# 3 Algorismes d'aproximació

3.1. El problema del Tall Maxim (MAX-CUT). Donat un graf no dirigit $G = (V, E)$, el problema del maximum cut (MAX-CUT) es trobar la partició $S \cup \bar{S}$ de $V$ tal que maximitze el nombre d'arestes entre $S$ i $\bar{S}$ ($C(S, \bar{S})$ ,on $S \subseteq V$ i $\bar{S} = V - S$. La maximització entre totes les possibles particiones de $V$. El problema del MAX-CUT és NP-hard en general. Donat

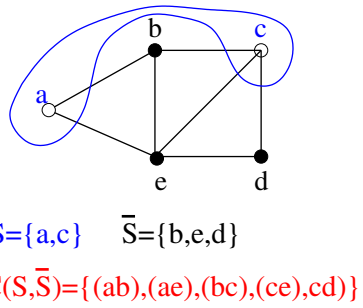


Figura 1: Exemple de MAX-CUT, amb cost òptim 5

$G = (V, E)$ considereu el següent algorisme golafre (greedy) pel problema del MAX-CUT:
Ordeneu els vèrtexs en ordre decreixent respecte al seu grau (nombre veïns). Considereu el resultat de l'ordenació s'escriu com a $(v_1, v_2, \ldots, v_n)$.
Inicialment tenim $S = \emptyset = \bar{S}$. Al primer pas $v_1 \in S$. Al pas $i$-èsim col.loquem $v_i$ a $S$ o $\bar{S}$ de manera que maximitze $|C(S, \bar{S})|$.

(a) Demostreu la correctessa i doneu la complexitat de l'algorisme golafre.

(b) Demostreu que aquest algorisme golafre és una 2-aproximació al MAX-CUT

**A solution**

(a) El grau d'un vertex es com am molt $n$. Podem ordenar en orde decreixent de grau en temps $O(n)$ fent servir counting sort. A cada pas, l'algorisme accedeix només a la llista del seus veïns per comtar quants son a cada costat de la partició i decidir a on s'afegeix. El cost total es $O(n + m)$.

Per una altre part l'algorisme sempre produeix una partició, per tant es correcte.

(b) Per a cada aresta $e_i = (u_i, v_i)$ definim un *vèrtex responsable* (de $e_i$) com el primer vertex d'entre $u$ o $v$ que tracta l'algorisme. Sigui $r_i$ el nombre d'arestes de les que $v_i \in V$ és responsable. Com cada aresta te exactament un vèrtex responsable, aleshores $\sum_{i=1}^{n} r_i = m$.

Quan l'algorisme tracta el vèrtex $v_i$, estem afegint al menys $r_i/2$ arestes al tall $C$. Notem que si el conjunt (d'entre $S$ i $\bar{S}$) que conte el màxim nombre de ($u_j$s als $r_i$ arestes que

tenen $v_i$ com responsable). Aquest conjunt ha de tenir com a mínim $r_i/2$ veïns a l'altre costat.

Per tant, nombre d'arestes al $C$ final $\geq \sum_{i=1}^{n} r_i/2 \geq m/2$. Com tenim un problema de maximització, el $S^*$ optim és $\leq m$. Pertant l'algorisme proposat es una 2-aproximació.

**A solution**

(a) El algoritmo es correcto ya que acaba computando una partición en dos conjuntos de los vértices del grafo, obteniendo así una solución factible. Podemos ordenar los vértices del grafo en tiempo $O(n)$ usando, por ejemplo counting sort, ya que el grado es un valor entre 0 y $|V|$. Para decidir dónde colocar un vértice es suficiente con consultar la lista de vecinos del vértice actual. Como tratamos todos los vértices el coste del algoritmo es $O(n + m)$

(b) Para analizar la tasa de aproximación, llamemos $\mathsf{opt}(G)$ al número de aristas en un max-cut de $G$ i $c(G)$ al número de aristas en el corte obtenido por el algoritmo.

En un corte nunca podemos tener más que el total de las arista, por lo que tenemos que $\mathsf{opt}(G) \leq |E|$.

El algoritmo trata los vértices en orden, podemos pensar que estamos descubriendo el grafo en a medida que añadimos nuevos vértices. En el paso $i$ añadimos el vértice $v_i$ y las aristas $(v_i, v_j) \in E$ con $j < i$. Llamemos $E_i$ a este conjunto de aristas, es decir $E_i = \{(v_i, v_j) \mid j < i\}$. Observemos que $\{E_i\}_{i=1}^{n}$ es una partición de $E$, ya que cada arista queda asignada de forma única al conjunto correspondiente al vértice con índice mayor.

