Optimización de Redes de Distribución Energética

Daniel Machado Perez Ariel Gonzalez Gómez
Leonardo Artiles Montero Alex Bas Beovides

1. Contexto práctico

En una red de distribución energética de la UNE, las subestaciones están interconectadas mediante una red de líneas de transmisión. Cada línea tiene un costo de mantenimiento. Cuando ocurre un fallo en una línea de transmisión [1], la red debe reorganizarse dinámicamente para minimizar los costos de reparación y garantizar que el suministro energético a las regiones más críticas sea continuo [2].

El jefe de la UNE desea dividir la red en k grupos independientes (subredes autónomas) durante situaciones de mantenimiento o emergencias, asegurándose de que la desconexión sea lo menos costosa posible (en términos de la suma de los costos de las líneas cortadas).

Adicionalmente, al jefe de la UNE le llegó de las altas esferas la orientación de resolver otra tarea. Se quiere garantizar que existan zonas priorizadas que no sufran cortes de electricidad por su relevancia. Estas zonas pueden incluir hospitales, instituciones importantes como el ICRT, residencias de mandatarios... Por ello, se plantea la necesidad de determinar la cantidad mínima de aristas que deben removerse para obtener al menos una componente conexa con exactamente t nodos, representando estas zonas priorizadas.

2. Formalización del problema

Dado un grafo ponderado $G = (V, E)$, donde:

- V: Conjunto de nodos que representan subestaciones.
- E: Conjunto de aristas que representan líneas de transmisión, cada una con un costo $w(e)$ asociado.
- k: Número de subredes requeridas.
- t: Tamaño de la componente priorizada.

Problemas:

1. Encontrar un conjunto mínimo de aristas $E' \subseteq E$ tal que al eliminar E':

[1] Nada frecuente, por cierto.
[2] Qué buen chiste.

a) G se divide en k componentes conexas.

b) La suma de los costos de las aristas eliminadas $\sum_{e \in E'} w(e)$ es mínima.

2. Determinar la cantidad mínima de aristas a remover del grafo G tal que quede al menos una componente conexa con exactamente t nodos.

3. Justificación del Problema

El problema que planteamos tiene una relevancia práctica evidente y un interés computacional significativo, como se detalla a continuación:

3.1. Importancia Práctica

- **Gestión de Redes Energéticas:** Las redes de transmisión eléctrica son infraestructuras críticas en cualquier sociedad moderna. Garantizar que dichas redes sean resilientes a fallos y puedan reorganizarse eficientemente en situaciones de emergencia o mantenimiento es fundamental para asegurar la continuidad del suministro eléctrico. Este problema no solo aborda la minimización de costos económicos asociados con los cortes, sino también prioriza el suministro de energía a zonas críticas como hospitales, instituciones públicas relevantes o residencias estratégicas.

- **Seguridad y Resiliencia:** Dividir la red en k subredes autónomas durante emergencias no solo reduce los costos, sino que fortalece la seguridad al evitar apagones masivos. Además, identificar las aristas mínimas necesarias para garantizar una subred priorizada permite planificar contingencias con antelación.

3.2. Interés Computacional

- **Complejidad Algorítmica:** El problema presenta retos importantes desde el punto de vista computacional. Dividir un grafo ponderado en k subcomponentes conexas con costo mínimo, así como garantizar subredes priorizadas, son problemas que combinan técnicas avanzadas de análisis de grafos, optimización y conectividad.

- **Modelación Compleja:** Formalizar este problema implica trabajar con múltiples restricciones, como los costos de las aristas, cardinalidad de componentes conexas y prioridades en nodos específicos. Estos desafíos hacen que sea ideal para aplicar y demostrar habilidades avanzadas de diseño y análisis algorítmico.

3.3. Aplicaciones Generales

El problema general de partición de grafos tiene aplicaciones relevantes en múltiples dominios. Entre las más destacadas se encuentran las siguientes:

- **Particionamiento y Clustering:** La partición de grafos es una técnica ampliamente utilizada en el análisis de datos. En este contexto, las similitudes entre objetos se modelan como una matriz de adyacencia ponderada, que contiene la información necesaria para realizar agrupamientos óptimos.

- **Confiabilidad de Redes:** Determinar la conectividad mínima de una red es esencial en el diseño y análisis de redes confiables. Las aplicaciones incluyen redes de telecomunicaciones, diseño de infraestructura crítica y sistemas distribuidos.
- **Algoritmos de Optimización:** Los cálculos de cortes mínimos son fundamentales en algoritmos de eliminación de subtours para resolver problemas combinatorios como el *Traveling Salesman Problem* (TSP). Este enfoque ha demostrado ser clave en algoritmos basados en planos de corte para problemas de grafos conexos.
- **Diseño de Circuitos y CAD:** En diseño de circuitos integrados (VLSI), la partición eficiente de circuitos es un paso crucial en herramientas de diseño asistido por computadora (CAD). Esta técnica también se utiliza en modelos de optimización a gran escala y problemas de elementos finitos.
- **Compiladores para Lenguajes Paralelos:** En compiladores de lenguajes paralelos, la partición de grafos optimiza la distribución de operaciones del programa en arquitecturas de memoria distribuida, mejorando la eficiencia computacional.
- **Modelos Físicos y Visión Computacional:** Los problemas de partición y corte en grafos también aparecen en la resolución de modelos físicos como los *spin glass models*, en programación cuadrática 0-1 sin restricciones, y en problemas de segmentación de imágenes, donde las aristas modelan relaciones entre píxeles adyacentes.

3.4. Segunda Parte del Problema: Subred Priorizada

Un aspecto crucial del problema es la identificación de las aristas mínimas necesarias para garantizar que una subred específica, considerada prioritaria, permanezca operativa en situaciones de emergencia. Esto tiene aplicaciones concretas como:

- **Planificación de Contingencias:** En sistemas críticos, como redes eléctricas o de telecomunicaciones, es esencial garantizar que ciertas áreas estratégicas sigan operativas. Por ejemplo, hospitales, estaciones de emergencia y centros de comando requieren una conexión continua incluso en escenarios de fallo masivo.
- **Optimización de Recursos:** La identificación de aristas mínimas permite optimizar los recursos disponibles, como redundancias de red, mantenimiento preventivo y planificación de reparaciones, reduciendo los costos operativos.
- **Resiliencia en Infraestructuras Críticas:** El diseño de infraestructuras resilientes exige priorizar las conexiones más críticas dentro de la red. Este enfoque no solo mejora la robustez general, sino que también asegura una recuperación más rápida y eficiente ante eventos adversos.

