Flujo

Modelado de Problemas con Max Flow / Min Cut

Lautaro Lasorsa

Universidad de Buenos Aires - FCEN || Universidad Nacional de La Matanza - DIIT

Training Camp 2025

Gracias Sponsors!

Organizador

Diamond Plus

Gracias Sponsors!

Platino

FOLDER IT

INTERNATIONAL
SOFTWARE COMPANY

Gold

NeuralSoft

Oro

JS JERÁRQUICOS

Gracias Sponsors!

Aliado

Outline

Outline

Definición de una red de flujo

Dado un grafo dirigido $G = (V, E)$ con capacidades $c : E \to N$, y dos nodos $S, T \in V$:

- Un flujo $f : E \to N$ es una función que asigna un valor a cada arista, cumpliendo las siguientes condiciones:
  - Capacidad: $0 \leq f(e) \leq c(e)$ para todo $e \in E$.
  - Conservación: Para todo nodo $v \in V - \{S, T\}$ el flujo de entrada y el flujo de salida son iguales:
    - Flujo de entrada: $\sum_{u:(u,v)\in E} f(u, v)$.
    - Flujo de salida: $\sum_{w:(v,w)\in E} f(v, w)$.

Definición de una red de flujo

Dado un grafo dirigido $G = (V, E)$ con capacidades $c : E \to N$, y dos nodos $S, T \in V$:

- Un flujo $f : E \to N$ es una función que asigna un valor a cada arista, cumpliendo las siguientes condiciones:
  - Capacidad: $0 \leq f(e) \leq c(e)$ para todo $e \in E$.
  - Conservación: Para todo nodo $v \in V - \{S, T\}$ el flujo de entrada y el flujo de salida son iguales:
    - Flujo de entrada: $\sum_{u:(u,v)\in E} f(u, v)$.
    - Flujo de salida: $\sum_{w:(v,w)\in E} f(v, w)$.
- El valor del flujo se define como $|f| = \sum_{(S,v)\in E} f(S, v) = \sum_{(v,T)\in E} f(v, T)$.

Definición del problema de flujo máximo

Dada una red de flujo, consiste en elegir un flujo f que maximice su valor $|f|$, cumpliendo las condiciones de capacidad y conservación.

Theorem

Si todas las capacidades son enteras, entonces existe un flujo máximo f tal que $|f|$ es un entero y $f(e)$ es un entero para todo $e \in E$.

Definición del problema de flujo máximo

Dada una red de flujo, consiste en elegir un flujo f que maximice su valor $|f|$, cumpliendo las condiciones de capacidad y conservación.

Theorem

Si todas las capacidades son enteras, entonces existe un flujo máximo f tal que $|f|$ es un entero y $f(e)$ es un entero para todo $e \in E$.

Lo importante, que usaremos para modelado, es que podemos pensar cada unidad de flujo como un cámino que va desde S hasta T, de tal forma que por una arista e no pasan más de $c(e)$ cáminos distintos.
Esto puede verse, por ejemplo, en el problema Distinct Routes (1711) de CSES.

Red residual

Definition (Red residual)

Dado un grafo $G = (V, E)$ con funciones de capacidad c y de flujo f, definimos la red residual $R = (V', E')$ del grafo como:

- Un grafo con el mismo conjunto de nodos V. $V' = V$.
- E' tiene todas las aristas de E con sus mismas capacidades y flujos.
- Para cada arista $e = (u, v) \in E$, existe una arista $e' = (v, u) \in E'$ con $c(e') = 0$ y $f(e') = -f(e)$.

Red residual

Definition (Red residual)

Dado un grafo $G = (V, E)$ con funciones de capacidad c y de flujo f, definimos la red residual $R = (V', E')$ del grafo como:

- Un grafo con el mismo conjunto de nodos V. $V' = V$.
- E' tiene todas las aristas de E con sus mismas capacidades y flujos.
- Para cada arista $e = (u, v) \in E$, existe una arista $e' = (v, u) \in E'$ con $c(e') = 0$ y $f(e') = -f(e)$.

Theorem

Existe un cámino C en R desde S hasta T tal que para cada arista $e \in C$, $f(e) < c(e)$ si y solo si el flujo f no es máximo.
Este cámino C se llama cámino aumentante y enviar una unidad de flujo por C (aumentar en 1 flujo de todas las aristas de C y reducir en 1 el flujo de las aristas inversas) aumenta el valor del flujo $|f|$ en 1.

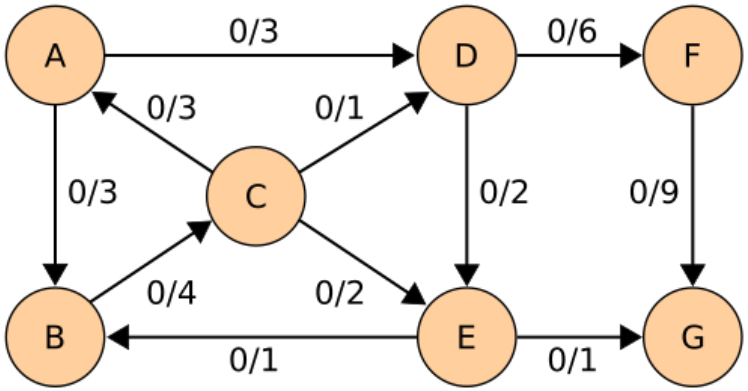
Si una arista e tiene $f(e) = c(e)$, se dice que está saturada.

Soluciones al problema de flujo máximo

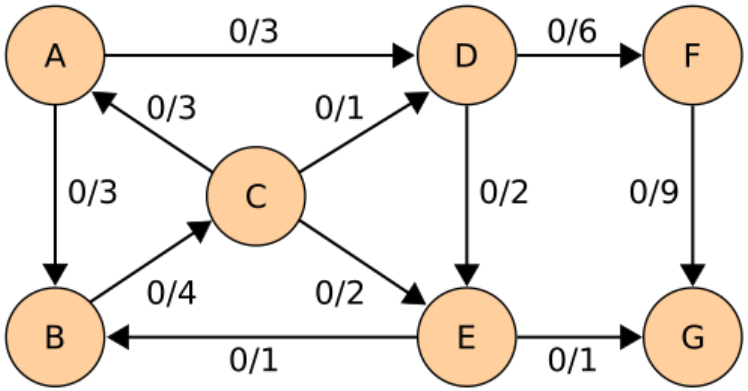
- Ford Fulkerson - Edmonds Karp: En cada iteración busca un cámino aumentante en la red residual utilizando BFS y lo utiliza. Termina cuando no hay más cáminos aumentantes. Es $O(VE^2)$.
- Dinic: Pueden ver los detalles en CP Algorithms. Tiene complejidad general $O(V^2E)$, pero en casos interesantes como redes unitarias o planares es $O(E\sqrt{V})$.
- Programación lineal: Simplex, excesivamente complejo pero interesante desde el punto de vista teórico.

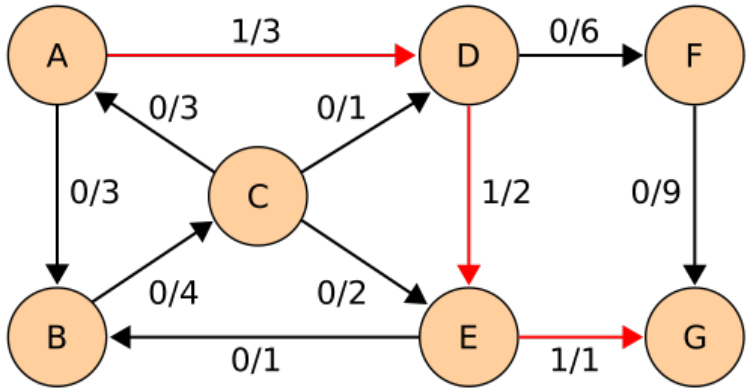
Todos los algoritmos encuentran un flujo máximo f y no solo el valor máximo posible.

