 v2Dinic | 0.038 |
| instance 100 23 v1Edmond | 0.179 |
| instance 100 23 v2Edmond | 0.192 |
| instance 100 23 v1Ford | 0.319 |
| instance 100 23 v2Ford | 1.053 |
| instance 100 24 v1Dinic | 0.03 |
| instance 100 24 v2Dinic | 0.039 |
| instance 100 24 v1Edmond | 0.168 |
| instance 100 24 v2Edmond | 0.171 |
| instance 100 24 v1Ford | 0.291 |
| instance 100 24 v2Ford | 0.926 |
| instance 100 25 v1Dinic | 0.029 |
| instance 100 25 v2Dinic | 0.037 |
| instance 100 25 v1Edmond | 0.164 |
| instance 100 25 v2Edmond | 0.172 |
| instance 100 25 v1Ford | 0.3 |
| instance 100 25 v2Ford | 0.927 |
| instance 100 26 v1Dinic | 0.031 |
| instance 100 26 v2Dinic | 0.039 |
| instance 100 26 v1Edmond | 0.185 |
| instance 100 26 v2Edmond | 0.19 |
| instance 100 26 v1Ford | 0.311 |
| instance 100 26 v2Ford | 1.074 |

| | |
|---|---|
| instance 100 27 v1Dinic | 0.029 |
| instance 100 27 v2Dinic | 0.036 |
| instance 100 27 v1Edmond | 0.17 |
| instance 100 27 v2Edmond | 0.18 |
| instance 100 27 v1Ford | 0.322 |
| instance 100 27 v2Ford | 1.015 |
| instance 100 28 v1Dinic | 0.029 |
| instance 100 28 v2Dinic | 0.034 |
| instance 100 28 v1Edmond | 0.174 |
| instance 100 28 v2Edmond | 0.177 |
| instance 100 28 v1Ford | 0.324 |
| instance 100 28 v2Ford | 1.022 |
| instance 100 29 v1Dinic | 0.029 |
| instance 100 29 v2Dinic | 0.032 |
| instance 100 29 v1Edmond | 0.166 |
| instance 100 29 v2Edmond | 0.175 |
| instance 100 29 v1Ford | 0.297 |
| instance 100 29 v2Ford | 0.967 |
| instance 100 3 v1Dinic | 0.096 |
| instance 100 3 v2Dinic | 0.113 |
| instance 100 3 v1Edmond | 0.399 |
| instance 100 3 v2Edmond | 0.428 |
| instance 100 3 v1Ford | 0.696 |
| instance 100 3 v2Ford | 1.764 |
| instance 100 30 v1Dinic | 0.027 |
| instance 100 30 v2Dinic | 0.034 |
| instance 100 30 v1Edmond | 0.162 |
| instance 100 30 v2Edmond | 0.167 |
| instance 100 30 v1Ford | 0.305 |
| instance 100 30 v2Ford | 0.966 |
| instance 100 4 v1Dinic | 0.082 |
| instance 100 4 v2Dinic | 0.097 |
| instance 100 4 v1Edmond | 0.326 |
| instance 100 4 v2Edmond | 0.357 |
| instance 100 4 v1Ford | 0.577 |
| instance 100 4 v2Ford | 1.59 |

| | |
|---|---|
| instance 100 5 v1Dinic | 0.075 |
| instance 100 5 v2Dinic | 0.086 |
| instance 100 5 v1Edmond | 0.309 |
| instance 100 5 v2Edmond | 0.303 |
| instance 100 5 v1Ford | 0.506 |
| instance 100 5 v2Ford | 1.451 |
| instance 100 6 v1Dinic | 0.064 |
| instance 100 6 v2Dinic | 0.075 |
| instance 100 6 v1Edmond | 0.282 |
| instance 100 6 v2Edmond | 0.299 |
| instance 100 6 v1Ford | 0.461 |
| instance 100 6 v2Ford | 1.412 |
| instance 100 7 v1Dinic | 0.065 |
| instance 100 7 v2Dinic | 0.07 |
| instance 100 7 v1Edmond | 0.273 |
| instance 100 7 v2Edmond | 0.295 |
| instance 100 7 v1Ford | 0.463 |
| instance 100 7 v2Ford | 1.456 |
| instance 100 8 v1Dinic | 0.057 |
| instance 100 8 v2Dinic | 0.067 |
| instance 100 8 v1Edmond | 0.246 |
| instance 100 8 v2Edmond | 0.261 |
| instance 100 8 v1Ford | 0.428 |
| instance 100 8 v2Ford | 1.409 |
| instance 100 9 v1Dinic | 0.059 |
| instance 100 9 v2Dinic | 0.07 |
| instance 100 9 v1Edmond | 0.256 |
| instance 100 9 v2Edmond | 0.28 |
| instance 100 9 v1Ford | 0.42 |
| instance 100 9 v2Ford | 1.472 |

# 5 Referencias bibliograficas


- Kleinberg and Tardos, 2005: Kleinberg, J. and Tardos, E. (2009), Algorithm Design.

  [online] Available at: `http://www.icsd.aegean.gr/kaporisa/index_files/Algorithm_Design.pdf` [Accessed 27/12/17].

- Diapositivas de clase, 2017. Available at: `http://www.cs.upc.edu/~mjserna/docencia/grauA/T17/MaxFlow-fib.pdf`

- Borodin, A. (2009). Dinic's max flow algorithm. Available at `http://www.cs.toronto.edu/~bor/375s06/dinic-sketch.pdf` [Accessed 29/12/17].

- Tarau, P. (2011). Dinic's algorithm. Available at `http://www.cse.unt.edu/~tarau/teaching/AnAlgo/Dinic's%20algorithm.pdf` [Accessed 29/12/17]. Imai, H. (1983). On the practical efficiency of various maximum flow algorithms. Available at: `http://www.orsj.or.jp/~archive/pdf/e_mag/Vol.26_01_061.pdf` [Accessed 29/12/17].