La solución de esta parte del problema se basa en un análisis detallado de la conectividad de la red, utilizando técnicas avanzadas de teoría de grafos y algoritmos de optimización para garantizar que las restricciones se cumplan con el menor costo posible.

4. Trabajo propuesto

1. **Demostración de NP-Completitud**: Demostración de que la primera parte del problema es **NP-Completo**, utilizando una reducción al problema *Max-Clique*.

2. **Reducción a árboles y solución greedy**: Reducción de la primera parte del problema a árboles y diseñar una solución greedy con complejidad $O(n \log n)$.

3. **Algoritmo de aproximación con flujo**: Proponer un algoritmo de aproximación basado en técnicas de flujo para resolver la primera parte del problema en grafos generales.

4. **Reducción de la segunda parte del problema a árboles**: Reducir la segunda parte del problema a árboles y resolverlo utilizando un increible algoritmo de programación dinámica, con una demostración de complejidad $O(n^2)$ inesperada.

5. Demostración de que el problema de decisión es NP-Completo

El problema de decisión se puede definir de la siguiente manera:

Entrada: Un grafo $G = (V, E)$, un número entero positivo k, una función no negativa de pesos sobre las aristas $w(e)$ y un umbral M.

Pregunta: Determinar si es posible particionar G en k componentes conexas de manera que la suma de los pesos de las aristas que conectan nodos de componentes conexas distintas sea a lo sumo M.

Por simplicidad, en lo siguiente llamaremos $P - decision$ a este problema.

5.1. Pertenencia a NP

Para verificar si $P - decision$ pertenece a NP, se necesita demostrar que cualquier solución candidata puede ser verificada en tiempo polinomial.

Dado un grafo $G = (V, E)$, una partición de los vértices en k subconjuntos $S_1, S_2, \ldots, S_k$ y un umbral M, se puede verificar en tiempo polinomial si la solución es válida de la siguiente manera:

- Comprobar que la partición es válida, es decir, que cada S_i es no vacío, la unión de todos los S_i es V y las intersecciones son vacías ($S_i \cap S_j = \emptyset$ para $i \neq j$). Esto se puede realizar en tiempo $O(|V|)$ usando estructuras para marcar vértices visitados.

- Comprobar que cada subconjunto S_i es una componente conexa. Esto se puede hacer con BFS/DFS en complejidad $O(|V| + |E|)$.

- Comprobar que la suma de los pesos de las aristas que cruzan componentes conexas es menor o igual que M. Esto se resuelve iterando por todas las aristas, y si una

arista involucra vértices de subconjuntos distintos, se suma $w(e)$ al total. Al final se verifica si $w(e) \leq M$. La complejidad es $O(|E|)$.

Dado que cada paso se puede realizar en tiempo polinomial, $P - decision$ pertenece a NP.

5.2. NP-Hardness

Procederemos a demostrar que $P - decision$ es NP-Hard mediante una reducción polinomial desde el problema de decisión de **Clique Máximo**, que es NP-Completo.

Recordemos la definición de **Clique Máximo**:
Entrada: Un grafo no dirigido $G = (V, E)$ y $M \in \mathbb{Z}^+$
Pregunta: Determinar si existe un clique en G de tamaño mayor o igual que M.

5.2.1. Reducción desde Clique Máximo

Demostración. Consideremos un grafo no dirigido con pesos de $\{0, 1\}$ en las aristas. Encontrar la mínima configuración de aristas que al ser eliminadas particionan el grafo en k componentes conexas, es equivalente a maximizar la cantidad de aristas dentro de cada componente conexa. Supongamos que G tiene un clique H de tamaño $M = |V| - (k-1)$. Entonces el número de aristas entre cualquier vértice $v \notin H$ y $u \in H$ es menor que la cantidad de aristas entre vértices dentro de H, pues en un clique hay una arista entre cualquier par de vértices:

$$\sum_{u \in H, v \notin H} w(e_{uv}) < \sum_{u \in H, v \in H} w(e_{uv})$$

Entonces cuando se divide el grafo en k particiones con una cantidad mínima de aristas de cruce entre componentes conexas, el clique H debe ser una de esas componentes.

$\Rightarrow$ El problema de decisión de **Clique Máximo** puede reducirse en tiempo polinomial a $P - decision$.
Como se sabe que **Clique Máximo** es NP-Completo $\Rightarrow$ $P - decision$ es NP-Hard.

□

5.3. NP-Completitud

Dado que $P - decision$ pertenece a NP y es NP-Hard, concluimos que es NP-Completo.

■

6. Demostración de que el problema de optimización es NP-Hard

El problema de optimización se puede definir de la siguiente manera:

Entrada: Un grafo $G = (V, E)$, un número entero positivo k y una función no negativa de pesos sobre las aristas $w(e)$.

Salida: Encontrar una partición de G en k componentes conexas de manera que la suma de los pesos de las aristas que conectan nodos de componentes conexas distintas es mínima.

Por simplicidad, en lo siguiente llamaremos $P - optimization$ a este problema.

6.1. NP-Hardness

Demostración. Como se conoce que dado un problema, si su variante de decisión es NP-Completo entonces su variante de optimización es NP-Hard, y se demostró que $P - decision$ es NP-Completo $\Rightarrow$ $P - optimization$ es NP-Hard. □

7. Solución de subproblemas de la primera parte con estrategia *Greedy*

En las dos secciones siguientes se expone la idea de los algoritmos *Greedy* para resolver el problema en dos casos particulares: cuando el grafo es un árbol y cuando es un bosque.

7.1. Caso en que el grafo es un árbol

Dado un árbol $T = (V, E)$ con pesos no negativos en sus aristas, la idea es aprovechar la propiedad de que, al remover $k-1$ aristas, el árbol se divide en k componentes conexas. El algoritmo Greedy consiste en:

1. Ordenar las aristas del árbol en forma no decreciente según su peso.
2. Seleccionar las $k - 1$ aristas de menor peso.

7.1.1. Análisis de Complejidad

- Ordenar las aristas requiere $O(n \log n)$, donde n es el número de vértices (recordando que un árbol tiene $n - 1$ aristas).
- La selección de las $k - 1$ aristas es $O(k)$, lo cual es polinomial.

Por lo tanto, el algoritmo Greedy para árboles se resuelve en tiempo polinomial.

