Algoritmos de aproximación para k-corte mínimo

February 7, 2025

1 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.

2 Algoritmo EFFICIENT

Sea $G = (V, E)$ un grafo no dirigido y conexo, y sea $\text{wt} : E \to \mathbb{Z}^+$ 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 $\text{wt}(S)$. Para cualquier conjunto $S \subseteq E$, denotaremos el número de componentes conexas del grafo $G' = (V, E - S)$ como $\text{comps}(S)$.

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 $\text{comps}(B) > k$, podemos eliminar fácilmente aristas de B para obtener un corte B' tal que $\text{comps}(B') = k$.)

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.

2.1 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

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

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

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

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,

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

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

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

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

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, $1 \leq i \leq$ 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 $\text{comps}(A_i) \leq \text{comps}(B_i)$.

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

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.

3 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.

References

[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.