MA4701. Optimización Combinatorial. 2013.
Profesor: José Soto
Escriba(s): Sebastián Pérez y Gianmarco Sperone.
Fecha: 11 de Noviembre de 2013.

Cátedra 21

De la cátedra pasada teníamos que, para una función $f : 2^V \to \mathbb{R}$ modular y simétrica, la bifunción de adyacencia asociada:

$$d(A, B) = \frac{1}{2}(f(A) + f(B) - f(A \cup B)),$$

satisface que:

1. Es simétrica, es decir, $d(A, B) = d(B, A)$, $\forall\, A, B \subseteq V$.
2. $f(X) = d(X, V \setminus X) - 2f(\emptyset)$, $\forall\, X \subseteq V$.
3. Es monótona: $\forall\, A, B, C \subseteq V$ disjuntos se tiene que $d(A, B) \leq d(A, B \cup C)$.
4. Es consistente: $\forall\, A, B, C \subseteq V$ disjuntos se tiene que, si $d(A, B) \leq d(A, C)$ entonces $d(A \cup C, B) \leq d(A \cup B, C)$.

Además decíamos que $(s, t) \in V \times V$ es un *par colgante* de f si $f(\{s\}) \leq f(S)$, para todo $S \subseteq V$ que sea (s, t)-separador y esto es equivalente a decir que $d(\{s\}, V \setminus \{s\}) \leq d(S, V \setminus S)$, $\forall S \subseteq V$ (s, t)-separador. A continuación repasemos la demostración del teorema de Queyranne.

Teorema 1 (Queyranne)**.** *Si $(v_1, v_2, \ldots, v_{n-1}, v_n)$ es orden de máxima adyacencia de V, es decir, $d(\{v_1, \ldots, v_{i-1}\}, v_i) \geq d(\{v_1, \ldots, v_{i-1}\}, v_j), \forall i \geq 2, \forall j \geq i$, entonces (v_n, v_{n-1}) es par colgante.*

Demostración. Supondremos que $n \geq 4$, pues los casos restantes fueron estudiados la clase pasada. Sea $S \subseteq V$ un (v_{n-1}, v_n)-separador mínimo, es decir, que minimiza la cantidad $d(S, V \setminus S)$. Analicemos los posibles casos que pueden darse:

1. **Caso** 1: si S no separa v_1 de v_2, habíamos ya demostrado la clase pasada que $d(\{v_n\}, V \setminus \{v_n\}) \leq d(S, V \setminus S)$, lo cual asegura que (v_{n-1}, v_n) es par colgante.
2. **Caso** 2: si S no separa v_2 de v_3, fusionamos estos dos vértices en uno solo, a saber, w_{23}. Sea f_{23} la función submodular y simétrica en este nuevo sistema, y d_{23} la bifunción de adyacencia asociada. Probaremos que $(v_1, w_{23}, \ldots, v_n)$ es orden de máxima adyacencia para f_{23}. Para ello, lo que debemos probar es que $d_{23}(\{v_1\}, \{w_{23}\}) \geq d_{23}(\{v_1\}, \{v_j\})$, para todo $j \geq 4$, o equivalentemente, que:

   $$d(\{v_1\}, \{v_2, v_3\}) \geq d(\{v_1\}, \{v_j\}), \forall j \geq 4.$$

   En efecto, como $(v_1, v_2, \ldots, v_{n-1}, v_n)$ es orden de máxima adyacencia de f, se tiene que para cualquier $j \geq 4$:

   $$d(\{v_1\}, \{v_j\}) \leq d(\{v_1\}, \{v_2\}) \leq d(\{v_1\}, \{v_2, v_3\}),$$

   que es la desigualdad buscada. Por ende, si $n = 4$, entonces (v_4, w_{23}) será par colgante, y así, $d(\{v_4\}, V \setminus \{v_4\}) \leq d(S, V \setminus S)$. Por otra parte, si $n \geq 5$, por inducción tendremos que (v_{n-1}, v_n) es par colgante de d_{23}, y en tal caso podemos decir que $d_{23}(\{v_n\}, V \setminus \{v_n\}) \leq d_{23}(S, V \setminus S)$. Expandiendo adecuadamente estos conjuntos (para volver al sistema original), esta última desigualdad implica que $d(\{v_n\}, V \setminus \{v_n\}) \leq d(S, V \setminus S)$.
3. **Caso** 3: si S separa v_1 de v_2 y también separa v_1 de v_2, entonces S no separa v_1 de v_3. Ahora fusionamos v_1 con v_3 (en un nodo w_{13}) y afirmamos que $(v_2, w_{13}, v_4 \ldots, v_n)$ es orden de máxima adyacencia para f_{13}. Tal cual se hizo en el caso anterior, basta verificar que:

   $$d(\{v_2\}, \{v_1, v_3\}) \geq d(\{v_2\}, \{v_j\}), \forall j \geq 4.$$

   En efecto, dado $j \geq 4$, por monotonía se tiene que $d(\{v_2\}, \{v_j\}) \leq d(\{v_1, v_2\}, \{v_j\})$, y a su vez, por máxima adyacencia se cumple que $d(\{v_1, v_2\}, \{v_j\}) \leq d(\{v_1, v_2\}, \{v_3\})$. Finalmente, aplicando consistencia a la desigualdad siguiente:

   $$d(\{v_1\}, \{v_3\}) \leq d(\{v_1\}, \{v_2\}),$$

obtenemos que $d(\{v_1, v_2\}, \{v_3\}) \leq d(\{v_1, v_3\}, \{v_2\})$, lo cual concluye la demostración de la afirmación. Luego, los últimos dos elementos de $(v_2, w_{13}, v_4 \ldots, v_n)$ serán pares colgantes de f_{13}, implicando que:

$$d(\{v_n\}, V \setminus \{v_n\}) \leq d(S, V \setminus S).$$

Esto concluye la demostración del teorema.

□

Ahora veremos un algoritmo que nos permite encontrar $\emptyset \neq S \subsetneq V$ que minimice $f(S)$, con f una función submodular y simétrica. Para ello, supondremos que poseemos un *oráculo* que, para S dado, nos calcula $f(S)$ en tiempo $T(|V|)$. Veamos a continuación la versión recursiva y la versión iterativa del algoritmo.

1. Versión recursiva

Algorithm 1 mín(f, V)

```
if |V| = 2 then
    V = {v_1, v_2}.
    return {v_1}.
end if
if |V| ≥ 3 then
    Calcular (s, t) par colgante de f (usando máxima adyacencia).
    X' ← mín(f_st, V_st), en donde se ha tomado la fusión de s y t.
    Sea X el conjunto X' expandiendo s y t.
    if f(X) ≤ f({s}) then
        return X.
    else
        return {s}.
    end if
end if
```

2. Versión iterativa

Algorithm 2 mín(f, V)

```
cand ← ∅, f' ← f, V' ← V.
while |V| ≥ 2 do
    Calcular (s, t) par colgante del sistema (f', V').
    Sea S el conjunto obtenido de {s} expandiendo todos los elementos fusionados.
    cand ← cand{s}.
    (f', V') ← sistema donde s y t se fusionan.
end while
return el objeto S ∈ cand de menor f(S).
```

En general, estudiar la complejidad de un algoritmo recursivo no es tan fácil como estudiar la complejidad de un algoritmo iterativo, por eso es que se ha querido mostrar ambas implementaciones. Además, si $|V| = n$, entonces encontrar un par colgante toma $O(n^2)$ trabajo y $O(n^2)$ llamadas al oráculo, pues en una iteración hay que comparar elementos de un conjunto de n elementos, luego de $n-1$ y así sucesivamente hasta tener un único elemento, lo cual evidentemente tomará $O(n^2)$. Finalmente como son n iteraciones, se hace en total:

$$O(1) + O(n)O(n^2) + O(n) = O(n^3)(\text{trabajo y llamadas al oráculo}).$$

Podemos reescribir lo anterior como un teorema:

Teorema 2. *Podemos minimizar f submodular simétrica en tiempo polinomial ($O(n^3) + O(n^3)$ llamadas a f).*

Sea $\omega : E \to \mathbb{R}_0^+$ función de costo, encontrar un corte mínimo en un grafo no dirigido toma

$$O(n)O(\text{encontrar un par colgante}) = O(n)O(\text{encontrar orden de máxima adyacencia}).$$

Suponiendo que G es un grafo simple, veamos un algoritmo *malo*(no es que sea malo en si, sólo que a veces lo intuitivo no es la mejor implementación) para obtener orden de máxima adyacencia:

Algorithm 3 orden de máxima adyacencia para G no dirigido.

$v_i \leftarrow$ arbitrario.
$d(v) \doteq d(v, v_1) = \begin{cases} \omega(vv_1) & \text{si } vv_1 \in E \\ 0 & \text{si } vv_1 \notin E \end{cases}$.
$v_2 \leftarrow \text{argmax}(d(v))$.
for $j = 3, \ldots, n$ **do**
  for $v \in V \setminus \{v_1, \ldots, v_{j-1}\}$ **do**
    $d(v) \leftarrow d(v) + w(vv_{j-1})$.
  end for
  $v_j \leftarrow \text{argmax}(d(v))$.
end for
return $(v_1, \ldots, v_n)$.

Algunas observaciones del algoritmo anterior son:

- Este algoritmo tiene complejidad $O(n) + O(n)O(n) = O(n^2)$.
- Usando una mejor estructura de datos se puede mejorar a $O(m + n\log(n))$ si es que se usan *Heaps de Fibonacci* y $O(m\log(n))$ si es que se usan *Heaps binarios*.

Juntando lo anterior se obtiene el siguiente corolario,

Corolario 1. *Podemos calcular un corte mínimo en un grafo G no dirigido con pesos no negativos en:*

- $O(n^3)$ *si es que no se usa una estructura de datos compleja.*
- $O(n(m + \log(n)))$ *si es que se usa cola de prioridad.*

Hacemos a continuación las siguientes observaciones

- Sea $f : 2^V \to \mathbb{R}$ una función submodular. El algoritmo para minimizar f funciona cada vez que podamos encontrar pares colgantes de f. Por ejemplo, si f es suma de una función submodular simétrica y una función modular, la idea sería encontrar f' submodular simétrica con los mismo mínimos que f.
- El *mínimo globarl* de f submodular y simétrica no es interesante pues

$$\begin{aligned} 2f(\emptyset) &= f(\emptyset) + f(V) \\ &= f(A \cap (V \setminus A)) + f(A \cup (V \setminus A)) \\ &\leq f(A) + f(V \setminus A) \\ &= 2f(A). \end{aligned}$$

  Lo anterior es válido para cualquier $A \subseteq V$, por lo tanto $\emptyset$ es mínimo, usando la simetría de f se deduce que V también es mínimo. Por todo lo dicho anteriormente, es normal olvidarse de $\emptyset$ y V.

1. Programación lineal en tiempo polinomial

Supongamos que tenemos $G = (A \cup B, E)$ bipartito y $\omega : E \to \mathbb{R}$. Entonces queremos obtener un matching perfecto de costo mínimo en G. Podríamos resolver

$$\begin{aligned} &\text{mín } \omega^T x \\ P \text{ s.a. } &x(\delta(u)) = 1, \forall u \in V \\ &x \geq 0. \end{aligned}$$

El teorema de Birkhoff-Von-Neumann nos dice que P es integral, luego encontrar un matching perfecto de costo mínimo se puede hacer resolviendo el PL. Dicho PL puede ser resuelto usando *simplex*, el problema es que simplex no es un algoritmo polinomial, por ejemplo existen casos degenerados en el cual simplex recorre todos los vértices de un cubo d-dimensional (2^d vértices.), aunque en la práctica estos casos casi nunca ocurren.

En las cátedras siguientes veremos un método que fue el primer método polinomial para resolver PL, el llamado método de la elipsoide.