7.1.2. Demostración de Correctitud

Recordemos que, en un árbol, al eliminar ciertas aristas se incrementa el número de componentes conexas.

Lema 7.1 (Número de componentes en un árbol)**.** *Si eliminamos todas las aristas de un árbol con n vértices, es decir, eliminamos $n - 1$ aristas, obtenemos n componentes conexas.*

Demostración. Procedemos por inducción en el número de vértices n.

Caso base: Si $n = 2$, un árbol tiene exactamente una arista. Al remover esta única arista, el grafo se divide en 2 componentes conexas, lo cual cumple la afirmación.

Hipótesis de inducción: Supongamos que para un árbol con $n = k - 1$ vértices, removiendo $(k - 2)$ aristas obtenemos $k - 1$ componentes conexas.

Paso inductivo: Consideremos un árbol T con k vértices. Removamos una arista e del árbol. Al hacerlo, el árbol se divide en dos subárboles, digamos T_1 y T_2, que tienen n_1 y n_2 vértices respectivamente, donde $n_1 + n_2 = k$ y ambos $n_1, n_2 \geq 1$. Por la hipótesis de inducción, removiendo $n_1 - 1$ aristas de T_1 se obtienen n_1 componentes conexas y removiendo $n_2 - 1$ aristas de T_2 se obtienen n_2 componentes conexas. En total, removiendo $1 + (n_1 - 1) + (n_2 - 1) = n_1 + n_2 - 1 = k - 1$ aristas, obtenemos $n_1 + n_2 = k$ componentes conexas. □

Lema 7.2. *Sea $T = (V, E)$ un árbol con pesos no negativos en sus aristas, y sea la lista de aristas ordenada de forma no decreciente según sus pesos:*

$$w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{n-1}).$$

Entonces, el conjunto de aristas de menor peso que separa el árbol en k componentes conexas está dado por:

$$C = \{e_1, e_2, \ldots, e_{k-1}\}.$$

Demostración. Procedemos por contradicción. Supongamos que existe otro conjunto C' de $k - 1$ aristas tal que el peso total $w(C')$ es menor que $w(C)$. Sin embargo, al remover $k - 1$ aristas de T, por el resultado del lema anterior, el árbol se divide en k componentes conexas. Como C está formado por las $k-1$ aristas de menor peso, cualquier otro conjunto C' de $k - 1$ aristas necesariamente tendrá un peso total mayor o igual a $w(C)$. Esto contradice la hipótesis de que $w(C') < w(C)$, por lo que C debe ser el conjunto deseado. □

Corollary 1. *Sobre árboles, el problema se puede resolver en tiempo polinomial, ya que basta con ordenar las aristas (lo cual se puede hacer en tiempo $O(n \log n)$) y seleccionar las $k - 1$ aristas de menor peso.*

7.1.3. Pseudocódigo

```
Entrada: Arbol T = (V, E) con pesos no negativos, entero k
Salida: Conjunto de aristas C que forma un corte minimo k

1. Ordenar las aristas E de T en forma no decreciente segun su peso:
     w(e_1) <= w(e_2) <= ... <= w(e_{|V|-1})
2. Seleccionar las primeras k-1 aristas de la lista ordenada y asignarlas a C.
3. Retornar C.
```

7.2. Caso en que el grafo es un bosque

Para un bosque $F = (V, E)$ con t componentes conexas, la idea es similar. Se debe notar que, para obtener una partición en k componentes, es necesario remover $k-t$ aristas, ya que el bosque ya cuenta con t componentes. El algoritmo Greedy consiste en:

1. Ordenar todas las aristas del bosque en forma no decreciente según su peso.
2. Seleccionar las $k - t$ aristas de menor peso.

7.2.1. Análisis de Complejidad

- Ordenar las aristas requiere $O(|E| \log |E|)$; dado que el bosque tiene a lo sumo $|V|-1$ aristas, se cumple que es polinomial.
- La selección de las $k - t$ aristas es $O(k)$.

Por lo tanto, este algoritmo también se ejecuta en tiempo polinomial.

7.2.2. Demostración de Correctitud

Consideremos ahora un bosque, es decir, un grafo disconexo cuyos componentes son árboles.

Proposición 7.1. *Sea t el número de componentes conexas en un bosque $F = (V, E)$. Para cualquier $n \geq t$, si eliminamos $(n - t)$ aristas del bosque, obtenemos n componentes conexas.*

Demostración. Procedemos por inducción en n.

Caso base: Si $n = t$, no eliminamos ninguna arista, y el bosque ya tiene t componentes, lo que cumple la afirmación.

Hipótesis de inducción: Supongamos que para $n = k-1$ (con $k-1 \geq t$), removiendo $(k - 1 - t)$ aristas se obtienen $k - 1$ componentes conexas.

Paso inductivo: Consideremos un bosque F y supongamos que removiendo $(k-1-t)$ aristas se obtienen $k - 1$ componentes conexas. Ahora, removamos una arista adicional de uno de los árboles resultantes. Por el lema aplicado a árboles, al remover una arista de ese árbol se incrementa el número de componentes en 1. Así, en total, removiendo $(k - 1 - t) + 1 = k - t$ aristas, obtenemos k componentes conexas. □

Lema 7.3. *Sea $F = (V, E)$ un bosque con pesos no negativos en sus aristas, y sean t el número de componentes conexas de F y las aristas ordenadas por la función de peso de forma no decreciente:*

$$w(e_1) \leq w(e_2) \leq \cdots \leq w(e_{|V|}).$$

Entonces, el conjunto de aristas de menor peso cuya eliminación divide el bosque en k componentes conexas está dado por:

$$C = \{e_1, e_2, \ldots, e_{k-t}\}.$$

Demostración. Supongamos, para obtener una contradicción, que existe un conjunto C' de aristas con $|C'| < k - t$ y $w(C') < w(C)$ que, al removerlas, divide el bosque en k componentes conexas. Sin embargo, según la proposición anterior, remover menos de $k-t$ aristas en un bosque con t componentes produce menos de k componentes conexas. Esto contradice la definición de C'. Por lo tanto, C es el conjunto de aristas de menor peso cuya eliminación divide F en k componentes conexas. □

7.2.3. Pseudocódigo

```
Entrada: Bosque F = (V, E) con pesos no negativos, entero k,
        donde t es el numero de componentes conexas en F.
Salida: Conjunto de aristas C que forma un corte minimo k en F

1. Calcular t, el numero de componentes conexas de F.
2. Ordenar las aristas E de F en forma no decreciente segun su peso:
     w(e_1) <= w(e_2) <= ... <= w(e_{|V|-?})
3. Seleccionar las primeras k-t aristas de la lista ordenada y asignarlas a C.
4. Retornar C.
```

8. Reducción de la segunda parte del problema a árboles y solución con Programación Dinámica

8.1. Enunciado del problema

Dado un árbol con N nodos, calcular para todo $K \leq N$ el mínimo número de aristas a ser cortadas tal que quede una componente conexa con exactamente K nodos.