Ejemplo: Ejecución de Ford Fulkerson Edmon Karp

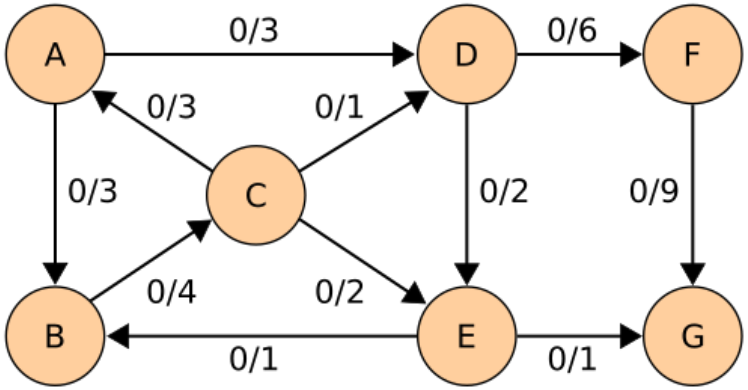


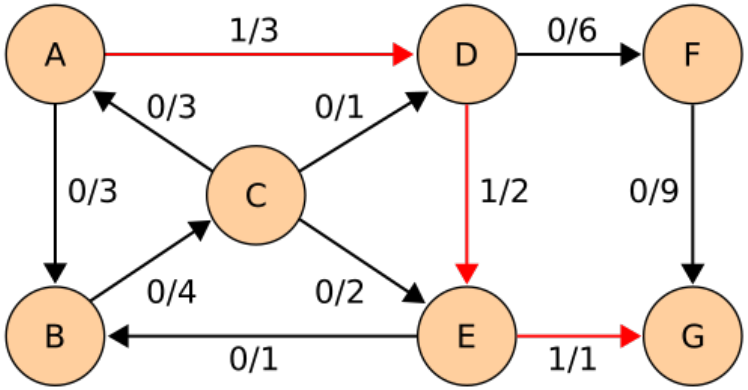
Ejemplo: Ejecución de Ford Fulkerson Edmon Karp

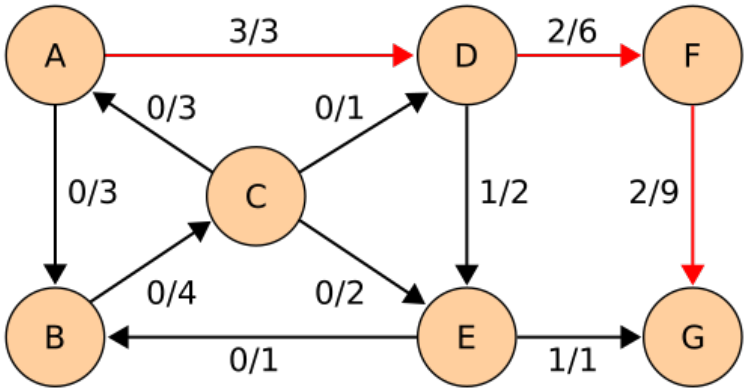




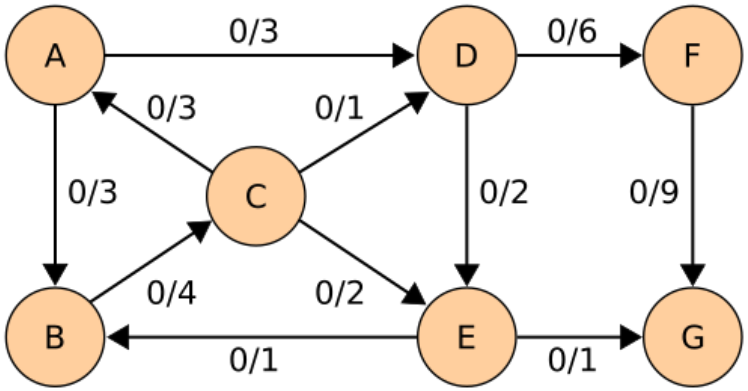
Ejemplo: Ejecución de Ford Fulkerson Edmon Karp

