Clase 19

Sistemas de subconjuntos y matroides

Para muchos problemas hay una respuesta a la pregunta de por qué funcionan los algoritmos voraces.

Definición 19.1. Un *sistema de subconjuntos* es un conjunto $\mathscr{I}$ de subconjuntos de un conjunto $\mathscr{E}$ (el *conjunto base*) de modo que $\mathscr{I}$ es cerrado bajo inclusión.

O sea, si $\mathscr{B} \in \mathscr{I}$ y $\mathscr{A} \subseteq \mathscr{B}$ entonces $\mathscr{A} \in \mathscr{I}$. Note en particular que si $\mathscr{I} \neq \varnothing$, siempre es $\varnothing \in \mathscr{I}$.

El *problema de optimización* para un sistema de subconjuntos asigna un peso positivo a cada elemento de $\mathscr{E}$, y busca un conjunto $\mathscr{X} \in \mathscr{I}$ cuyo peso sea máximo en todos los conjuntos de $\mathscr{I}$. Es claro que podemos definir en forma afín la búsqueda de un mínimo. Retendremos esta definición para no complicar innecesariamente la discusión general. Parte de la discusión y los ejemplos que siguen vienen de Erickson [2].

Algunos ejemplos:

- Sea $\mathscr{E}$ un conjunto cualquiera de vectores de un espacio vectorial V, y sea $\mathscr{I}$ el conjunto de subconjuntos de $\mathscr{E}$ que son linealmente independientes. Claramente $\mathscr{I}$ es cerrado bajo inclusión.

  Por este caso es que al conjunto $\mathscr{I}$ se le suele llamar «de conjuntos independientes». En los casos de interés igualmente podremos identificar una relación de «dependencia» como acá.

- Considere:

  $$\mathscr{E} = \{a, b, c\}$$
  $$\mathscr{I} = \{\varnothing, \{a\}, \{b\}, \{a, b\}, \{b, c\}\}$$

  Podemos verificar directamente que es cerrado bajo inclusión. Una manera alternativa de describir $\mathscr{I}$ es como todos los conjuntos que no contienen a a y c.

- Sea $\mathscr{E}$ el conjunto de arcos de un grafo, y sean $\mathscr{I}$ los conjuntos de arcos que no comparten vértices (exactamente los conjuntos independientes del problema INDEPENDENT SET, también se les llama *matching* del grafo).

19.1. Algoritmo voraz genérico

Dado un sistema finito de subconjuntos $(\mathscr{E},\mathscr{I})$ hallamos un conjunto en $\mathscr{I}$ mediante el algoritmo 19.1. El algoritmo retorna un conjunto *maximal* (no se le pueden agregar elementos de $\mathscr{E}$

Algoritmo 19.1: Algoritmo voraz genérico

```
function Greedy(E, I)
    X ← ∅
    Ordene los elementos de E en orden de peso decreciente
    for x ∈ E do
        if X ∪ {x} ∈ I then
            X ← X ∪ {x}
        end
    end
    return X
end
```

sin salir de $\mathscr{I}$), pero no necesariamente *máximo* (no hay elementos de $\mathscr{I}$ de mayor peso). Nuestro problema de optimización pide un conjunto máximo.

Nuestro resultado es que hay una propiedad de sistemas de subconjuntos finitos que garantiza que el algoritmo voraz 19.1 da un conjunto maximal para todas las funciones de peso.

En nuestros ejemplos previos:

- Si consideramos los subconjuntos de $\{a,b,c\}$ que no tienen $\{a,c\}$ como subconjunto, y asignamos peso a los elementos, agregaremos b y aquél de $\{a,c\}$ de mayor peso.
- En los conjuntos de arcos que no forman ciclos, lo que tenemos es el problema *Maximal Weight Forest* (hallar el bosque de mayor peso que es subgrafo de G, abreviado MWF). Este problema es equivalente a MST, si el máximo peso de un arco en MST es m, asígnele peso $2m-w(e)$ al arco e. El algoritmo 19.1 para MWF aplicado a esto entrega un MST (es el algoritmo de Kruskal 17.2 disfrazado).