8.2. Solución

Resolvamos el problema utilizando programación dinámica. Rootiemos el árbol T, llamemos T_u al subárbol del nodo u. Formulemos nuestra DP de la siguiente forma:

Sea $dp[u][i]$ la mínima cantidad de aristas que hay que eliminar del árbol para obtener una componente conexa en T_u que contenga al nodo u y sea de tamaño i. Es trivial que teniendo este arreglo rellenado podemos calcular la solución para cada K en $O(N)$.

Veamos cómo calcular esta DP. Denotemos T_u^j como el árbol T_u pero conteniendo solo sus j primeros hijos y sus subárboles.

Llamemos $dp_j[u][i]$ el valor de $dp[u][i]$ en T_u^j, lo cual es básicamente habiendo procesado los j primeros hijos de u. La idea es calcular $dp_h[u][i]$ donde h es la cantidad de nodos hijos de u, es decir, $h = |\{v_1, v_2, v_3, \ldots, v_h\}|$ donde v_i es el hijo i-ésimo de u.

Empezamos con $j = 0$, en ese caso u es tomado como una hoja en T_u^j, así que $dp_j[u][1] = 0$. Luego computamos los valores de $dp_j[u][i]$ siguiendo la siguiente fórmula:

$$dp_j[u][i] = \text{mín}\left(dp_{j-1}[u][i] + 1,\ \min_{\substack{a+b=i \\ a,b\geq 1}}(dp_{j-1}[u][a] + dp[v_j][b])\right)$$

Lo cual es básicamente tomar el mínimo entre:

- Cortar la arista hacia el j-ésimo hijo de u.
- El mínimo para todo a, b de tener a nodos en T_u^{j-1} y tener b nodos en el subárbol del j-ésimo hijo de u.

8.3. Análisis de complejidad

8.3.1. Cota máxima del algoritmo

Podríamos establecer una cota máxima de $O(N^3)$ para nuestro algoritmo, ya que para cada nodo u, debemos llenar todos los valores de i en $dp[u][i]$, y para hacer esto tenemos que recorrer todas las posibles particiones a, b tal que $a + b = i$. Sin embargo, podemos encontrar una mejor cota.

8.3.2. Enunciado

Para un nodo fijo u, el número de operaciones necesarias para computar todos los valores de $dp[v][:]$ donde v es un nodo que pertenece a T_u es $O(|T_u|^2)$.

8.3.3. Demostración

Demostremos este resultado por inducción sobre la profundidad del árbol T_u.

- Caso base: Si T_u es una hoja, el lema se cumple trivialmente, ya que solo necesitamos $O(1)$ operaciones.
- Paso inductivo: Supongamos que el nodo u tiene h hijos, denotados como $v_1, v_2, \ldots, v_h$, y definamos $a_j = |T_{v_j}|$.

  Al fusionar el j-ésimo hijo, es decir, al calcular $dp_j[u][:]$, recorremos todos los valores de $dp_{j-1}[u][a]$ y $dp[v_j][b]$ para todo a y b. Sin embargo, el valor de a puede ser escogido en $|T_u^{j-1}|$ formas y el de b en $|T_{v_j}|$ formas, lo que nos da un número total de operaciones:

$$O(|T_u^{j-1}| \cdot |T_{v_j}|) = O\left((1 + a_1 + a_2 + \cdots + a_{j-1}) \cdot a_j\right)$$

  Por la hipótesis de inducción, calcular todos los valores de dp para los nodos en el subárbol de u tomará:

$$O\left(\sum_{j=1}^{h} a_j^2\right)$$

  Sumando el número de operaciones necesarias para calcular los valores de $dp[v][:]$ para todos los vértices v pertenecientes a T_u, obtenemos:

$$O\left(\sum_{i=1}^{h}\sum_{j=1}^{h} a_i \cdot a_j + \sum_{j=1}^{h} a_j + \sum_{j=1}^{h} a_j^2\right)$$

  Como el primer término es una doble sumatoria sobre los tamaños de los subárboles, podemos agrupar los términos para obtener:

$$O\left(\left(1 + \sum_{j=1}^{h} a_j\right)^2\right) = O(|T_u|^2)$$

8.3.4. Demostración combinatoria alternativa

También podemos obtener una demostración combinatoria del enunciado. Al fusionar un subárbol se realizan:

$$O(|T_u^{j-1}| \cdot |T_{v_j}|)$$

operaciones. Esto se puede interpretar como el número de pares de nodos que tienen a u como su ancestro común más cercano (LCA). Como cualquier par de nodos tiene un único LCA, cada par se cuenta una sola vez, lo que nos lleva a una cota total de:

$$O(N^2)$$

8.4. Demostración de Correctitud

Sea un árbol T con N nodos y sea T_u el subárbol enraizado en el nodo u. Para cada nodo u y para cada entero i en $1, \ldots, |T_u|$, definimos $dp[u][i]$ como el mínimo número de aristas a cortar en T_u para obtener una componente conexa de tamaño i que contiene a u. Además, al procesar el nodo u consideramos sus hijos en un orden fijo y definimos T_u^j como el árbol formado por u y los subárboles de sus primeros j hijos. Sea

$dp_j[u][i] =$ costo mínimo en T_u^j para obtener una componente conexa de tamaño i que contiene a u.

El caso base es $j = 0$, donde el único nodo es u, de modo que

$$dp_0[u][1] = 0,$$

y para $i \neq 1$ se establece $dp_0[u][i] = +\infty$ (o un valor que indique imposibilidad).

Procedemos a demostrar la correctitud mediante doble inducción: primero sobre el número de hijos procesados (índice j) y luego sobre la estructura del árbol.

1. Inducción sobre la fusión de hijos en u

Caso base: Con $j = 0$, el subárbol parcial T_u^0 consiste únicamente en u; por lo tanto,

$$dp_0[u][1] = 0,$$

lo cual es correcto.