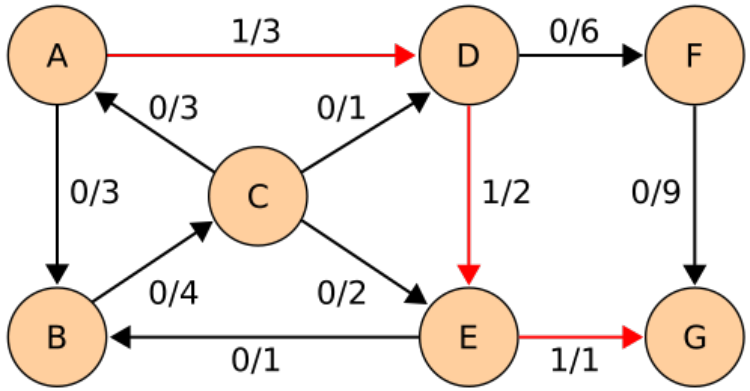


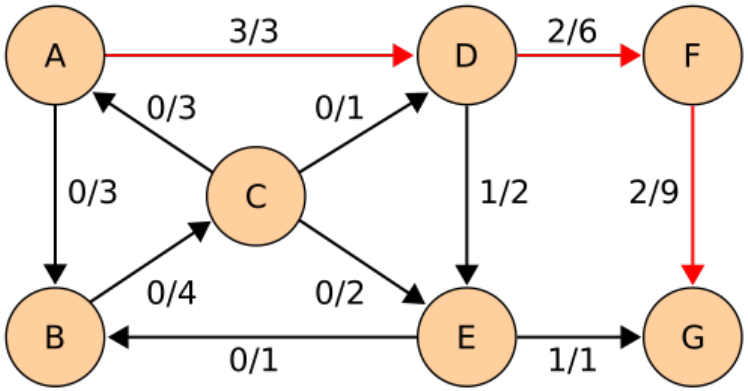


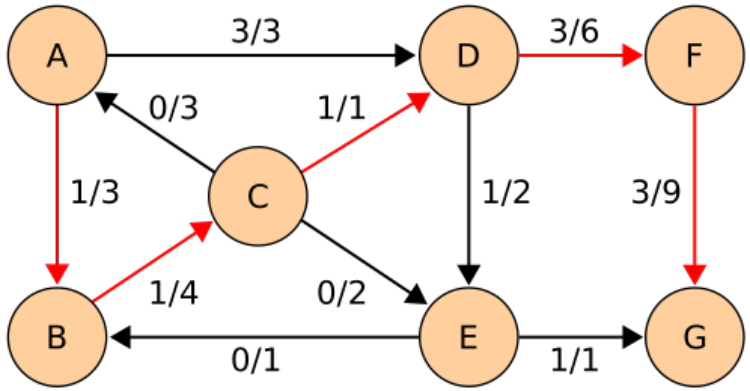


Ejemplo: Ejecución de Ford Fulkerson Edmon Karp

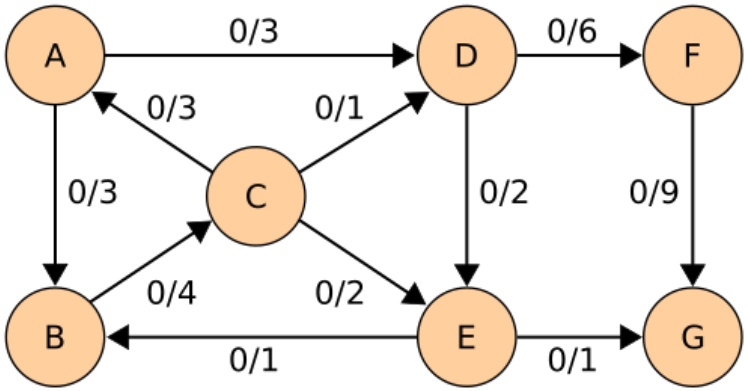


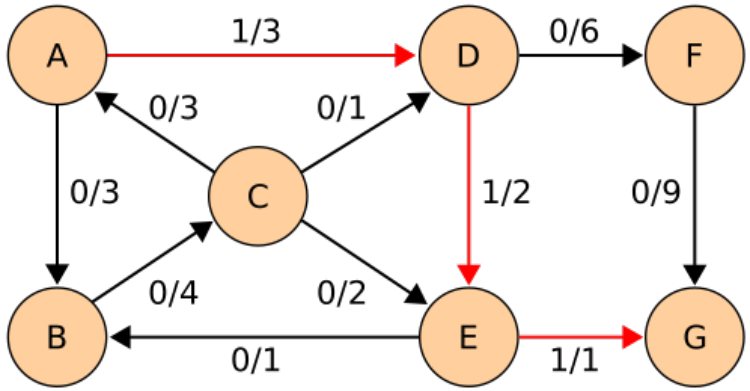


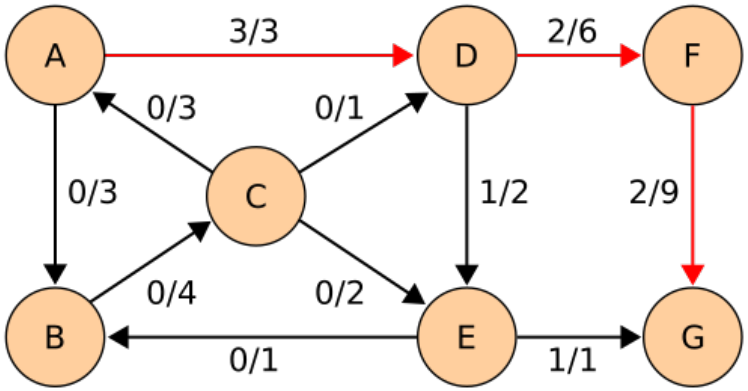


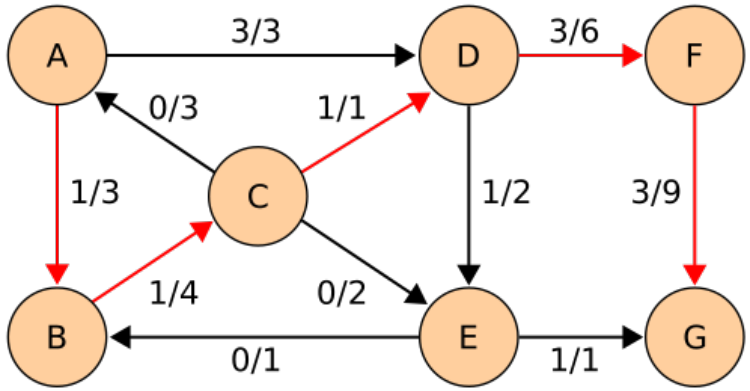


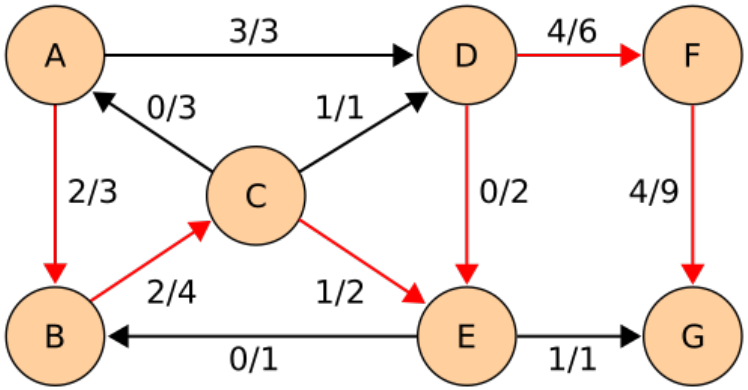
Ejemplo: Ejecución de Ford Fulkerson Edmon Karp











Definición de corte

Sea un grafo dirigido ponderado $G = (V, E)$ y dos nodos $S, T \in V$, hay 2 definiciones posibles de corte:

1. Un conjunto de aristas $E' \subseteq E$ tal que si eliminamos todas las aristas de E' del grafo, no hay un cámino desde S hasta T. En este caso, el costo del corte es $c(E') = \sum_{e \in E'} c(e)$.
2. Una partición de los nodos $V = V_1 \cup V_2$ tal que $S \in V_1$ y $T \in V_2$. En este caso, el costo del corte es $c(V_1, V_2) = \sum_{(u,v) \in E, u \in V_1, v \in V_2} c(u, v)$.

Problema de Corte Mínimo

Dado un grafo dirigido ponderado $G = (V, E)$ y un par de nodos $S, T \in V$, el problema de corte mínimo consiste en encontrar un corte de costo mínimo entre S y T.

Teorema de Max Flow - Min Cut

Theorem (Max Flow - Min Cut)

Dado un grafo dirigido ponderado $G = (V, E)$ y un par de nodos $S, T \in V$, el valor del flujo máximo desde S hasta T es igual al costo del corte mínimo entre S y T. (los costos de las aristas se consideran como capacidades)

Teorema de Max Flow - Min Cut

Theorem (Max Flow - Min Cut)

Dado un grafo dirigido ponderado $G = (V, E)$ y un par de nodos $S, T \in V$, el valor del flujo máximo desde S hasta T es igual al costo del corte mínimo entre S y T. (los costos de las aristas se consideran como capacidades)

Theorem

Sea f un flujo máximo en G y R su red residual. Entonces, el conjunto de nodos alcanzables, sin pasar por aristas saturadas, desde S en R forma un lado del corte mínimo, y el conjunto de nodos no alcanzables desde S forma el otro lado del corte mínimo.

Costo de un flujo

Dado un grafo dirigido $G = (V, E)$, dos nodos $S, T \in V$, y las siguientes funciones:

- Capacidad $c : E \to N$.
- Costo $w : E \to R_{\geq 0}$.

Definimos el costo de un flujo f como:

$$C(f) = \sum_{e \in E} w(e) f(e)$$

Problema de Min Cost Max Flow

El problema consiste en encontrar un flujo f de valor máximo, y entre los flujos de valor máximo, el que tenga mínimo costo.
Se resuelve mediante el algoritmo de Edmonds-Karp, pero en lugar de usar BFS, se utiliza un algoritmo de cámino mínimo en grafo ponderado.
Aunque Bellman-Ford es una opción valida, se puede utilizar Dijkstra utilizando una función de potencial para evitar aristas negativas, permitiendo una complejidad de $O(|E||f|\log(|V|))$.

Outline

Definición de Matching

- Sea un grafo $G = (V, E)$, un matching es un conjunto de aristas $M \subseteq E$ tal que no hay dos aristas en M que compartan un nodo. Es decir, cada nodo está emparejado con a lo sumo una arista en M.
- El problema de matching máximo consiste en encontrar un matching M de tamaño (cantidad de aristas) máximo.
- En el caso general se puede resolver con el Algoritmo de Blossom en $O(|E| * |V|^2)$ y en $O(|E| * |V|^{1/2})$ con el algoritmo de Micali y Vazirani, pero este es demasiado complicado para usarse en programación competitiva.
- Para el caso particular de grafos bipartitos, puede resolver en $O(|E| * \sqrt{|V|})$ utilizando flujo máximo.

