Max Flow - Min Cut: Examples

Final's Scheduling

We have as input:

- n courses, each one with a final. Each exam must be given in one room. Each course c_i has $E[i]$ students.
- r rooms. Each r_j has a capacity $S[j]$,
- τ time slots. For each room and time slot, we only can schedule one final.
- p professors to watch exams. Each exam needs one professor in each class and time. Each professor has its own restrictions of availability and no professor can oversee more than 6 finals. For each p_ℓ and τ_k define a Boolean variable $A[k, \ell] = T$ if p_ℓ is available at τ_k.

Design an efficient algorithm that correctly schedules a room, a time slot and a professor to every final, or report that not such schedule is possible.

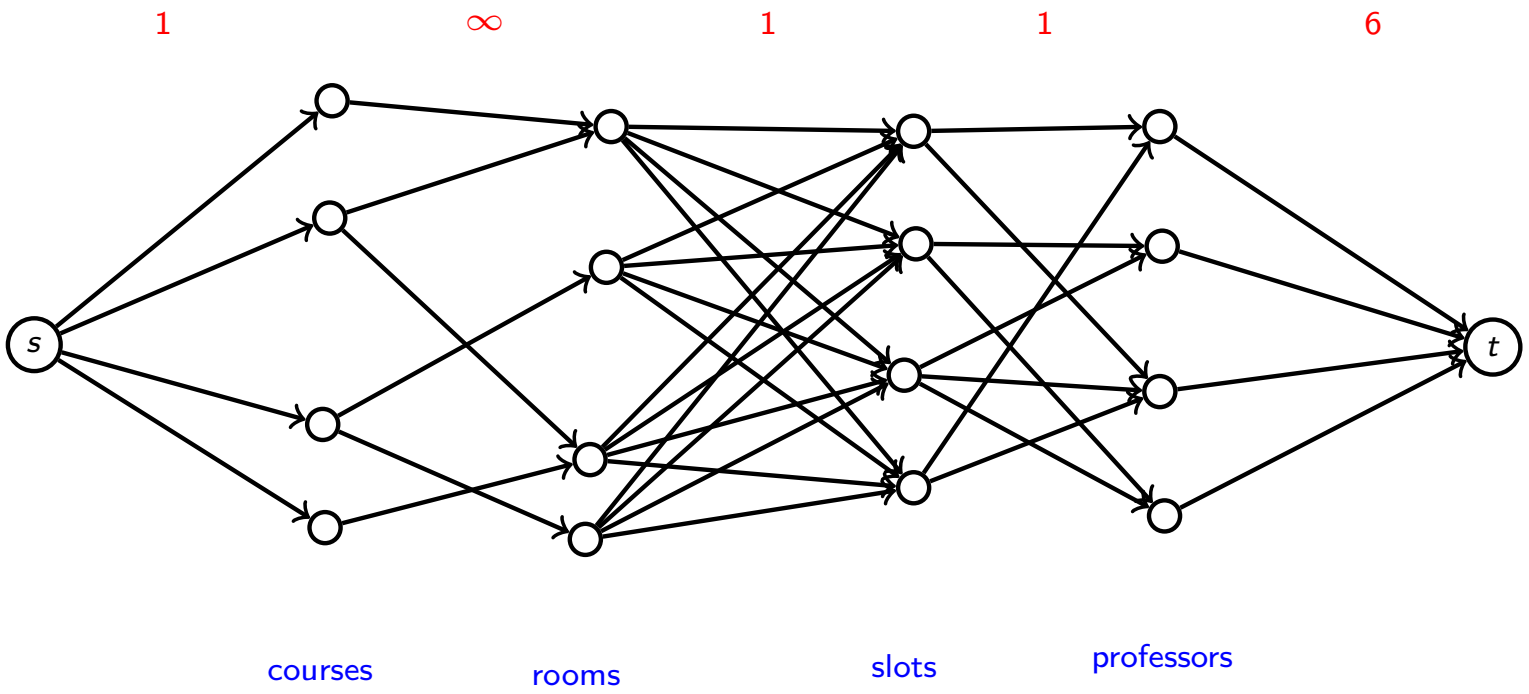
Construction of the network

Construct the network $\mathcal{N}$ with vertices $\{s, t, \{c_i\}, \{r_j\}, \{\tau_k\}, \{p_\ell\}\}$. Edges and capacities:

- (s, c_i) with capacity 1 (each course has one final)
- (c_i, r_j), if $E[i] \leq S[j]$, with capacity ∞ (can go to a room in which it fits)
- $\forall j, k$, (r_j, τ_k), with capacity 1 (one final per room and time slot).
- (τ_k, p_ℓ), if $A[k, \ell] = T$, capacity 1 (p can watch one final, if p is available at τ_k).
- (p_ℓ, t), capacity 6 (each p can watch ≤ 6 finals)

Notice that neither rooms nor time slots have individual restrictions.