Paso inductivo: Supongamos que para cierto $j-1$ la tabla $dp_{j-1}[u][\cdot]$ calcula correctamente el costo mínimo para cada tamaño i en el subárbol T_u^{j-1}. Sea v_j el j-ésimo hijo de u. Al incorporar el subárbol T_{v_j}, se tienen dos opciones:

1. **No incluir T_{v_j}:** Se corta la arista (u, v_j) y se conserva el estado anterior, con un costo adicional de 1:
$$dp_j[u][i] \leq dp_{j-1}[u][i] + 1.$$
2. **Fusionar T_{v_j}:** Se toma una componente en T_u^{j-1} de tamaño a (con costo $dp_{j-1}[u][a]$) y una componente en T_{v_j} de tamaño b (con costo $dp[v_j][b]$); al fusionarlas se obtiene una componente de tamaño $a + b$ con costo:
$$dp_j[u][a+b] \leq dp_{j-1}[u][a] + dp[v_j][b].$$

Por lo tanto, para cada i se tiene:

$$dp_j[u][i] = \min\left\{ dp_{j-1}[u][i] + 1,\ \min_{\substack{a+b=i \\ a,b\geq 1}} \Big(dp_{j-1}[u][a] + dp[v_j][b] \Big) \right\}.$$

Dado que, por hipótesis inductiva, $dp_{j-1}[u][\cdot]$ y $dp[v_j][\cdot]$ son correctos, la fusión produce correctamente $dp_j[u][\cdot]$ en T_u^j.

Al procesar todos los hijos de u (es decir, $j = h$, donde h es el número total de hijos de u), se obtiene:

$$dp_h[u][\cdot] = dp[u][\cdot],$$

lo que demuestra la corrección del estado $dp[u][\cdot]$ en T_u.

2. Inducción en la estructura del árbol

Asumiendo que para cada hijo v de u la tabla $dp[v][\cdot]$ es correcta, el proceso de fusión en u descrito anteriormente garantiza que $dp[u][\cdot]$ se calcula correctamente. Por inducción sobre la estructura del árbol, la DP es correcta para todo el árbol T.

8.5. Pseudocódigo en $O(N^2)$

A continuación se presenta el pseudocódigo que implementa la solución mediante DFS y fusión de estados. Se aprovecha que, al fusionar el estado del nodo u (de tamaño size$[u]$) con el del hijo v (de tamaño size$[v]$), se realizan size$[u]$ $\times$ size$[v]$ operaciones, y gracias a la propiedad combinatoria (cada par de nodos se fusiona una única vez, correspondiente a su LCA) la complejidad total es $O(N^2)$.

```
// Constante que representa un valor "infinito"
const INF = infinito;

// dp[u][i]: Costo minimo para obtener una componente conexa de tamanno i en
    ↪ T_u que contiene a u.
// size[u]: Tamano actual del arreglo dp[u] (inicialmente 1, pues solo se
    ↪ cuenta el nodo u).

procedure DFS(u, parent)
    // Inicializar dp[u] con el caso base: solo el nodo u.
    dp[u] := array[1 ... (tamano maximo posible)] of INF;
    dp[u][1] := 0;
    size[u] := 1;

    // Procesar cada hijo v de u (evitando regresar al padre)
    for each v in vecinos(u) do
        if v == parent then continue;
        DFS(v, u);

        // Preparar un nuevo arreglo para fusionar dp[u] y dp[v]
        newSize := size[u] + size[v];
```

```
        newDP := array[1 ... newSize] of INF;

        // Opcion 1: No fusionar v (cortar la arista (u,v))
        for i from 1 to size[u] do
            newDP[i] := min(newDP[i], dp[u][i] + 1);
        end for;

        // Opcion 2: Fusionar el subarbol de v sin cortar (usar dp[v])
        for i from 1 to size[u] do
            for j from 1 to size[v] do
                newDP[i + j] := min(newDP[i + j], dp[u][i] + dp[v][j]);
            end for;
        end for;

        // Actualizar dp[u] y su tamano
        size[u] := newSize;
        dp[u] := newDP;
    end for;
end procedure;

// Funcion principal: se asume que el arbol tiene N nodos.
procedure Main()
    // Se asume que el arbol esta almacenado en una lista de adyacencia: G.
    // Se elige un nodo arbitrario como raiz, por ejemplo, 1.
    DFS(1, NIL);

    // Actualizar la respuesta global para cada tamano K.
    // La solucion final es el minimo dp[u][K] entre todos los nodos u.
    answer := array[1 ... N] of INF;
    for each nodo u de 1 a N do
        for k from 1 to size[u] do
            answer[k] := min(answer[k], dp[u][k]);
        end for;
    end for;

    // Imprimir o retornar la respuesta para cada K = 1, 2, ..., N.
end procedure;
```

Listing 1: Pseudocódigo en $O(N^2)$

9. Algoritmos de aproximación para k-corte mínimo

A continuación, se presentan dos algoritmos de aproximación propuestos por Huzur Saran y Vijay V. Vazirani [1] para el problema del k-corte mínimo. Cada algoritmo encuentra un k-corte con un peso dentro de un factor de $(2 - 2/k)$ del óptimo. Uno de estos algoritmos es particularmente eficiente, ya que solo requiere un total de $n - 1$ cálculos de flujo máximo para encontrar un conjunto de k-cortes casi óptimos, uno para cada valor de k entre 2 y n.

10. Algoritmo EFFICIENT

Sea $G = (V, E)$ un grafo no dirigido y conexo, y sea wt : $E \to \mathbb{Z}^+$ una asignación de pesos a las aristas de G. Extenderemos la función wt a subconjuntos de E de la manera obvia. Una partición $(V', V - V')$ de V especifica un corte; el corte consiste de todas las aristas, S, que tienen un extremo en cada partición. Denotaremos un corte por el conjunto de sus aristas, S, y definiremos su peso como wt(S). Para cualquier conjunto $S \subseteq E$, denotaremos el número de componentes conexas del grafo $G' = (V, E - S)$ como comps(S).

Primero encontraremos un k-corte para un k especificado. El algoritmo se basa en la siguiente estrategia voraz: sigue eligiendo cortes hasta que su unión sea un k-corte y, en cada iteración, elige el corte más ligero; en el grafo original, esto crea componentes adicionales.

Algoritmo:

1. Para cada arista e, elige un corte de peso mínimo, digamos, s_e, que separe los extremos de e.

2. Ordena estos cortes en orden creciente de peso, obteniendo la lista $s_1, \ldots, s_m$.

3. Selecciona de manera voraz cortes de esta lista hasta que su unión sea un k-corte; el corte s_i se elige solo si no está contenido en $s_1 \cup \cdots \cup s_{i-1}$.