19.2. Matroides y algoritmos voraces

Diremos que un sistema de subconjuntos $(\mathscr{E},\mathscr{I})$ tiene la *propiedad de intercambio* si:

$$\forall \mathscr{A},\mathscr{B}\in\mathscr{I}, (|\mathscr{A}|<|\mathscr{B}|) \Longrightarrow (\exists e\in\mathscr{B}\setminus\mathscr{A} \text{ tal que } \mathscr{A}\cup\{e\}\in\mathscr{I}) \tag{19.1}$$

En estos términos:

Definición 19.2. Un *matroide* es un sistema de subconjuntos $M=(\mathscr{E},\mathscr{I})$ con la propiedad de intercambio.

Se les llama *conjuntos dependientes* (¡Sorpresa!) a los subconjuntos de $\mathscr{E}$ que no pertenecen a $\mathscr{I}$. El *rango* de $\mathscr{X}\subseteq\mathscr{E}$ es la cardinalidad de su máximo subconjunto independiente. Un conjunto independiente de dice que es una *base* de M si no es subconjunto de ningún conjunto independiente (es un conjunto independiente maximal). A un conjunto dependiente cuyos subconjuntos propios son todos independientes se le llama *circuito*.

Algunos ejemplos, varios de los cuales aparecerán nuevamente luego. Que son matroides quedará como ejercicio:

Matroide uniforme $U_{k,n}$: Un subconjunto $X \subseteq \{1,2,\ldots,n\}$ es independiente si y solo si $|X| \leq k$.

Todo subconjunto de k elementos es una base; todo conjunto de $k+1$ elementos es un circuito.

Matroide gráfico $\mathscr{M}(G)$: Sea $G = (V,E)$ un grafo. Un subconjunto de E es independiente si no contiene ciclos.

Una base del matroide es un árbol recubridor de G; un circuito es un ciclo en G.

Matroide cográfico $\mathscr{M}^*(G)$: Sea $G = (V,E)$ un grafo. Un subconjunto $I \subseteq E$ es independiente si el subgrafo complementario $(V, E \setminus I)$ es conexo.

Una base del matroide es el complemento de un árbol recubridor de G; un circuito es un *cociclo* de G, un conjunto mínimo de arcos que desconecta a G.

Matroide de correspondencias: Sea $G = (V,E)$ un grafo. Un conjunto $I \subseteq V$ es independiente si y solo si hay un *matching* (conjunto de arcos que no tienen vértices en común) que los cubre.

Caminos disjuntos: Sea $G = (V,E)$ un grafo dirigido, y sea s un vértice fijo de G. Un subconjunto $I \subseteq V$ es independiente si y solo si hay caminos que no comparten arcos desde s a cada elemento de I.

El resultado general es de Radó y Edmonds [1]:

Teorema 19.1 (Rado-Edmonds)**.** *Dado un sistema de subconjuntos $(\mathscr{E},\mathscr{I})$, las siguientes son equivalentes:*

1. *El algoritmo voraz 19.1 entrega una solución óptima para toda función de peso*
2. *El sistema de subconjuntos es un matroide*

Demostración. Demostramos implicancia en ambas direcciones.

Demostramos que si $M = (\mathscr{E},\mathscr{I})$ es un matroide entonces el algoritmo voraz entrega un óptimo por contradicción. Sea $\mathscr{A} = \{a_1, a_2, \cdots, a_k\}$ la solución entregada por el algoritmo voraz, y sea $\mathscr{B} = \{b_1, b_2, \cdots, b_{k'}\}$ una solución óptima, donde suponemos $w(\mathscr{B}) > w(\mathscr{A})$. Primero, $k = k'$, ya que si fuera $k' \neq k$ por la propiedad de intercambio podríamos agregar un elemento del conjunto mayor al otro. Esto o contradice la optimalidad de $\mathscr{B}$ o contradice el que el algoritmo terminó con $\mathscr{A}$. Luego, podemos suponer que los elementos de $\mathscr{A}$ y $\mathscr{B}$ se listan en orden de mayor a menor (en $\mathscr{A}$ es el orden en que los incluyó nuestro algoritmo), y consideremos el mínimo s tal que $w(b_s) > w(a_s)$. Sean los subconjuntos:

$$\alpha = \{a_1, \cdots, a_{s-1}\}$$
$$\beta = \{b_1, \cdots, b_s\}$$

Por ser parte de $\mathscr{I}$ los conjuntos $\mathscr{A}$ y $\mathscr{B}$, también son parte de $\mathscr{I}$ los conjuntos α y β. Por la propiedad de intercambio, hay t con $1 \leq t \leq s$ tal que $b_t \in \beta \setminus \alpha$ y $\alpha \cup \{b_t\} \in \mathscr{I}$. Pero $w(b_t) \geq w(b_s) > w(a_s)$, y nuestro algoritmo hubiese preferido b_t a a_s.

Al revés, si el algoritmo voraz siempre entrega un óptimo entonces $(\mathscr{E},\mathscr{I})$ es un matroide. Para esto basta demostrar que el algoritmo voraz puede no entregar un óptimo si $(\mathscr{E},\mathscr{I})$ no es un matroide. Si $(\mathscr{E},\mathscr{I})$ no es un matroide, entonces:

$$\exists \mathscr{A}, \mathscr{B} \in \mathscr{I}, (|\mathscr{A}| < |\mathscr{B}|) \wedge (\nexists e \in \mathscr{B} \setminus \mathscr{A} \text{ tal que } \mathscr{A} \cup \{e\} \in \mathscr{I})$$

Sean $m = |\mathscr{A}|$ y $n = |\mathscr{E}|$. Defina:

$$w(e) = \begin{cases} m+2 & e \in \mathscr{A} \\ m+1 & e \in \mathscr{B} \setminus \mathscr{A} \\ 1/(2n) & \text{caso contrario} \end{cases}$$

El algoritmo voraz retorna $\mathscr{A}$, con peso a lo más $m(m+2)+1/2 = m^2+2m+1/2$; una solución mejor es $\mathscr{B}$ con peso al menos $(m+1)^2 = m^2+2m+1$. □

Otra propiedad interesante resulta de lo siguiente:

Definición 19.3. Un sistema de subconjuntos $(\mathscr{E}, \mathscr{I})$ tiene la *propiedad de cardinalidad* si:

$$\forall \mathscr{E}' \subseteq \mathscr{E}, (A, B \in \mathscr{I} \text{ subconjuntos maximales de } \mathscr{E}') \implies (|A| = |B|) \tag{19.2}$$

Decimos que $A \in \mathscr{I}$ es *subconjunto maximal* de $\mathscr{E}'$ si $A \subseteq \mathscr{E}'$ y no hay $a \in \mathscr{E}'$ tal que $A \cup \{a\} \in \mathscr{I}$. Con esto tenemos:

Teorema 19.2 (Propiedad de cardinalidad)**.** *Sea un sistema de subconjuntos $(\mathscr{E}, \mathscr{I})$. Entonces $(\mathscr{E}, \mathscr{I})$ es un matroide si y solo si cumple la propiedad de cardinalidad.*

Demostración. Es un si y solo si, demostramos implicancia en ambas direcciones.

Sean A, B subconjuntos maximales de $\mathscr{E}' \subseteq \mathscr{E}$. Debemos demostrar $|A| = |B|$, cosa que haremos por contradicción. Supongamos $|A| < |B|$, por la propiedad de intercambio:

$$\exists e \in B \setminus A, (A \cup \{e\} \in \mathscr{I})$$

Note que $A \cup \{e\} \in \mathscr{E}'$ ya que $e \in B \subseteq \mathscr{E}'$, o sea, A no sería maximal. El caso $|A| > |B|$ es simétrico.