Matching Máximo Bipartito como problema de flujo máximo

Sea un grafo bipartito $G = (U \cup V, E)$, donde U y V son los conjuntos de nodos. Definimos el grafo de flujo máximo $G' = (V', E')$ de la siguiente manera:

- Agregamos un nodo fuente S y un nodo sumidero T.
- Para cada nodo $u \in U$, agregamos una arista (S, u) con capacidad 1.
- Para cada nodo $v \in V$, agregamos una arista (v, T) con capacidad 1.
- Para cada arista $(u, v) \in E$, agregamos una arista (u, v) con capacidad ∞.

El matching máximo en el grafo original corresponde al flujo máximo en el grafo de flujo máximo, y viceversa.
A su vez, podemos recuperar el matching máximo a partir de la función de flujo f. Para cada arista $(u, v) \in E$, pertenece al matching máximo si y solo si $f(u, v) = 1$.

Propiedades del Matching Máximo Bipartito

Sea $G = (V, E)$, con $V = V_1 \bigcup V_2$, un grafo bipartito y M un matching máximo en G. Entonces:

- El Minimum Vertex Cover (MVC) de G es el mínimo conjunto de nodos tal que cada arista es incidente en al menos un nodo del conjunto.

Propiedades del Matching Máximo Bipartito

Sea $G = (V, E)$, con $V = V_1 \bigcup V_2$, un grafo bipartito y M un matching máximo en G. Entonces:

- El Minimum Vertex Cover (MVC) de G es el mínimo conjunto de nodos tal que cada arista es incidente en al menos un nodo del conjunto.
- Teorema de Konig: $|M|$ es el tamaño del MVC de G.

Propiedades del Matching Máximo Bipartito

Sea $G = (V, E)$, con $V = V_1 \bigcup V_2$, un grafo bipartito y M un matching máximo en G. Entonces:

- El Minimum Vertex Cover (MVC) de G es el mínimo conjunto de nodos tal que cada arista es incidente en al menos un nodo del conjunto.
- Teorema de Konig: $|M|$ es el tamaño del MVC de G.
- El MVC está compuesto por los nodos de V_2 alcanzables desde S y los nodos de V_1 que alcanzan a T en el resultado de eliminar las aristas de un min cut del grafo original.

Propiedades del Matching Máximo Bipartito

Sea $G = (V, E)$, con $V = V_1 \bigcup V_2$, un grafo bipartito y M un matching máximo en G. Entonces:

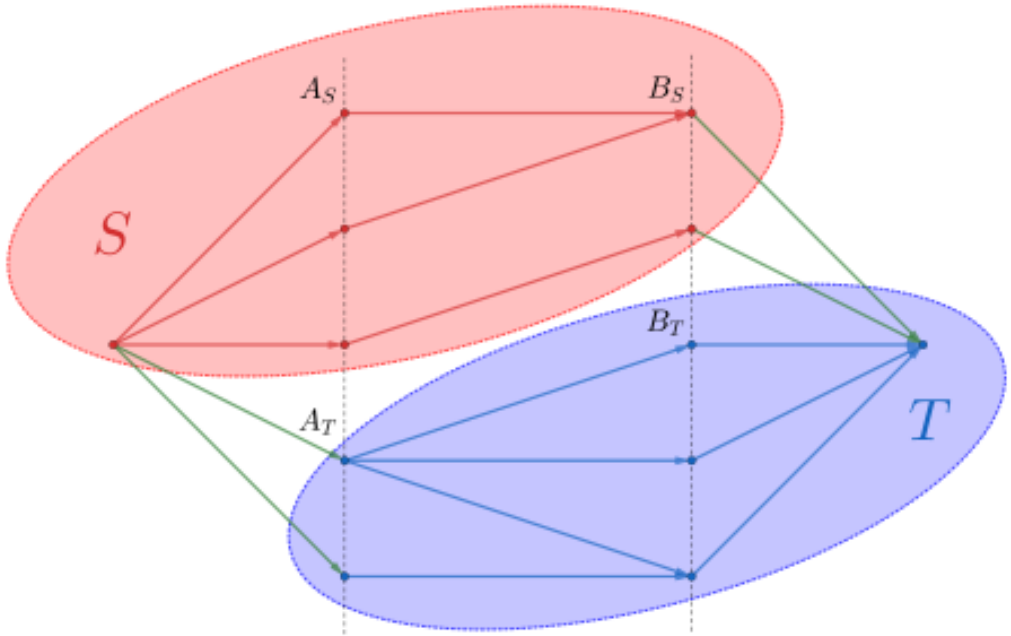
- El Minimum Vertex Cover (MVC) de G es el mínimo conjunto de nodos tal que cada arista es incidente en al menos un nodo del conjunto.
- Teorema de Konig: $|M|$ es el tamaño del MVC de G.
- El MVC está compuesto por los nodos de V_2 alcanzables desde S y los nodos de V_1 que alcanzan a T en el resultado de eliminar las aristas de un min cut del grafo original.
- En general, $V - MVC = MIS$, donde MIS es el Maximum Independent Set.

Propiedades del Matching Máximo Bipartito

Sea $G = (V, E)$, con $V = V_1 \bigcup V_2$, un grafo bipartito y M un matching máximo en G. Entonces:

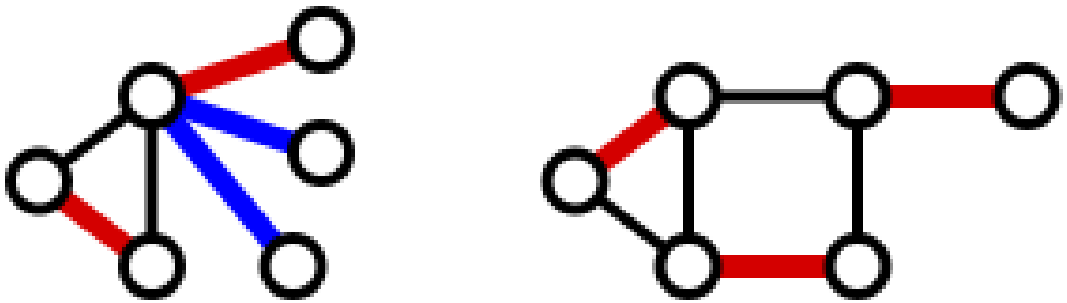
- El Minimum Vertex Cover (MVC) de G es el mínimo conjunto de nodos tal que cada arista es incidente en al menos un nodo del conjunto.
- Teorema de Konig: $|M|$ es el tamaño del MVC de G.
- El MVC está compuesto por los nodos de V_2 alcanzables desde S y los nodos de V_1 que alcanzan a T en el resultado de eliminar las aristas de un min cut del grafo original.
- En general, $V - MVC = MIS$, donde MIS es el Maximum Independent Set.
- En general, puedo construir un Minimum Edge Cover (MEC) utilizando las aristas del Matching Máximo y agregando una arista cualquiera incidente en cada nodo no cubierto por le matching.
- $|MEC| = |V| - |M|$.