   (Nota: en caso de que el algoritmo termine con un corte B tal que comps$(B) > k$, podemos eliminar fácilmente aristas de B para obtener un corte B' tal que comps$(B') = k$.)

Obsérvese que, dado que la arista e está en s_e, se cumple que $s_1 \cup \cdots \cup s_m = E$, por lo que esto debe ser un n-corte. Por lo tanto, el algoritmo ciertamente tendrá éxito en encontrar un k-corte.

Sean $b_1, \ldots, b_l$ los cortes sucesivos seleccionados por el algoritmo. En el siguiente lema mostramos que con cada corte elegido, aumentamos el número de componentes creadas y, por lo tanto, el número total de cortes seleccionados es como máximo $k - 1$.

Lema 1.1

Para cada i, $1 \leq i < l$,

$$\text{comps}(b_1 \cup \cdots \cup b_{i+1}) > \text{comps}(b_1 \cup \cdots \cup b_i).$$

Demostración. Debido a la manera en que EFFICIENT selecciona los cortes, $b_{i+1} \nsubseteq (b_1 \cup \cdots \cup b_i)$. Sea (u, v) una arista en $b_{i+1} - (b_1 \cup \cdots \cup b_i)$. Claramente, u y v están en la misma componente en el grafo obtenido al eliminar las aristas de $b_1 \cup \cdots \cup b_i$ de G, y están en diferentes componentes en el grafo obtenido al eliminar $b_1 \cup \cdots \cup b_{i+1}$ de G. Por lo tanto, el último grafo tiene más componentes.

Corolario 1.2

El número de cortes seleccionados, l, es como máximo $k-1$.

Obsérvese que en cada paso estamos efectivamente seleccionando el corte más ligero que crea componentes adicionales: entre todas las aristas que se encuentran dentro de componentes conexas, elegimos la arista cuyos extremos pueden ser desconectados con el corte más ligero del grafo original.

La complejidad del algoritmo está dominada por el tiempo requerido para encontrar los cortes $s_1, \ldots, s_m$. Esto puede hacerse claramente con m cálculos de flujo máximo. Se obtiene una implementación más EFFICIENT utilizando cortes de Gomory-Hu. Gomory y Hu [4] muestran que existe un conjunto de $n-1$ cortes en G tal que para cada par de vértices, $u, v \in V$, el conjunto contiene un corte de peso mínimo que separa u y v; además, muestran cómo encontrar dicho conjunto usando solo $n-1$ cálculos de flujo máximo. Los cortes $s_1, \ldots, s_m$ pueden obtenerse claramente a partir de dicho conjunto de $n-1$ cortes.

Incorporando todo esto, obtenemos una descripción particularmente simple del algoritmo. Primero, obsérvese que el paso 3 es equivalente a:

3'. Encuentra el mínimo i tal que $\text{comps}(s_1 \cup \cdots \cup s_i) \geq k$. Devuelve el k-corte $s_1 \cup \cdots \cup s_i$.

A continuación, obsérvese que cuando se implementa con cortes de Gomory-Hu, el Algoritmo EFFICIENT esencialmente se reduce a lo siguiente:

1. Encuentra un conjunto de cortes de Gomory-Hu en G.
2. Ordena estos cortes en orden creciente de peso, obteniendo la lista $g_1, \ldots, g_{n-1}$.
3. Encuentra el mínimo i tal que $\text{comps}(g_1 \cup \cdots \cup g_i) \geq k$.

El algoritmo se extiende de manera directa para obtener k-cortes casi óptimos para cada k, $2 \leq k \leq n$, con el mismo conjunto de cortes de Gomory-Hu.

10.1. Garantía de rendimiento para EFFICIENT

En esta sección demostraremos el siguiente teorema.

Teorema 2.1

El algoritmo EFFICIENT encuentra un k-corte con un peso dentro de un factor de $2 - \frac{2}{k}$ del óptimo.

Sea $b_1, \ldots, b_l$ la sucesión de cortes encontrados por el algoritmo EFFICIENT; sea $B = b_1 \cup \cdots \cup b_l$. El núcleo de la demostración es una propiedad especial de estos cortes con respecto a una enumeración de todos los cortes en G. Sea $c_1, c_2, \ldots$ una tal enumeración, ordenada por peso creciente, tal que $b_1, \ldots, b_l$ aparecen en este orden dentro de la enumeración. A una enumeración de este tipo la llamaremos *consistente* con los cortes $b_1, \ldots, b_l$.

Para cualquier corte c en G, definimos índice(c) como su índice en esta enumeración.

La Propiedad de Unión

Sean $s_1, \ldots, s_p$ un conjunto de cortes en G, ordenados por peso creciente. Consideremos cualquier enumeración de todos los cortes en G, $c_1, c_2, \ldots$, que sea consistente con $s_1, \ldots, s_p$. Diremos que $s_1, \ldots, s_p$ satisfacen la *propiedad de unión* si, intuitivamente, con respecto a cualquier enumeración de este tipo, la unión de los cortes en cualquier segmento inicial de $c_1, c_2, \ldots$ es igual a la unión de todos los cortes s_j contenidos en dicho segmento inicial.

Más formalmente, tomemos cualquier enumeración consistente de todos los cortes en G, sea i cualquier índice tal que $1 \leq i \leq$ índice(s_p), y sea s_q el último corte en el orden que tenga un índice a lo sumo i. Entonces,

$$c_1 \cup \cdots \cup c_i = s_1 \cup \cdots \cup s_q.$$

Lema 2.2

Los cortes $b_1, \ldots, b_l$ satisfacen la propiedad de unión.

Demostración. Supongamos que los cortes no satisfacen la propiedad de unión. Entonces, existe una enumeración de todos los cortes en G, $c_1, c_2, \ldots$, que es consistente con $b_1, \ldots, b_l$, y sin embargo, existe un índice i, $i \leq$ índice(b_l), tal que

$$b_1 \cup \cdots \cup b_q \neq c_1 \cup \cdots \cup c_i,$$

donde b_q es el último corte en la lista que tiene un índice a lo sumo i. Al menor índice de este tipo lo llamaremos el *punto de discrepancia*. Fijemos una enumeración en la que el punto de discrepancia sea máximo. Claramente, el punto de discrepancia tendrá un índice menor que índice(b_l).