En el paso $i$-ésimo, asignamos $v_i$ a un conjunto de manera que el número de aristas de $E_i$ en el corte es mayor que el número de aristas de $E_i$ que no forman parte del corte. Así la contribución de $v_i$ a $c(G)$ es mayor o igual que $|E_i|/2$.

Si sumamos para todos los vértices tenemos que $c(G) \geq \sum_{i=1}^{n} |E_i|/2 = |E|/2$, ya que $\{E_i\}_{i=1}^{n}$ es una partición de $E$.

Por otra parte $\mathsf{opt}(G) \leq |E|$, por lo que tenemos que $c(G) \geq \mathsf{opt}(G)/2$. Al ser un problema de maximización $c(G) \leq \mathsf{opt}(G)$, poniendo todo junto tenemos que

$$\frac{1}{2} \leq \frac{opt(G}{c(G)} \leq 2$$

por lo que el algoritmo propuesto es una 2-aproximación.

3.2. Consider the following algorithms that have as input a boolean formula in 3-CNF

Algorithm A

(a) Let $c_0$ be the number of true clauses when all variables are set to value 0.

(b) Sea $c_1$ be the number of true clauses when all variables are set to value 1.

(c) return the maximum among $c_0$ and $c_1$.

Is there a constant $r$ for which the algorithms is a $r$-approximations for MaxSat ?

**A solution**

- En el problema MaxSat el valor del óptimo es el máximo número de clausulas satisfechas por una asignación. Supongamos que la fórmula $F$ utiliza $n$ variables y está formada por $m$ clausulas, cada clausula con 3 literales. Tenemos $\mathsf{opt}(F) \leq m$, en el caso mejor $F$ es satisfactible.
- Los dos algoritmos se ejecutan en tiempo polinómico y proporcionan el número de clausulas satisfechas por una asignación. Podríamos adaptar el algoritmo para que devolviese la asignación en vez de su coste, manteniendo coste polinómico.
- Algorithm A. La asignación que pone todas las variables a 0 satisface todas las clausulas con, al menos, una negación y la de todos 1's las que tienen, al menos, una variable sin negar. Por lo tanto $c_0 + c_1 \geq m$. Como consecuencia $\max\{c_0, c_1\} \geq m/2$. Por lo que el algoritmo A es una 2-aproximación.
- Algorithm B. Para calcular la esperanza utilizaremos variables indicadoras $C_j$ es la variable indicadora con valor 1 si la clausula $j$, $1 \leq j \leq m$, es satisfecha por la asignación seleccionada. El número de clausulas satisfechas es $\sum_{1 \leq j \leq m} C_i$. Por la linealidad de la esperanza, el número esperado de clausulas satisfechas es $\sum 1 \leq j \leq m E[C_j]$.

  La probabilidad de que una clausula con 3 literales no sea satisfecha es $(1/2)^3 = 1/8$, no acertamos la asignación correcta para ninguna de sus variables. Por tanto $E[C_j] = 7/8$. Lo que nos da como valor esperado $\frac{7}{8}m$. Combinándolo con la cota superior, tenemos que Algorithm B es una 8/7 aproximación.

3.3. Consider the following algorithm:

**function** EDGES($G$:graph)
  $U = \emptyset$; $E = E(G)$;
  **while** $E \neq \emptyset$ **do**
    select any edge $e = (u, v) \in E$;
    $U = U \cup \{u, v\}$;
    $E = E - \{e' \in E \mid e' \text{ incident to } e\}$
  **return** $U$

Prove that Edges is a 2-approximation for the Minimum vertex cover problem.

**A solution** El algoritmo tiene coste polinómico. Cada vez que se elimina una arista del grafo es debido a que está cubierta por un vértice en $U$. El algoritmo acaba cuando no quedan aristas por cubrir. Por lo tanto, el algoritmo calcula un vertex cover.

Si miramos las aristas elegidas en el primer paso del while, vemos que entre ellas no hay ningún tipo de conexión, o lo que es lo mismo forman un matching en el grafo $G$. Cualquier vertex cover que cubra un matching tiene que tener al menos un vértice por cada arista del matching, mientras que Edges se queda con los dos. Por lo tanto, $opt(G) \geq \mathsf{Edges}(G)/2$.