Al revés, debemos demostrar que si $(\mathscr{E}, \mathscr{I})$ no es matroide entonces hay $\mathscr{E}'$ y $A, B \in \mathscr{I}$ con A, B maximales en $\mathscr{E}'$ con $|A| \neq |B|$. Si $(\mathscr{E}, \mathscr{I})$ no es matroide:

$$\exists A, C \in \mathscr{I}, |A| < |C| \wedge \nexists e \in C \setminus A \text{ con } A \cup \{e\} \in \mathscr{I}$$

Defina $\mathscr{E}' = A \cup C$, note que A es maximal en $\mathscr{E}'$. Hay $B \in \mathscr{I}$ tal que $C \subseteq B$ y B es maximal en $\mathscr{E}'$. Pero $|B| \geq |A| + 1$, como debíamos demostrar. □

Tenemos dos propiedades diferentes (intercambio y cardinalidad) que describen los matroides.

Note que nuestro primer ejemplo de algoritmo voraz (programar observaciones de ALMA) tiene un sistema de subconjuntos natural asociado: dos tareas son independientes si no traslapan (conjuntos independientes son los INDEPENDENT SET del grafo en el cual cada observación es un vértice, y dos vértices están unidos si traslapan). Esto *no* es un matroide, pueden haber conjuntos independientes maximales de tamaños distintos. Nuestro algoritmo voraz entrega un óptimo por la elección de función de peso (todos iguales), para otras funciones de peso falla.

Algunos ejemplos de matroides y los algoritmos voraces correspondientes:

- Los subconjuntos de un conjunto finito $\mathscr{U}$ de cardinalidad a lo más k.

  Podemos hallar el subconjunto más pesado usando el algoritmo voraz.

- Matroide de columnas. Sea **A** una matriz, $\mathscr{E} = \{\mathbf{x}\colon \mathbf{x} \text{ es columna de } \mathbf{A}\}$, y sea $\mathscr{I}$ los conjuntos de columnas linealmente independientes.

  Podemos hallar la base más pesada para las columnas de **A** usando el algoritmo voraz.

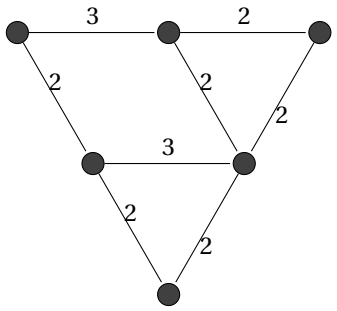
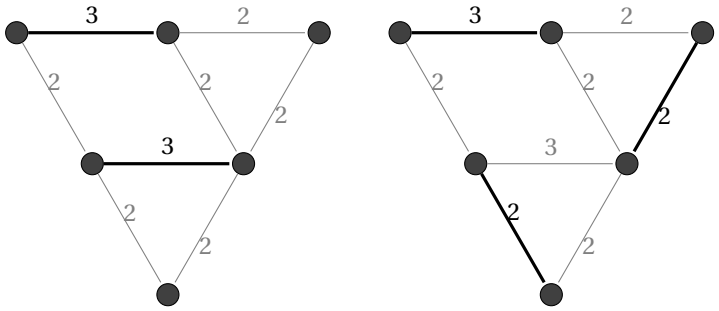


Figura 19.1 – Un grafo con arcos rotulados



(a) Resultado del algoritmo voraz

(b) Un óptimo

Nuestro último ejemplo de sistema de subconjuntos (arcos independientes en un grafo) no es un matroide, y el algoritmo voraz no siempre da un máximo. Considere el grafo de la figura 19.1. Al buscar un conjunto de arcos de máximo peso que no comparten vértices, el algoritmo voraz da el conjunto maximal que consta de los arcos de peso 3, para un total de 6 (ver 19.2a); pero el máximo es 7 (ver 19.2b).

La teoría de matroides nació de consideraciones sobre conjuntos linealmente independientes de vectores, y se vio aplicable a teoría de grafos. Rápidamente se extendió a otras áreas, un ejemplo es el texto de Lawler [3] sobre optimización combinatoria y matroides.