Sea b_{q+1} el siguiente corte seleccionado por el algoritmo, y sea índice$(b_{q+1}) = j$. Debido a la forma en que fijamos la enumeración, $\text{wt}(c_j) > \text{wt}(c_i)$ (de lo contrario, podríamos intercambiar c_i y c_j, obteniendo así una enumeración en la que la discrepancia ocurre en un índice mayor).

Claramente, $b_q \neq c_i$ y

$$c_1 \cup \cdots \cup c_{i-1} = b_1 \cup \cdots \cup b_q.$$

Sea $e \in c_i - (c_1 \cup \cdots \cup c_{i-1})$. Entonces,

$$e \notin b_1 \cup \cdots \cup b_q.$$

Claramente, $\text{wt}(s_e) \leq \text{wt}(c_i)$, donde s_e es un corte de peso mínimo que desconecta los extremos de e. Ahora, el algoritmo debe seleccionar la arista e con un corte de peso a lo sumo $\text{wt}(s_e)$, y por lo tanto, a lo sumo $\text{wt}(c_i)$. Dado que $\text{wt}(b_{q+1}) > \text{wt}(c_i)$, esto implica que $e \in b_1 \cup \cdots \cup b_q$, lo que lleva a una contradicción.

Observación

Dado que $b_1, \ldots, b_l$ se derivan de $s_1, \ldots, s_m$, estos últimos cortes también satisfacen la propiedad de unión. Por razones similares, $g_1, \ldots, g_{n-1}$ también satisfacen la propiedad de unión.

Sea A un k-corte de peso mínimo en G. La segunda idea clave de la demostración es interpretar A como la unión de k cortes de la siguiente manera: Sean $V_1, \ldots, V_k$ las componentes conexas de $G' = (V, E - A)$.

Sea a_i el corte que separa V_i de $V - V_i$ para $1 \leq i \leq k$. Entonces,

$$A = \bigcup_{i=1}^{k} a_i.$$

Obsérvese que

$$\sum_{i=1}^{k} \mathrm{wt}(a_i) = 2\mathrm{wt}(A)$$

(porque cada arista de A se cuenta dos veces en la suma). Supongamos, sin pérdida de generalidad, que

$$\mathrm{wt}(a_1) \leq \mathrm{wt}(a_2) \leq \cdots \leq \mathrm{wt}(a_k).$$

La cota $(2 - \frac{2}{k})$ se establece mostrando que la suma de los pesos de los cortes, $b_1, \ldots, b_l$, es como máximo la suma de los pesos de $a_1, \ldots, a_{k-1}$, es decir,

$$\mathrm{wt}(B) \leq \sum_{i=1}^{l} \mathrm{wt}(b_i) \leq \sum_{i=1}^{k-1} \mathrm{wt}(a_i)$$

$$\leq 2(1 - \frac{1}{k})\mathrm{wt}(A),$$

ya que a_k es el corte más pesado de A.

De hecho, será más sencillo probar una afirmación más fuerte. Consideraremos una ligera variante de EFFICIENT que selecciona cortes con multiplicidad; el corte b_i será seleccionado t veces si su inclusión crea t componentes adicionales, es decir, si

$$\mathrm{comps}(b_1 \cup \cdots \cup b_i) - \mathrm{comps}(b_1 \cup \cdots \cup b_{i-1}) = t.$$

De este modo, ahora tenemos exactamente $k - 1$ cortes; los denominaremos $b_1, \ldots, b_{k-1}$ para evitar introducir notación excesiva. Como antes, $b_1, \ldots, b_{k-1}$ están ordenados por peso creciente, y además asumiremos que los cortes con multiplicidad ocurren consecutivamente. Mostraremos que

$$\sum_{i=1}^{k-1} \mathrm{wt}(b_i) \leq \sum_{i=1}^{k-1} \mathrm{wt}(a_i) \qquad \text{(I)}$$

Para el resto de la demostración, estudiaremos cómo se distribuyen los cortes a_i y b_j en $c_1, c_2, \ldots$, una enumeración de todos los cortes en G respecto de la cual $b_1, \ldots, b_{k-1}$ satisfacen la propiedad de unión. Denotemos por α_i el número de todos los cortes a_j que tienen índice $\leq i$, es decir,

$$\alpha_i = |\{a_j \mid \text{índice}(a_j) \leq i\}|.$$

Mostraremos que, para cada índice i, $1 \leq i \leq \text{índice}(a_{k-1})$, el número de cortes b_j que tienen índice $\leq i$ es al menos α_i (por supuesto, los cortes b_j se cuentan con multiplicidad).

Si es así, habrá una correspondencia uno a uno de $\{b_1, \ldots, b_{k-1}\}$ a $\{a_1, \ldots, a_{k-1}\}$ tal que, si b_i se corresponde con a_j, entonces índice$(b_i) \leq$ índice(a_j). Esto probará (I).

Dos propiedades interesantes de los cortes a_i y b_j ayudarán a demostrar la afirmación del párrafo anterior. Denotemos por A_i la unión de todos los cortes a_j que tienen índice $\leq i$, es decir,

$$A_i = \bigcup_{\text{índice}(a_j) \leq i} a_j.$$

De manera similar, sea

$$B_i = \bigcup_{\text{índice}(b_j) \leq i} b_j.$$

El siguiente lema muestra que, para cada índice i, los cortes b_j están haciendo al menos tanto progreso como los cortes a_j, donde el progreso se mide por el número de componentes creadas.

Lema 2.3.

Para cada índice i, se cumple que comps$(A_i) \leq$ comps(B_i).

Demostración. El lema es claramente cierto para $i >$ índice(b_{k-1}). Para $i \leq$ índice(b_{k-1}), se tiene que $B_i = c_1 \cup \cdots \cup c_i$, dado que los cortes $b_1, \ldots, b_{k-1}$ satisfacen la propiedad de unión. Por lo tanto, $A_i \subseteq B_i$, y el lema queda demostrado con esto.

Es fácil construir un ejemplo que muestre que cada corte a_j no necesariamente crea componentes adicionales. Sin embargo, para cada índice i, el número de componentes creadas por los cortes a_j es al menos $\alpha_i + 1$. Esto se establece en el siguiente lema.

Lema 2.4.

Para cada i, $1 \leq i \leq$ índice(a_{k-1}), se cumple que comps$(A_i) \geq \alpha_i + 1$.