Final's Scheduling: Flow Network



Final's Scheduling

- Notice the input size to the problem is $N = n + r + \tau + p + 2$. and size of the network is $O(N)$ vertices and $O(N^2)$ edges, why?
- Every path $s \rightsquigarrow t$ is an assignment of room-time-professor to a final, and any assignment room-time-professor to a final can be represented by a path $s \rightsquigarrow t$.
- Every integral flow identifies a collection of $|f|$ (s, t)-paths leading to a valid assignment for $|f|$ finals and viceversa.

Final's Scheduling

- To maximize the number of finals to be given, we compute the max-flow f^* from s to t.
- If $|f^*| = n$, then we can schedule all finals, otherwise we can not.
- To recover the assignment we have to consider the edges with positive flow and extract assignment from the n (s, t)-paths
- Complexity:
  - To construct $\mathcal{N}$, we need $O(N^2)$.
  - As $|f^*| \leq n$ integral, we can use Ford-Fulkerson to compute f^*, with cost $O(nN^2)$.
  - The second part requires $O(N^2)$ time.
  - So, the cost of the algorithm is $O(nN^2) = O(n(n + r + \tau + p)^2)$.

Matchings en grafos regulares

Un grafo bipartido $G = (V, E)$, dónde $V = L \cup R$, es d-regular si todo vértice tiene grado d.

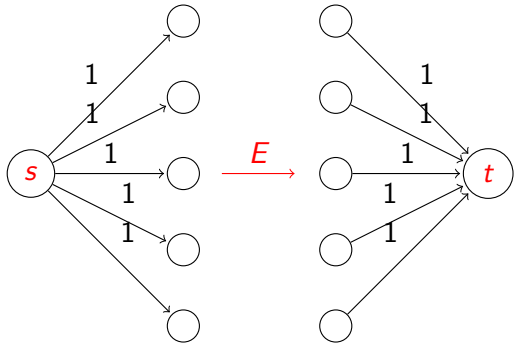
Demuestra que todo grafo bipartido d-regular tiene un matching the tamaño $|L|$. Siempre tiene un matching perfecto?

Ayuda: considerad la red de flujo correspondiente y analizar la capacidad del min cut.

Una solución

Si el grafo es bipartido y d-regular, $d|L| = |E|$ y $d|M| = |E|$. Por lo tanto $|L| = |R|$.

Como el grafo es bipartido podemos utilizar la formulación de maximum matching como problema de flujo y analizar la capacidad de los cortes.



En la red de flujo conectamos s con todas los vértices de L y todos los vértices de R con t. Luego orientamos todas las aristas de L hacia R. Asignamos capacidad 1 a todas las aristas.

s, t-cortes

Como $cut(\{s\}, V \cup \{t\}) = |L|$. Nos basta con demostrar que cualquier otro s, t-corte tiene capacidad $\geq |L|$.

- $cut(\{s\} \cup V, \{t\}) = |R| = |L|$
- $cut(\{s\} \cup L, R \cup \{t\}) = d|L| \geq |L|$

Nos queda analizar cortes que dividan L y R.

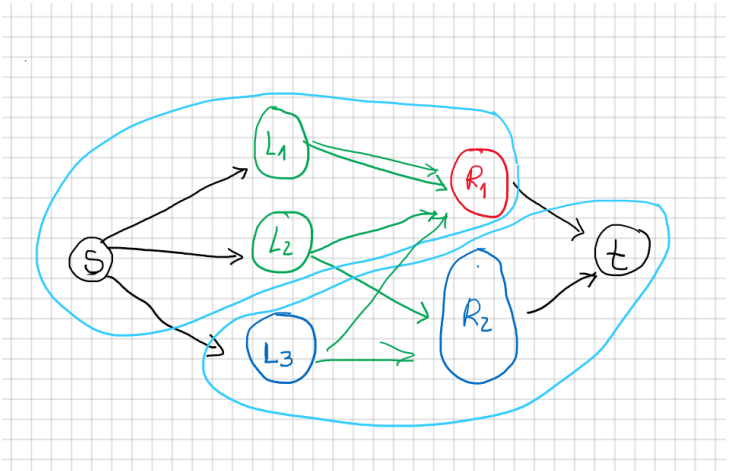
s, t-cortes

Consideremos un s, t-corte $(\{s\} \cup L_1 \cup L_2 \cup R_1, L_3 \cup R_2 \cup \{t\})$