19.3. Programación con plazos fatales

Suponga que debe completar n tareas en n días, donde cada tarea demanda un día completo de dedicación. Cada tarea tiene un plazo fatal, si se completa después de su plazo fatal hay un costo a pagar. Se busca el orden en el cual ejecutar las tareas de forma de pagar el mínimo costo.

Formalmente, numeramos las tareas de 1 a n, dados un arreglo de plazos fatales D (un entero entre 1 y n) y uno de penalizaciones P (números reales no negativos). Un programa π es una permutación de $\{1,2,\ldots,n\}$. Buscamos un programa π que minimice:

$$\sum_{1\le i\le n} P[i]\cdot[\pi(i) > D[i]]$$

No parece para nada un ejemplo de optimización de matroide, allí solicitan un subconjunto y buscamos una permutación. Sorprendentemente, hay un matroide disfrazado en él. Para el programa π, diga que las tareas para las cuales $\pi(i) > D[i]$ están *atrasadas*, las demás *a tiempo*. La

observación trivial de que el costo de una programación queda determinada por sus tareas a tiempo lleva a revelar el matroide.

Llame *realista* a un conjunto de tareas X tal que hay una programación π en la que todas las tareas de X se completan a tiempo. Podemos caracterizar los subconjuntos realistas de la siguiente forma. Sea $X(t)$ el conjunto de tareas en X con plazo fatal en o antes de t:

$$X(t) = \{i \in X : D[i] \leq t\}$$

En particular, $X(0) = \varnothing$ y $X(n) = X$.

Proposición 19.1. *Sea $X \subseteq \{1,2,\ldots,n\}$ un conjunto arbitrario de tareas. Entonces X es realista si y solo si $|X(t)| \leq t$ para todo t.*

Demostración. Es un si y solo si, demostramos implicancia en ambas direcciones.

Sea π una programación en la que todas las tareas de X están a tiempo. Sea i_t la t-ésima de X a completar. Por un lado, $\pi(i_t) \geq t$, ya que completamos $t-1$ tareas antes; por el otro, $\pi(i_t) \leq D[i_t]$ ya que i_t está a tiempo. Concluimos $D[i_t] \geq t$, por lo que $|X(t)| \leq t$.

Suponga ahora que $|X(t)| \leq t$ para todo t. Si ejecutamos las tareas de X en orden de plazo fatal, completamos las tareas con plazo fatal a más tardar t el día t. Para todo $i \in X$ estamos completando i antes de $D[i]$, X es realista. □

Llamemos *canónica* una programación para el conjunto de tareas X en la cual se ejecutan las tareas de X en orden de plazo fatal creciente, y las demás tareas en orden arbitrario. La proposición 19.1 nos dice que X es realista si y solo si todas sus tareas se completan a tiempo en su programación canónica. O sea, nuestro problema se puede reformular como hallar un subconjunto realista X que maximice:

$$\sum_{i \in X} P[i]$$

Estamos buscando un subconjunto óptimo.

Proposición 19.2. *La colección de conjuntos realistas es un matroide.*

Demostración. El conjunto vacío es realista (vacuamente), todo subconjunto de un conjunto realista es obviamente realista. Resta demostrar que se cumple la propiedad de intercambio. Sean entonces X e Y conjuntos realistas, con $|X| > |Y|$.

Sea t^* el máximo entero tal que $|X(t^*)| \leq |Y(t^*)|$. Debe existir, ya que $|X(0)| = 0 \leq 0 = |Y(0)|$ mientras $|X(n)| = |X| > |Y| = |Y(n)|$. Por la definición de t^*, hay más tareas con plazo fatal t^*+1 que t^* en X que en Y. O sea, podemos elegir $j \in X \setminus Y$ con plazo fatal t^*+1. Llamemos $Z = Y \cup \{j\}$.