Minimum Vertex Cover



Minimum Edge Cover

La aristas rojas son las pertenecientes al matching máximo y las azules las que hay que agregar



Matching Máximo Bipartito Ponderado

Si al grafo bipartito G le agregamos pesos en las aristas, podemos definir el peso de un matching como la suma de los pesos de las aristas utilizadas en él.
Entonces, podemos buscar, entre los matchings de tamaño máximo, aquel que maximice o minimice el peso total.
Esto puede alcanzarse de 2 formas:

1. Agregando el peso de las aristas a la red de flujo y utilizando Min Cost Max Flow en $O(|E| * |f|)$
2. Algoritmo Hungaro: $O(|V|^3)$, con mejor constante que el algoritmo de flujo.

Outline

Partición de un DAG en caminos

Definition

Sea un DAG $G = (V, E)$, podemos buscar una partición de sus nodos en caminos disjuntos. Es decir, buscar caminos $C_1, C-2, ..., C_k \in G$ tales que $V = \bigcup_{i=1}^{k} V(C_i)$ y $C_i \cap C_j = \emptyset$ para todo $i \neq j$.

También podemos buscar una partición no necesariamente disjunta. Esta va a ser equivalente a una partición en caminos disjuntos de la clausura transitiva de G, y es así como la vamos a calcular.
Expresar V como unión de caminos no necesariamente disjuntos también se llama una covertura.
El problema consiste en buscar una partición en caminos de tamaño mínimo, es decir, con la mínima cantidad de caminos.

Reducción a un problema de Matching Máximo

Sea un DAG $G = (V, E)$, para encontrar su mínima partición en caminos disjuntos, podemos reducirlo al problema de Matching Máximo Bipartito:

1. Defino un nuevo grafo bipartito $G' = (V', E')$
2. Para cada nodo $v \in V$, agrego dos nodos $v_{in}, v_{out} \in V'$.

Reducción a un problema de Matching Máximo

Sea un DAG $G = (V, E)$, para encontrar su mínima partición en caminos disjuntos, podemos reducirlo al problema de Matching Máximo Bipartito:

1. Defino un nuevo grafo bipartito $G' = (V', E')$
2. Para cada nodo $v \in V$, agrego dos nodos $v_{in}, v_{out} \in V'$.
3. Para cada arista $(u, v) \in E$, agrego una arista $(u_{out}, v_{in}) \in E'$.

Reducción a un problema de Matching Máximo

Sea un DAG $G = (V, E)$, para encontrar su mínima partición en caminos disjuntos, podemos reducirlo al problema de Matching Máximo Bipartito:

1. Defino un nuevo grafo bipartito $G' = (V', E')$
2. Para cada nodo $v \in V$, agrego dos nodos $v_{in}, v_{out} \in V'$.
3. Para cada arista $(u, v) \in E$, agrego una arista $(u_{out}, v_{in}) \in E'$.
4. Calculo el Matching Máximo Bipartito M en G'.
5. El tamaño de la mínima partición en caminos será $|V| - |M|$.

Reducción a un problema de Matching Máximo

Sea un DAG $G = (V, E)$, para encontrar su mínima partición en caminos disjuntos, podemos reducirlo al problema de Matching Máximo Bipartito:

1. Defino un nuevo grafo bipartito $G' = (V', E')$
2. Para cada nodo $v \in V$, agrego dos nodos $v_{in}, v_{out} \in V'$.
3. Para cada arista $(u, v) \in E$, agrego una arista $(u_{out}, v_{in}) \in E'$.
4. Calculo el Matching Máximo Bipartito M en G'.
5. El tamaño de la mínima partición en caminos será $|V| - |M|$.
6. Las aristas que se usan en los caminos de la partición son las aristas del matching.
7. Para saber donde empiezan los caminos, tomamos los v tales que v_{in} no está matcheado.

Teorema de Dilworth

Definition

Anticadena Sea un DAG $G = (V, E)$, una anticadena es $V' \subseteq V$ tal que no hay dos nodos $u, v \in V'$ tales que u puede alcanzar a v en G.

Teorema de Dilworth

Definition

Anticadena Sea un DAG $G = (V, E)$, una anticadena es $V' \subseteq V$ tal que no hay dos nodos $u, v \in V'$ tales que u puede alcanzar a v en G.

Theorem (Dilworth)

Sea un DAG $G = (V, E)$ y C su mínima covertura en cáminos no disjuntos, el tamaño de la máxima anticadena A es $|C|$.

Teorema de Dilworth: Reconstrucción

Para reconstruir la máxima anticadena A de un DAG $G = (V, E)$, podemos:

1. Construimos la clausura transitiva de G, $G' = (V, E')$.

Teorema de Dilworth: Reconstrucción

Para reconstruir la máxima anticadena A de un DAG $G = (V, E)$, podemos:

1. Construimos la clausura transitiva de G, $G' = (V, E')$.
2. Definimos el grafo bipartito $B = (V_B, E_B)$, donde $V_B = \{v_{in}, v_{out} : v \in V\}$ y $E_B = \{(u_{out}, v_{in}) : (u, v) \in E'\}$.

Teorema de Dilworth: Reconstrucción

Para reconstruir la máxima anticadena A de un DAG $G = (V, E)$, podemos:

1. Construimos la clausura transitiva de G, $G' = (V, E')$.
2. Definimos el grafo bipartito $B = (V_B, E_B)$, donde $V_B = \{v_{in}, v_{out} : v \in V\}$ y $E_B = \{(u_{out}, v_{in}) : (u, v) \in E'\}$.
3. Calculamos un MVC M de B.

Teorema de Dilworth: Reconstrucción

Para reconstruir la máxima anticadena A de un DAG $G = (V, E)$, podemos:

1. Construimos la clausura transitiva de G, $G' = (V, E')$.
2. Definimos el grafo bipartito $B = (V_B, E_B)$, donde $V_B = \{v_{in}, v_{out} : v \in V\}$ y $E_B = \{(u_{out}, v_{in}) : (u, v) \in E'\}$.
3. Calculamos un MVC M de B.
4. Una máxima anticadena A de G es el conjunto de nodos $v \in V$ tales que $v_{in} \notin M \wedge v_{out} \notin M$

Outline

Problema de Asignación

Sea un conjunto T de tareas y un conjunto M de máquinas necesarias para realizar la tarea:

Problema de Asignación

Sea un conjunto T de tareas y un conjunto M de máquinas necesarias para realizar la tarea:

- Para comprar la máquina $m \in M$ necesitamos pagar un costo $c(m)$.

Problema de Asignación

Sea un conjunto T de tareas y un conjunto M de máquinas necesarias para realizar la tarea:

- Para comprar la máquina $m \in M$ necesitamos pagar un costo $c(m)$.
- Para realizar la tarea $t \in T$ necesitamos un conjunto $M_t \subseteq M$ de máquinas.

Problema de Asignación

Sea un conjunto T de tareas y un conjunto M de máquinas necesarias para realizar la tarea:

- Para comprar la máquina $m \in M$ necesitamos pagar un costo $c(m)$.
- Para realizar la tarea $t \in T$ necesitamos un conjunto $M_t \subseteq M$ de máquinas.
- Puedo utilizar una misma máquina para realizar cualquier cantidad de tareas, pero no puedo realizar una tarea si no tengo todas las máquinas necesarias.

Problema de Asignación

Sea un conjunto T de tareas y un conjunto M de máquinas necesarias para realizar la tarea:

- Para comprar la máquina $m \in M$ necesitamos pagar un costo $c(m)$.
- Para realizar la tarea $t \in T$ necesitamos un conjunto $M_t \subseteq M$ de máquinas.
- Puedo utilizar una misma máquina para realizar cualquier cantidad de tareas, pero no puedo realizar una tarea si no tengo todas las máquinas necesarias.
- Si realizo la tarea t obtengo un beneficio $b(t)$.