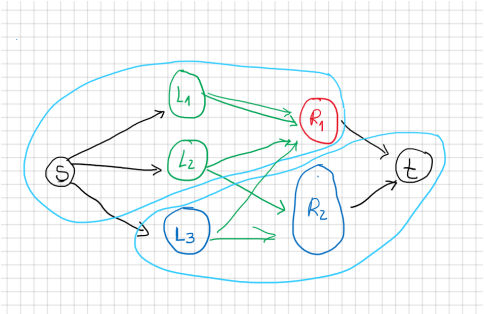
(L_1, L_2, L_3) es una partición de L y

(R_1, R_2) es una partición de R.

si $u \in L_1$, $N(u) \subseteq R_1$ y, si $u \in L_2$, $N(u) \cap R_2 \neq \emptyset$.



s, t-cortes



Como todos los vértices en L_2, R_1 tienen al menos un vecino en T y s está conectado con todos los vértices de L_3.

$$cut(S, T) = \geq |L_3| + |L_2| + |R_1|$$

- Todos los vecinos de vértices en L_1 están en R_1.
- El grafo es d-regular, $d|R_1| \geq d|L_1|$, y $|R_1| \geq |L_1|$.

$$cut(S, T) \geq |L_3| + |L_2| + |L_1| \geq |L|.$$

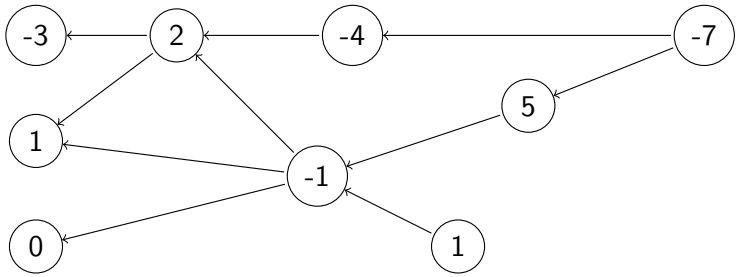
Siempre hay un matching de tamaño $|L|$ en un grafo bipartido d-regular y como $|L| = |R|$ este matching es perfecto.

Project selection

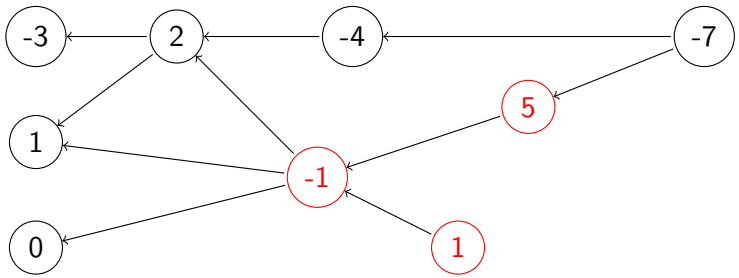
- We have a set of projects with prerequisites among them.
  - Set of possible projects P: project v has associated revenue p_v (positive or negative).
  - Set of prerequisites E: $(v, w) \in E$ means w is a prerequisite for v.
  - A subset of projects $A \subseteq P$ is feasible if the prerequisite of every project in A also belongs to A.
- Project selection problem. Given a set of projects P and prerequisites E, choose a feasible subset of projects to maximize revenue.

We can see (P, E) as a directed graph.