Como es un problema de minimización $\mathsf{opt}(G) \leq \mathsf{Edges}(G)$. Poniendolo todo junto tenemos que Edges es una 2-aproximación al Minimum vertex cover problem.

3.4. Consideramos el siguiente escenario: tenemos un conjunto de $n$ ciudades con distancias mínimas entre ellas (verifican la desigualdad triangular). Queremos seleccionar un subconjunto $C$ de $k$ ciudades en las que queremos ubicar un centro comercial. Asumiendo que las personas que viven en una ciudad compraran en el centro comercial más próximo se quiere buscar una ubicación $C$ de manera que todas las ciudades tengan un centro comercial a distancia menor que $r$ en la que intentamos minimizar $r$ sin perder cobertura. Para ello diremos que $C$ es un $r$-recubrimiento si todas las ciudades están a distancia como mucho $r$ de una ciudad en $C$. Sea $r(C)$ el mínimo $r$ para el que $C$ es un $r$-recubrimiento. Nuestro objetivo es encontrar $C$ con $k$ vértices para el que $r(C)$ es mínimo.

(a) Demuestra que si $k \geq n$, la solución formada por todas las ciudades es óptima.

A partir de ahora asumiremos que $k \leq n$.

(b) Suponiendo que $S$ es el conjunto de ciudades, considera el siguiente algoritmo:

Seleccionar cualquier ciudad $s \in S$ y define $C = \{s\}$
**while** $|C| \neq k$ **do**
    seleccionar una ciudad $s \in S$ que maximiza la distancia de $s$ a $C$;
    $C = C \cup \{s\}$
**return** C

Demuestra que es un algoritmo de aproximación polinómico con tasa de aproximación 2.

**A solution**

(a) En este caso, la solución que cubre todas las ciudades es un 0-cover, cualquier distancia es igual o mayor a 0. Por lo tanto no lo podremos mejorar y ha de ser un óptimo.

(b) El algoritmo es:

$r = 0$;
$S = \{s\}$ ▷ $s$ es una ciudad cualquiera
**for** $s \in S$ **do**
    **if** $s \notin C$ **then**
        $r' = \infty$;
        **for** $c \in C$ **do**;
            $r' = \min(r', dist(s, c))$;
        $r = \max r, r'$;
**return** $r$

Por cada ciudad $s \in S$, en el caso de que no pertenezca al conjunto $C$, calculamos su distancia con $C$, definida como $\min_{c \in C} d(s, c)$. Nos quedamos siempre con la mayor de las distancias obtenidas.

El coste del bucle interno es $O(k)$ y se ejecuta $n - k$ veces. Por lo tanto el coste total del algoritmo es $O(nk)$.

(c) Si partimos de un grafo $G = (V, E)$, podemos calcular las distancias $d(u, v)$ entre cualquier par de vértices. Considerando como conjunto de ciudades $V$ y la distancia $d$, para un valor de $k$, $r(C)$ es igual a 1 si y solo sí tenemos una selección de vértices para

la que todos los demás vértices están a distancia $\leq 1$. Lo que es equivalente a decir que la selección es un conjunto dominador de tamaño $k$ en $G$. Dominating Set es un problema NP-completo, por lo tanto si pudiésemos resolver nuestro problema en tiempo polinómico P=NP, por lo que el problema es NP-difícil.

(d) El algoritmo voraz propuesto tiene coste polinómico. Sea $C^*$ la selección óptima, sea $r = r(C^*)$. Al finalizar el algoritmo, $C$ es una selección de $k$ ciudades, por tanto $r(C) \leq r$. Para tener una 2-aproximación tenemos que demostrar que $r(C) \leq 2r$.

Supongamos que no es cierto, es decir $r(C^*) < r(C)/2$. Es decir, tenemos una ciudad $s$ tal que $d(s, C) > 2r$.

Consideremos una iteración del algoritmo en el que $C'$ es el conjunto de centros seleccionados y en la que seleccionamos un nuevo centro $c'$. Vamos a analizar la distancia de $c'$ a $C'$,

$$d(c', C') \geq d(s, C') \geq d(s, C) > 2r$$

Así, al finalizar el algoritmo, cualquier par de centros en $C$ están a distancia $> 2r$.

Por cada ciudad $c_i \in C$, consideramos las ciudades $C_i$ a distancia $\leq 2r$. Como al finalizar el algoritmo, cualquier par de centros en $C$ están a distancia $> (2r)$, los conjuntos $C_i$ son disjuntos.