Problema de Asignación

Sea un conjunto T de tareas y un conjunto M de máquinas necesarias para realizar la tarea:

- Para comprar la máquina $m \in M$ necesitamos pagar un costo $c(m)$.
- Para realizar la tarea $t \in T$ necesitamos un conjunto $M_t \subseteq M$ de máquinas.
- Puedo utilizar una misma máquina para realizar cualquier cantidad de tareas, pero no puedo realizar una tarea si no tengo todas las máquinas necesarias.
- Si realizo la tarea t obtengo un beneficio $b(t)$.

Quiero elegir un subconjunto de máquinas M' a comprar y uno de tareas T' a realizar, de tal forma que maximicemos el beneficio total neto.

$$\max_{M' \subseteq M, T' \subseteq T, T' \text{ es posible dado } M'} \left(\sum_{t \in T'} b(t) - \sum_{m \in M'} c(m) \right)$$

Reducción al problema de corte mínimo

Para resolver este problema, lo podemos reducir al problema de corte mínimo de la siguiente manera:

Reducción al problema de corte mínimo

Para resolver este problema, lo podemos reducir al problema de corte mínimo de la siguiente manera:

1. Asumo que ya me pagaron todos las tareas. Entonces, no realizar la tarea t tiene un costo $b(t)$.

Reducción al problema de corte mínimo

Para resolver este problema, lo podemos reducir al problema de corte mínimo de la siguiente manera:

1. Asumo que ya me pagaron todos las tareas. Entonces, no realizar la tarea t tiene un costo $b(t)$.
2. Defino un grafo $G = (V, E)$
3. Agrego un nodo fuente S y un nodo sumidero T.
4. Para cada máquina $m \in M$, agrego una arista (S, m) con capacidad $c(m)$.
5. Para cada tarea $t \in T$, agrego una arista (t, T) con capacidad $b(t)$.

Reducción al problema de corte mínimo

Para resolver este problema, lo podemos reducir al problema de corte mínimo de la siguiente manera:

1. Asumo que ya me pagaron todos las tareas. Entonces, no realizar la tarea t tiene un costo $b(t)$.
2. Defino un grafo $G = (V, E)$
3. Agrego un nodo fuente S y un nodo sumidero T.
4. Para cada máquina $m \in M$, agrego una arista (S, m) con capacidad $c(m)$.
5. Para cada tarea $t \in T$, agrego una arista (t, T) con capacidad $b(t)$.
6. Para cada tarea $t \in T$ y cada máquina $m \in M_t$, agrego una arista (m, t) con capacidad ∞.

Reducción al problema de corte mínimo

Para resolver este problema, lo podemos reducir al problema de corte mínimo de la siguiente manera:

1. Asumo que ya me pagaron todos las tareas. Entonces, no realizar la tarea t tiene un costo $b(t)$.
2. Defino un grafo $G = (V, E)$
3. Agrego un nodo fuente S y un nodo sumidero T.
4. Para cada máquina $m \in M$, agrego una arista (S, m) con capacidad $c(m)$.
5. Para cada tarea $t \in T$, agrego una arista (t, T) con capacidad $b(t)$.
6. Para cada tarea $t \in T$ y cada máquina $m \in M_t$, agrego una arista (m, t) con capacidad ∞.
7. Calculo el corte mínimo entre S y T en G.
8. El máximo beneficio alcanzable es $\sum_{t \in T} b(t)$ menos el costo del corte mínimo. Esto representa la necesidad de la maquina m para la tarea t.

Reducción al problema de corte mínimo

Para resolver este problema, lo podemos reducir al problema de corte mínimo de la siguiente manera:

1. Asumo que ya me pagaron todos las tareas. Entonces, no realizar la tarea t tiene un costo $b(t)$.
2. Defino un grafo $G = (V, E)$
3. Agrego un nodo fuente S y un nodo sumidero T.
4. Para cada máquina $m \in M$, agrego una arista (S, m) con capacidad $c(m)$.
5. Para cada tarea $t \in T$, agrego una arista (t, T) con capacidad $b(t)$.
6. Para cada tarea $t \in T$ y cada máquina $m \in M_t$, agrego una arista (m, t) con capacidad ∞.
7. Calculo el corte mínimo entre S y T en G.
8. El máximo beneficio alcanzable es $\sum_{t \in T} b(t)$ menos el costo del corte mínimo. Esto representa la necesidad de la maquina m para la tarea t.
9. Si la arista (S, m) forma parte del corte, compro la máquina m.
10. Si la arista (t, T) forma parte del corte, entonces no realizo la tarea t.

Generalización del problema de asignación

Sea $G = (V, E)$ un grafo dirigido y una función de valor $w : E \to \mathbb{Z}$, queremos elegir un subconjunto de nodos $V' \subseteq V$ tal que:

1. $\forall (u, v) \in E, v \in V' \implies u \in V'$.
2. Maximiza $\sum_{v \in V'} w(v)$

Generalización del problema de asignación

Sea $G = (V, E)$ un grafo dirigido y una función de valor $w : E \to \mathbb{Z}$, queremos elegir un subconjunto de nodos $V' \subseteq V$ tal que:

1. $\forall (u, v) \in E, v \in V' \implies u \in V'$.
2. Maximiza $\sum_{v \in V'} w(v)$

Para resolver este problema construimos un grafo $G' = (V', E')$:

1. $V' = V \cup \{S, T\}$, donde S es el nodo fuente y T es el nodo sumidero.

Generalización del problema de asignación

Sea $G = (V, E)$ un grafo dirigido y una función de valor $w : E \to \mathbb{Z}$, queremos elegir un subconjunto de nodos $V' \subseteq V$ tal que:

1. $\forall (u, v) \in E, v \in V' \implies u \in V'$.
2. Maximiza $\sum_{v \in V'} w(v)$

Para resolver este problema construimos un grafo $G' = (V', E')$:

1. $V' = V \cup \{S, T\}$, donde S es el nodo fuente y T es el nodo sumidero.
2. Para cada nodo $v \in V$, si $w(v) < 0$, agrego una arista (S, v) con capacidad $-w(v)$, y si $w(v) > 0$, agrego una arista (v, T) con capacidad $w(v)$.

Generalización del problema de asignación

Sea $G = (V, E)$ un grafo dirigido y una función de valor $w : E \to \mathbb{Z}$, queremos elegir un subconjunto de nodos $V' \subseteq V$ tal que:

1. $\forall (u, v) \in E, v \in V' \implies u \in V'$.
2. Maximiza $\sum_{v \in V'} w(v)$

Para resolver este problema construimos un grafo $G' = (V', E')$:

1. $V' = V \cup \{S, T\}$, donde S es el nodo fuente y T es el nodo sumidero.
2. Para cada nodo $v \in V$, si $w(v) < 0$, agrego una arista (S, v) con capacidad $-w(v)$, y si $w(v) > 0$, agrego una arista (v, T) con capacidad $w(v)$.
3. Para cada arista $(u, v) \in E$, agrego una arista $(u, v) \in E'$ con capacidad ∞.

Generalización del problema de asignación

Sea $G = (V, E)$ un grafo dirigido y una función de valor $w : E \to \mathbb{Z}$, queremos elegir un subconjunto de nodos $V' \subseteq V$ tal que:

1. $\forall (u, v) \in E, v \in V' \implies u \in V'$.
2. Maximiza $\sum_{v \in V'} w(v)$

Para resolver este problema construimos un grafo $G' = (V', E')$:

1. $V' = V \cup \{S, T\}$, donde S es el nodo fuente y T es el nodo sumidero.
2. Para cada nodo $v \in V$, si $w(v) < 0$, agrego una arista (S, v) con capacidad $-w(v)$, y si $w(v) > 0$, agrego una arista (v, T) con capacidad $w(v)$.
3. Para cada arista $(u, v) \in E$, agrego una arista $(u, v) \in E'$ con capacidad ∞.
4. Calculo el corte mínimo entre S y T en G'.