Project selection: feasibility

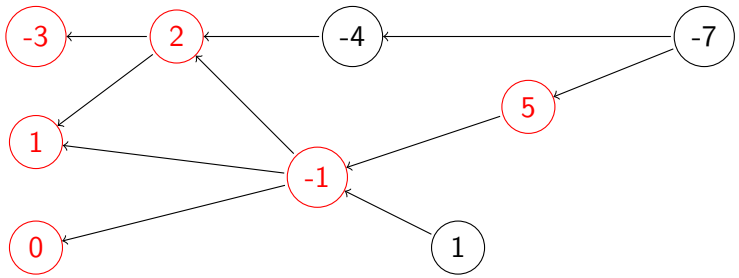


Project selection: feasibility



A is unfeasible

Project selection: feasibility



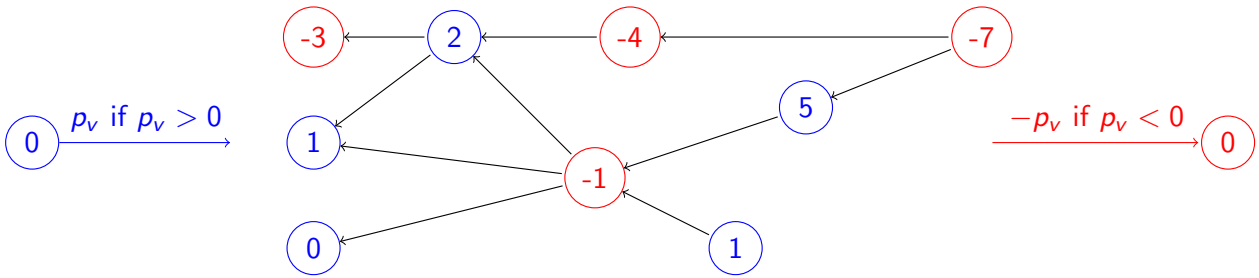
A is feasible and has revenue 4.

Project selection: flow network

To construct $\mathcal{N}$ we start with (P, E) and

- Add vertices s and t to (P, E)
  - Assign a capacity of ∞ to each prerequisite edge (in E).
  - Add edge (s, v) with capacity p_v if $p_v > 0$.
  - Add edge (v, t) with capacity $-p_v$ if $p_v < 0$.
  - For convenience, define $p_s = p_t = 0$.

Project selection: flow network $\mathcal{N}$ scheme



Black edges have capacity ∞

Labels inside nodes are the p values

Project selection: min-cut equivalence

Claim

Let A be a set of projects and $B = P \setminus A$.
A is feasible iff $cut(A + s, B + t) < \infty$.

- If A is feasible there are no edges in E connecting a project in A with a project in B. So the cut an only contain edges from s or to T but all of the have capacity $< \infty$.
- If $cut(A + s, B + t) < \infty$, there are no edges in E accross the cut, therefore all the prerequisites of projects in A belong to A. Therefore A is feasible.

Project selection: min-cut equivalence

Claim

(A, B) is a min cut iff $A - \{s\}$ is an optimal feasible set of projects

- In $\mathcal{N}$, $cut(\{s\}, P + t) < \infty$. So, a min cut has finite capacity. By the previous claim, $A - \{s\}$ is feasible.
- To see that it is an optimal solution let us analyze the capacity of a generic (s, t)-cut.
  Note that, as there are no edges from E across the cut

$$cut(A, B) = \sum_{v \in B : p_v > 0} p_v + \sum_{v \in A : p_v < 0} (-p_v)$$

Project selection min-cut equivalence

Let's rewrite the expression for the min-cut capacity:

$$
\begin{aligned}
cut(A,B) &= \sum_{v \in B : p_v > 0} p_v + \sum_{v \in A : p_v < 0} (-p_v) \\
&= \sum_{v \in B : p_v > 0} p_v + \sum_{v \in A : p_v > 0} p_v - \sum_{v \in A : p_v > 0} p_v - \sum_{v \in A : p_v < 0} p_v \\
&= \sum_{v \in P : p_v > 0} p_v - \sum_{v \in A} p_v
\end{aligned}
$$

- The first term does not depend on the cut, only of the input.
- Minimizing $cut(A,B)$ is the same as maximizing $\sum_{v \in A} p_v$, i.e, maximizing the revenue.

Doctors on Call

Per a cadascun dels pròxims n dies, DoC ha determinat el nombre de doctors disponibles que requereix.
Així al dia i-èsim, necessiten exactament p_i doctors.
Hi han k doctors en total, i cadascú d'ells ha donat una llista amb els dies en que està disposat a treballar. Així el doctor j proporciona un conjunt L_j de dies.

Doctors on Call

DoC vol, a partir d'aquesta informació, un procediment que permeti tornar a cada doctor j una llista definitiva de dies L'_j amb les propietats següents:

(1) el conjunto $\Delta_j = L'_j \setminus L_j$ té com a molt c dies; i

(2) quan es considera tot el conjunto de listas $L'_1, \ldots, L'_k$, per a cada dia $1 \leq i \leq n$, hi han exactament p_i doctors que tenen el dia i a la seva llista definitiva.