Además, como $r < r(C)/2$, en cada $C_i$ tenemos que tener un elemento de $C^*$. Como $|C| = |C^*|$, cada $C_i$ contiene exactamente un único elemento de $C^*$ al que llamamos $c_i^*$. Además, $c_i^*$ es el centro en $C^*$ que está más cerca de $c_i$, $d(c_i, c_i^*) \leq r$

Para una ciudad cualquiera, $s$ sea $c_i^*$ su centro mas cercano en $C^*$. Tenemos

$$d(s, C) \leq d(s, c_i) \leq d(s, c_i^*) + d(c_i, c_i^*) \leq 2r.$$

Con lo que llegamos a una contradicción.

3.5. *Planificació.*

Tenim un conjunt de $n$ tasques. La tasca $i$ es descriu per un parell $T_i = (s_i, d_i)$ on $s_i$ es el seu temps de disponibilitat i $d_i$ la seva duració. L'objectiu es planificar totes les tasques en un processador de manera que: (a) cap tasca s'executa abans del seu temps de disponibilitat, (b) les tasques s'executen sense interrumció, i (d) el temps de finalització del procesament de totes les tasques sigui mínim.

Se sap que, quan és possible interrompre qualsevol tasca en execució i reiniciar-le més tard des del punt d'aturada, l'algorisme voraç que executa en cada punt de temps la tasca (disponible) més propera al seu temps de finalització, aconsegueix el desitjat mínim. Anomenarem a aquest algorisme Voraç1.

Considereu el procediment següent:

(a) Executa Voraç1.

(b) Ordena les tasques en l'ordre ascendent del seu temps de finalització d'acord amb la solució proporcionada per versió Voraç1.

(c) Les tasues s'executeu en aquest ordre i sense interrupció, afegint el temps d'espera necesari fins que la tasca estigui disponible.

Demostreu que aquest algorisme es una 2-aproximació pel problema plantejat.

**A solution**

- El algoritmo tiene coste polinómico, ya que Voraz1 se puede implementar en tiempo $O(n \log n)$ utilizando una cola de prioridad en la que almacenaremos las tareas no acabadas con clave el tiempo que falte para su finalización.
- Voraz1 nos da una solución óptima para un problema con menos restricciones, por lo que el coste será menor o igual que el de la solución óptima del problema $\mathsf{opt}(x) \leq Voraz1(x)$
- En la planificación proporcionada por Voraz1 únicamente no se ejecutará ninguna tarea en instantes en los que no hay ninguna disponible. Por lo tanto $Voraz1(x)$ es la suma de las duraciones de todas las tareas más un tiempo $V \geq 0$ en el que no haya tareas disponibles y no se ejecuta nada.

$$Voraz1(x) = V + \sum_i d_i$$

- En el caso de que ninguna tarea fuese interrumpida, el orden de ejecución y fi nalización de las tareas será el mismo, por lo que el coste en $A(x)$ sería igual que en $Voraz1(x)$. El coste será mayor únicamente si ocurre una interrupción en $Voraz1(x)$.
- Consideremos un bloque de tareas que ejecuta Voraz1 sin ningún tiempo en el que no hay tareas disponibles. Voraz1 ejecuta la tarea con menos duración restante para ser finalizada. Cualquier tarea que se ejecute en este bloque estará disponible al finalizar la ejecución del bloque. Por lo que una vez llegado este punto se pueden ejecutar todas las tareas en el orden dado por Voraz1 sin problemas. Por lo tanto $A$ no puede tardar más del doble en finalizar la ejecución de este tareas siguiendo el orden establecido por Voraz1.

- Poniendo todo junto tenemos

$$A(x) \leq V + \sum_i 2d_i \leq 2V + 2\sum_i d_i \leq 2V oraz 1(x) \leq 2\mathsf{opt}(x)$$

y A es una 2-aproximación.

3.6. Donat com a entrada un graf no dirigit $G = (V, E)$, definim el problema de GST (graf sense triangles) com el problema de seleccionar el màxim nombre d'arestes $E$, de manera que el graf $G' = (V, E')$ no contingui cap triangle (i.e. no hi han 3 vèrtexs $x, y, z$ tal que $(x, y), (x, z)$ i $(y, z)$ siguin arestes a $G'$). Aquest problema és NP-hard.