Generalización del problema de asignación

Sea $G = (V, E)$ un grafo dirigido y una función de valor $w : E \to \mathbb{Z}$, queremos elegir un subconjunto de nodos $V' \subseteq V$ tal que:

1. $\forall (u, v) \in E, v \in V' \implies u \in V'$.
2. Maximiza $\sum_{v \in V'} w(v)$

Para resolver este problema construimos un grafo $G' = (V', E')$:

1. $V' = V \cup \{S, T\}$, donde S es el nodo fuente y T es el nodo sumidero.
2. Para cada nodo $v \in V$, si $w(v) < 0$, agrego una arista (S, v) con capacidad $-w(v)$, y si $w(v) > 0$, agrego una arista (v, T) con capacidad $w(v)$.
3. Para cada arista $(u, v) \in E$, agrego una arista $(u, v) \in E'$ con capacidad ∞.
4. Calculo el corte mínimo entre S y T en G'.
5. El conjunto de nodos V' alcanzables desde S en la red residual.
6. El valor de V' es la suma de los $w(v)$ positivos menos el corte minimo.

Outline

Capacidad en los nodos

Si necesitamos que los nodos $v \in V$ tengan una capacidad $c(v)$, podemos separar cada nodo v en v_{in}, v_{out} y :

- Reemplazar cada arista $(u, v) \in E$ por la arista (u_{out}, v_{in}) con capacidad $c(u, v)$.
- Agregar para cada nodo v una arista (v_{in}, v_{out}) con capacidad $c(v)$.

Aristas con flujo mínimo

Sea $G = (V, E)$ un grafo dirigido con nodos sumidero S y T, y funciones de flujo mínimo y máximo $c, d : E \to \mathbb{N}$. Queremos:

1. Decidir si es posible asignar un flujo $f : E \to \mathrm{N}$ tal que $c(e) \leq f(e) \leq d(e) \forall e \in E$ y $\forall v \in V - \{S, T\}, \sum_{u:(u,v)\in E} f(u, v) = \sum_{w:(v,w)\in E} f(v, w)$.
2. Si es posible, encontrar un flujo f que cumpla lo anterior y maximice el valor $|f|$.

Aristas con flujo mínimo - Solución (I)

Para resolver este problema:

1. Construimos un grafo $G' = (V', E')$ donde $V' = V \cup \{S', T'\}$, nuevos nodos fuente y sumidero.

Aristas con flujo mínimo - Solución (I)

Para resolver este problema:

1. Construimos un grafo $G' = (V', E')$ donde $V' = V \cup \{S', T'\}$, nuevos nodos fuente y sumidero.
2. Para cada $(u, v) \in E$, agregamos a E' una arista (u, v) con capacidad $d(u, v) - c(u, v)$, arista (S', v) y (u, T'), ambas con capacidad $c(u, v)$.

Aristas con flujo mínimo - Solución (I)

Para resolver este problema:

1. Construimos un grafo $G' = (V', E')$ donde $V' = V \cup \{S', T'\}$, nuevos nodos fuente y sumidero.
2. Para cada $(u, v) \in E$, agregamos a E' una arista (u, v) con capacidad $d(u, v) - c(u, v)$, arista (S', v) y (u, T'), ambas con capacidad $c(u, v)$.
3. Agrego una arista (T, S) con capacidad ∞
4. Calculamos el un flujo máximo f' de S' a T'. Si hay aristas salientes de S' o entrantes a T' que no esten saturadas, el flujo no es satisfacible.

Aristas con flujo mínimo - Solución (II)

Si el flujo es satisfacible, podemos recuperar el flujo f de la siguiente forma, metiendonos dentro del algoritmo de flujo máximo:

1. Aplicamos el algoritmo de flujo máximo a G.

Aristas con flujo mínimo - Solución (II)

Si el flujo es satisfacible, podemos recuperar el flujo f de la siguiente forma, metiendonos dentro del algoritmo de flujo máximo:

1. Aplicamos el algoritmo de flujo máximo a G.
2. Para la arista $(u, v) \in E$, la capacidad será $d(u, v)$ y para su arista complementaria su capacidad será $-c(u, v)$ en vez de 0.

Aristas con flujo mínimo - Solución (II)

Si el flujo es satisfacible, podemos recuperar el flujo f de la siguiente forma, metiendonos dentro del algoritmo de flujo máximo:

1. Aplicamos el algoritmo de flujo máximo a G.
2. Para la arista $(u, v) \in E$, la capacidad será $d(u, v)$ y para su arista complementaria su capacidad será $-c(u, v)$ en vez de 0.
3. En vez de empezar con el flujo neutro, empezamos con el flujo f'. Es decir, $f(e) = f'(e)$ y $f(e') = -f'(e)$ para cada $e \in E$.

Aristas con flujo mínimo - Solución (II)

Si el flujo es satisfacible, podemos recuperar el flujo f de la siguiente forma, metiendonos dentro del algoritmo de flujo máximo:

1. Aplicamos el algoritmo de flujo máximo a G.
2. Para la arista $(u, v) \in E$, la capacidad será $d(u, v)$ y para su arista complementaria su capacidad será $-c(u, v)$ en vez de 0.
3. En vez de empezar con el flujo neutro, empezamos con el flujo f'. Es decir, $f(e) = f'(e)$ y $f(e') = -f'(e)$ para cada $e \in E$.
4. Ejecutamos el resto del algoritmo de flujo máximo de S a T de forma normal. El flujo resultante será valido y máximo.