Demostración. Para cada corte a_j con índice $\leq i$, la partición V_j será una única componente conexa por sí misma en el grafo $G_i = (V, E - A_i)$. Asociemos a_j a esta componente de G_i. Como índice$(a_k) > i$, la componente de G_i que contiene V_k no será asociado a ningún corte. El lema queda demostrado con esto.

En este punto, tenemos todos los elementos necesarios para completar la demostración. Consideremos un índice i, $1 \leq i \leq$ índice(a_{k-1}). Por el Lema 2.4,

$$\text{comps}(A_i) \geq \alpha_i + 1.$$

Esto, junto con el Lema 2.3, implica que

$$\text{comps}(B_i) \geq \alpha_i + 1.$$

Por cada componente adicional creada, hemos incluido un corte b_j (al haber seleccionado cortes con la multiplicidad apropiada), y por lo tanto, se deduce que el número de cortes b_j con índice $\leq i$ es al menos α_i. De esta manera, el teorema queda demostrado.

Observación. La demostración anterior muestra que cualquier conjunto de cortes que satisfaga la propiedad de unión producirá un k-corte casi óptimo. Esto explica por qué los cortes de Gomory-Hu funcionan. Claramente, la demostración dada es válida simultáneamente para cada valor de k entre 2 y n. Así, obtenemos un conjunto de k-cortes casi óptimos para $2 \leq k \leq n$. Obsérvese que en los casos extremos, es decir, para $k = 2$ y $k = n$, los cortes obtenidos son óptimos.

Teorema 2.5.

El algoritmo EFFICIENT encuentra un conjunto de k cortes, uno para cada valor de k, $2 \leq k \leq n$; cada corte está dentro de un factor de $(2 - \frac{2}{k})$ del k-corte óptimo. El algoritmo requiere un total de $n - 1$ cálculos de flujo máximo. Utilizando algoritmos eficientes de flujo máximo [2, 3], el algoritmo tiene un tiempo de ejecución de

$$O(mn^2 + n^{3+\epsilon}),$$

para cualquier $\epsilon > 0$ fijo.

11. Algoritmo SPLIT

Quizás la primera heurística que viene a la mente para encontrar un k-corte es la siguiente.

Algoritmo:

Comenzamos con el grafo dado. En cada iteración, seleccionamos el corte de menor peso en el grafo actual que divida una componente y eliminamos las aristas de dicho corte. Detenemos el proceso cuando el grafo actual tenga k componentes conexas.

Nótese que SPLIT, al igual que EFFICIENT, es también un algoritmo voraz. Mientras que EFFICIENT selecciona un corte de menor peso en el grafo original que crea componentes adicionales, SPLIT selecciona un corte de menor peso en el grafo actual.

SPLIT necesita encontrar un corte de peso mínimo en cada nueva componente formada, lo cual puede hacerse utilizando $n - 1$ cálculos de flujo máximo en un grafo con n vértices. Por lo tanto, SPLIT requiere $O(kn)$ cálculos de flujo máximo para encontrar un k-corte.

Estableceremos una garantía de desempeño de $(2 - \frac{2}{k})$ para SPLIT. Sin embargo, primero señalemos que ninguno de los dos algoritmos domina al otro. Consideremos el siguiente grafo con ocho vértices, $\{a, b, c, d, e, f, g, h\}$. Las aristas y sus respectivos pesos son los siguientes:

$$\begin{aligned} &\mathrm{wt}(a,b) = 3, \quad \mathrm{wt}(a,c) = 3, \quad \mathrm{wt}(b,d) = 7, \quad \mathrm{wt}(c,e) = 7, \\ &\mathrm{wt}(d,e) = 10, \quad \mathrm{wt}(d,f) = 4, \quad \mathrm{wt}(f,g) = 5, \quad \mathrm{wt}(g,h) = 5, \quad \mathrm{wt}(e,h) = 4. \end{aligned}$$

Ahora, para $k = 3$, los cortes encontrados por SPLIT y EFFICIENT tienen pesos 13 y 14, respectivamente, pero para $k = 4$, los pesos son 20 y 19, respectivamente.

Teorema 3.1.

El k-corte encontrado por el algoritmo SPLIT tiene un peso dentro de un factor de $(2 - \frac{2}{k})$ del óptimo.

Demostración. En el Teorema 2.1 mostramos que una variante ligeramente modificada de EFFICIENT, que selecciona cortes con la multiplicidad adecuada, tiene una garantía de desempeño de $(2-\frac{2}{k})$. Ahora probaremos que el peso del k-corte encontrado por SPLIT está acotado superiormente por el peso del k-corte encontrado por esta variante.

Sea $b_1, \ldots, b_{k-1}$ el conjunto de cortes seleccionados por la variante. Obsérvese que, dado que SPLIT selecciona un corte de peso mínimo en una componente, la divide en exactamente dos componentes. Por lo tanto, SPLIT selecciona exactamente $k-1$ cortes, denotados como $d_1, \ldots, d_{k-1}$.

Por inducción sobre i, demostraremos que $\text{wt}(d_i) \leq \text{wt}(b_i)$ para $1 \leq i \leq k-1$. La afirmación es claramente cierta para $i = 1$. Para demostrar el paso inductivo, primero observemos que

$$\text{comps}(d_1 \cup \cdots \cup d_i) = i + 1$$

y

$$\text{comps}(b_1 \cup \cdots \cup b_{i+1}) \geq i + 2.$$

Por lo tanto, existe un corte b_j, con $1 \leq j \leq i+1$, que no está contenido en $(d_1 \cup \cdots \cup d_i)$. Por la demostración del Lema 1.1, este corte creará componentes adicionales y estará disponible para SPLIT. De este modo, SPLIT seleccionará un corte con peso a lo sumo $\text{wt}(b_j)$. Como $\text{wt}(b_j) \leq \text{wt}(b_{i+1})$, la afirmación queda demostrada.

Referencias

[1] Saran, H., & Vazirani, V. V. (1995). Finding k cuts within twice the optimal. SIAM Journal on Computing, 24(1), 101-108.

[2] Goldberg, A. V., & Tarjan, R. E. (1988). A new approach to the maximum-flow problem. Journal of the ACM (JACM), 35(4), 921-940.

[3] King, V., Rao, S., & Tarjan, R. (1994). A faster deterministic maximum flow algorithm. Journal of Algorithms, 17(3), 447-474.

[4] Gomory, R. E., & Hu, T. C. (1961). Multi-terminal network flows. Journal of the Society for Industrial and Applied Mathematics, 9(4), 551-570.