El paràmetre c reflecteix la tolerància de l'assignació i pot variar segons las circumstancies. Per suposat, si tal solució no es possible, el sistema ha de (correctament) informar de que aquest és el cas.

Doctors on Call

1. Proporcioneu un algorisme de cost polinòmic en n i k que resolgui el problema per un valor de tolerància c donat.
2. Proporcioneu un algorisme de cost polinòmic que obtingui la tolerància mínima per que el problema tingui solució, assumint que $p_i \leq k$ per $1 \leq i \leq n$.

Doctors on Call -1

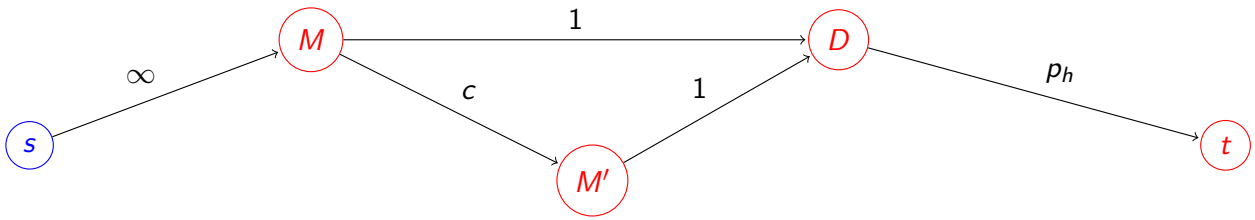
Lo plantearemos como un problema de asignación con restricciones en una red de flujo en la que una unidad de flujo de s a t represente la asignación de un médico a un día.

La red $\mathcal{N}$ tiene

- Nodos: s, t, M, M', D, $|M| = k$ representa médicos, $|D| =$n representa los días, $|M'| = k$ copia de M
- Aristas y capacidades:

$$\begin{aligned} \{(s,u) \mid u \in M\} &\quad \text{capacidad } \infty \\ \{(h,t) \mid h \in H\} &\quad \text{capacidad } p_h \\ \{(u,d) \mid u \in M, d \in L_u\} &\quad \text{capacidad 1} \\ \{(u,u') \mid u \in M\} &\quad \text{capacidad } c \\ \{(u',d) \mid d \notin L_{u'}\} &\quad \text{capacidad 1} \end{aligned}$$

DoC: Red schema



Validez de la reducción

- Un camino de s a t tiene la forma $s \to m \to d \to t$ o $s \to m \to m' \to d \to t$, si transporta una unidad de flujo, lo interpretaremos como: el médico m tendrá h en su lista final.
- Las conexiones con capacidad 1 garantizan que nunca se asigna a un doctor más de una guardia un día.
- Las conexiones con capacidad c garantizan que nunca se asigna un doctor más de c días que no están en su lista.
- El posible flujo que llega a d, total de médicos asignados a ese día se transmite por (d, t), la capacidad garantiza que el flujo no supera el tope.

Así, el problema tiene solución si y solo si el valor del MaxFlow en $\mathcal{N}$ es $\sum_i p_i$.

Algoritmo

Construir $\mathcal{N}$
$F = MaxFlow(\mathcal{N})$
if $|F| < \sum_i p_i$ **then**
  return NO
else
  return $\{(u, d) \mid F[u, d] = 1, u \in M \cup M', d \in D\}$

Coste

El coste de la llamada a MaxFlow: El número de vértices en la red es $N = n + 2k + 2$, el número de aristas es $M = n + 2k + nk$.

MaxFlow está acotado por $\sum_i p_i$ o por nk

- Con FF el coste es $O(nk(N + M)) = O(n^2k^2)$
- Con EK el coste es $O(M^2N) = O(n^2k^2(n + k))$

Utilizaremos el coste de FF.

Construir $\mathcal{N}$ tiene coste $O(N + M)$ y el último paso $O(nk)$. El coste total del algoritmo es $O(n^2k^2)$.

Doctors on Call - 2

- Si $c = 0$ el problema no tiene solución.
- S $c = n$ el problema tiene solución.
- para buscar el valor mínimo de c podemos hacer una búsqueda binaria utilizando el algoritmo del paso anterior.
- En el caso peor hacemos $O(logn)$ iteraciones en la búsqueda y el coste del algoritmo es tred$O(n^2 k^2 \log n)$.