Multi Source/ Multi Sink

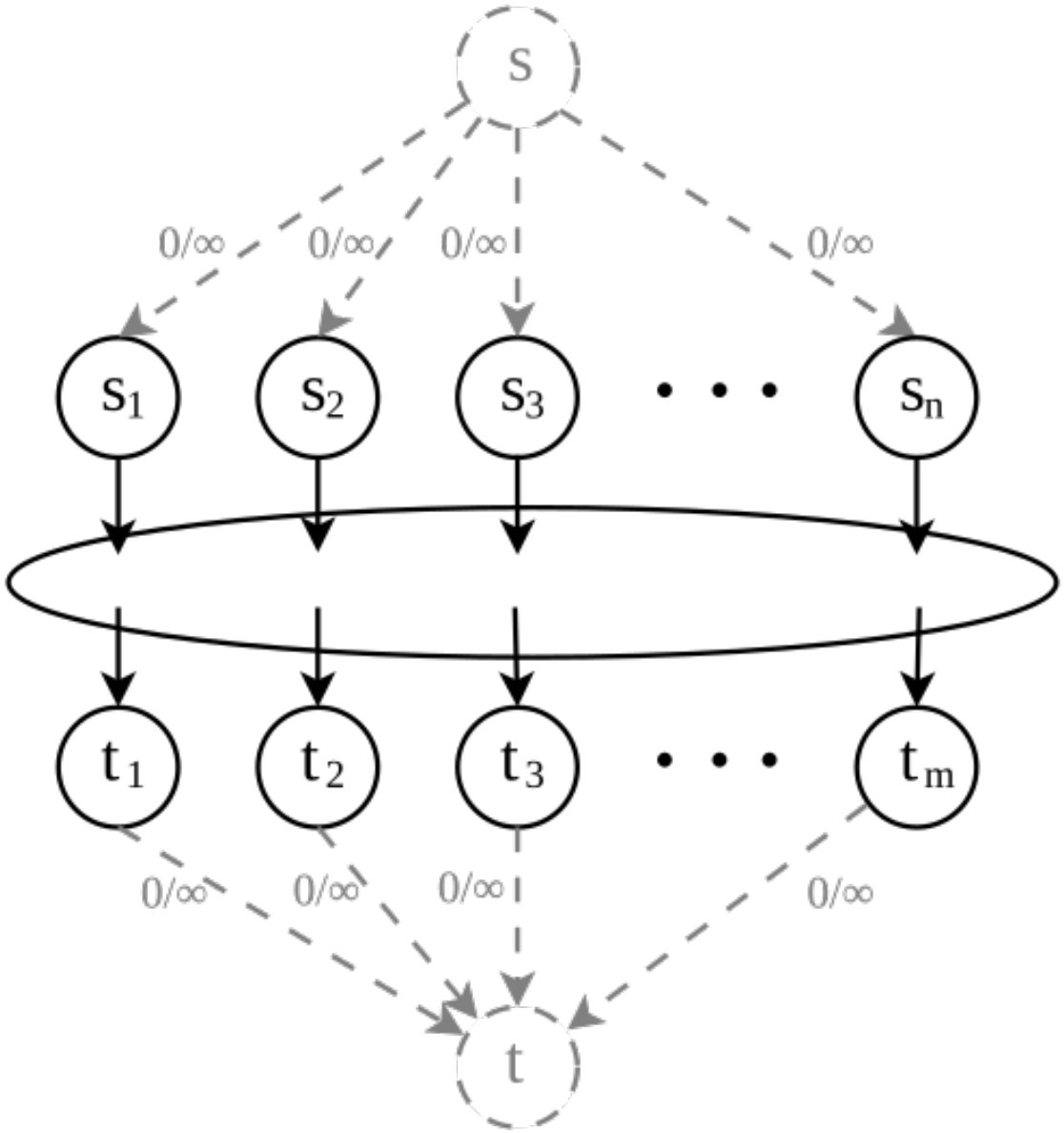
Si quiero tener multiples fuentes $S_1, S_2, \ldots S_n$ y multiples sumideros $T_1, T_2, \ldots, T_m$, lo que hago es tener fuente y sumidero fantasmas S', T' y agregar aristas (S', S_i) y (T_i, T') con capacidad infinita.

Multi Source/ Multi Sink

Si quiero tener multiples fuentes $S_1, S_2, \ldots S_n$ y multiples sumideros $T_1, T_2, \ldots, T_m$, lo que hago es tener fuente y sumidero fantasmas S', T' y agregar aristas (S', S_i) y (T_i, T') con capacidad infinita.

Si quiero vincular a una fuente especifica con un sumidero especifico (es decir, pensar que mando distintos fluidos por la red) no puedo utilizar los algoritmos comunes de flujo y debo recurrir a Programación Líneal (que si queremos valores enteros se vuelve NP-Hard, pero para valores reales se puede resolver en tiempo polinomial)

Multi Source/ Multi Sink (II)



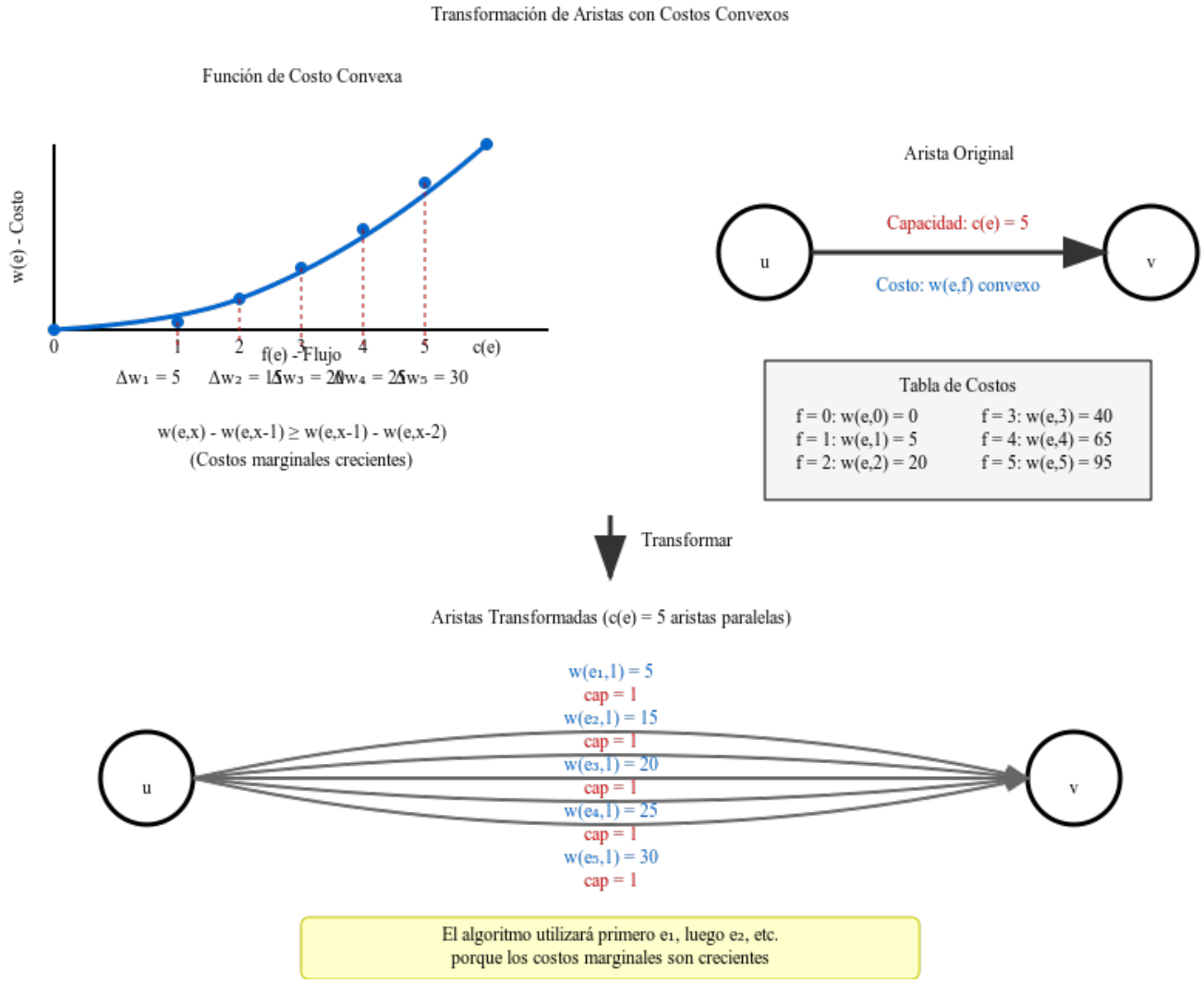
Costos convexos

En Min Cost Max Flow, si se cumple que:

1. El costo w de una arista no es líneal en la cantidad de flujo. Es decir, $w(e) \neq k(e) * f(e)$.
2. El costo w es una función convexa de $f(e)$, es decir, $w(e, x) - w(e, x-1) \geq w(e, x-1) - w(e, x-2) \forall x > 1$.

Podemos resolver el problema reemplazando la arista e por $c(e)$ aristas de capacidad 1 donde $w(e_i, 1) = w(e, i) - w(e, i-1)$. Es decir, $w(e_i, 1)$ es el costo marginal de enviar la i-ésima unidad de flujo por la arista e. Como los costos marginales son crecientes, el algoritmo utilizará primero las aristas correspondientes a las primeras unidades de flujo.

Costos convexos (II)



Cierre

¡Gracias por ver!