Sea t arbitrario. Si $t \leq t^*$, entonces $|Z(t)| = |Y(t)| \leq t$, ya que Y es realista. Por otro lado, si $t > t^*$, $|Z(t)| = |Y(t)| + 1 \leq |X(t)|$ por la definición de t^* y dado que X es realista. Por la proposición 19.1, Z es realista. Se cumple la propiedad de intercambio. □

Por la proposición 19.2, nuestro problema de hallar la programación óptima es una optimización de matroide, el algoritmo voraz 19.2 da detalles. Falta determinar si un conjunto de tareas es realista. La proposición 19.1 da una pista cómo hacerlo, dando el algoritmo 19.3. Esto supone que X viene ordenado por plazo fatal:

$$i \leq j \implies D[X[i]] \leq D[X[j]]$$

El resultado se ejecuta en tiempo $O(n^2)$, usando estructuras de datos apropiadas esto se reduce a $O(n \log n)$. Detalles quedan de ejercicio.

Algoritmo 19.2: Algoritmo voraz para programar tareas

```
Ordene P en orden decreciente, y ordene D acorde
j ← 0
for i ← 1 to n do
    X[j + 1] ← i
    if X[1..j + 1] es realista then
        j ← j + 1
    end
end
Retorne la programación canónica de X[1..j]
```

Algoritmo 19.3: Determinar si un conjunto es realista

```
function realista(X, D)
    N ← 0
    j ← 0
    for t ← 1 to n do
        if D[X[j]] = t then
            N ← N + 1; j ← j + 1
            if N > t then
                return F
            end
        end
    end
    return T
end
```

Ejercicios

1. Demuestre que los ejemplos de sistemas de subconjuntos citados realmente lo son. ¿Son matroides?

2. Demuestre que el matroide gráfico $\mathscr{M}(G)$ es un matroide para todo grafo G. Describa sus bases y circuitos.

3. Demuestre que para todo grafo G el matroide cográfico $\mathscr{M}^*(G)$ es un matroide.

4. Demuestre que para todo grafo G el matroide de correspondencias es un matroide.
   Pista: ¿Qué es la diferencia simétrica de dos correspondencias?

5. Indique qué entrega el algoritmo voraz en cada ejemplo de matroide citado.

6. Sea G un grafo. Un conjunto de ciclos $\{c_1, c_2, \ldots, c_k\}$ de G se llama *redundante* si cada arco de G aparece en un número par de c_i. Un conjunto de ciclos es *independiente* si no contiene subconjuntos redundantes. Un conjunto maximal de ciclos independientes es una *base de ciclos* de G.

   a) Sea C una base de ciclos de G. Demuestre que para cada ciclo γ de G, hay un subconjunto $A \subseteq C$ tal que $A \cap \{\gamma\}$ es redundante. O sea, γ es el «o exclusivo» de los ciclos de A.

   b) Demuestre que la colección de conjuntos de ciclos independientes es un matroide.

c) Suponga ahora que cada arco de G tiene un peso, que el peso de un ciclo es la suma de los pesos de sus arcos, y que el peso de un conjunto de ciclos es la suma de los pesos de los ciclos (note que arcos que se repiten se cuentan cada vez que aparecen). Describa y analice un algoritmo eficiente para hallar una base de ciclos de mínimo peso. (Esto no es sencillo, no es inmediato obtener los ciclos de G).

7. Demuestre que el sistema de subconjuntos del problema de programar observaciones de ALMA no es un matroide. Dé un ejemplo con una función de peso (retorno de la observación) tal que el algoritmo voraz no entregue una programación óptima.

8. Demuestre cómo programar tareas con plazo fatal en tiempo $O(n \log n)$.

   Pista: Use una estructura que permita determinar si $X \cup \{i\}$ es realista y agregar i a X en tiempo $O(\log n)$ cada operación.

Bibliografía

[1] Jack Edmonds: *Matroids and the greedy algorithm*. Mathematical Programming, 1(1):127–136, December 1971.

[2] Jeff Erickson: *Algorithms, etc.* http://jeffe.cs.illinois.edu/teaching/algorithms, January 2015. Department of Computer Science, University of Illinois at Urbana-Champaign.

[3] Eugene L. Lawler: *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.