Donat $G = (V, E)$ considereu el següent algorisme golafre (greedy):
Ordeneu els vèrtexs en ordre decreixent respecte al seu grau (nombre veïns). Considereu el resultat de l'ordenació s'escriu com a $(v_1, v_2, \ldots, v_n)$.
Inicialment tenim $S = \emptyset = \bar{S}$. Al primer pas $v_1 \in S$. Al pas $i$-èsim col.loquem $v_i$ a $S$ o $\bar{S}$ de manera que maximitze $|C(S, \bar{S})|$. Prenem com $E' = \{(u, v) \mid u \in S, v \notin S\}$

Aquest algorisme, és una 2-aproximació polinòmica al problema?.

**A solution** Es el mismo algoritmo que para el problema 1 (MaxCut), quedándonos al fianl con las aristas del corte que como hemos visto tiene al menos $m/2$ aristas. El algoritmo se ejecuta en tiempo polinómico. El grafo correspondiente a las aristas del corte es un grafo bipartido, que por supuesto no tiene triángulos, el algoritmo nos da una solución a este problema.

Además el número optimo de aristas a remover está acotado superiormente por $m$. Por lo que tenemos una 2-aproximación.

3.7. Consider the Max Clique problem. Given an undirected graph $G = (V, E)$ compute a set of vertices that induce a complete subgraph with maximum size.

For each $k \geq 1$, define $G^k$ to be the undirected graph $(V^k, E^k)$, where $V^k$ is the set of all ordered $k$-tuples of vertices from $V$. $E^k$ is defined so that $(v_1, \ldots, v_k)$ is adjacent to $(u_1, \ldots, u_k)$ iff, for $1 \leq i \leq k$, either $(v_i, u_i) \in E$ or $v_i = u_i$.

- (a) Prove that the size of a maximum size clique in $G^k$ is the $k$-th power of the corresponding size in $G$.
- (b) Argue that if there is a constant approximation algorithm for Max Clique, then there is a polynomial time approximation schema for the problem.

**A solution**

- (a) Prove that the size of a maximum size clique in $G^k$ is the $k$-th power of the corresponding size in $G$.

  Si $C$ es un clique en $G$, vamos a ver que $C^k$ es un clique en $G^k$.

  Si consideramos $(c_1, \ldots, c_k), (c'_1, \ldots, c'_k) \in C^k$, para $1 \leq i \leq k$, si $c_i \neq c'_i$, tenemos $(c_i, c'_i) \in E$ ya que $C$ es un clique en $G$. Por lo tanto $((c_1, \ldots, c_k), (c'_1, \ldots, c'_k)) \in E(G^k)$. Como $|C^k| = |C|^k$ tenemos que $\mathsf{opt}(G)^k \leq \mathsf{opt}(G^k)$.

  Sea $D$ un clique de tamaño máximo en $G^k$. Para $1 \leq i \leq k$, consideramos el conjunto $D_i = \{d_i \mid (d_1, \ldots, d_i, \ldots, d_k) \in D\} \subseteq V$ formado por todos los vértices que aparecen en la componente $i$-ésima de algún elemento de $D$. Si tomamos $u, v \in D_i$ tienen que aparecer en dos elementos de $D$ diferentes que están conectados en $G^k$, por lo tanto $(u, v) \in E$. Así $D_i$ es un clique en $G$ y además que $D_1 \times \cdots \times D_k$ es un clique en $G^k$, con el

  Sea $D_{\max}$ el conjunto con más elementos de entre $D_1, \ldots, D_k$, como $D_{\max}$ es un clique en $G$ tenemos que $D^k_{\max}$ es un clique en $G^k$.

  Por otra parte, $D \subseteq D_1 \times \cdots \times D_k$ y además

  $$|D| \leq |D_1 \times \cdots \times D_k| \leq |D_{\max}|^k.$$

  Pero $D$ es un clique de tamaño máximo en $G^k$, por lo que

  $$|D| = |D_{\max}|^k.$$

  Como $D_{\max}$ es un clique en $G$ tenemos

  $$\mathsf{opt}(G^k) = |D| = |D_{\max}|^k \leq \mathsf{opt}(G)^k.$$

- (b) Argue that if there is a constant approximation algorithm for Max Clique, then there is a polynomial time approximation schema for the problem.

  Sea $\mathcal{A}$ una $p$-aproximación a Max Clique. Utilizo la notación del apartado anterior para definir el esquema de aproximación.

  $\mathcal{EA}(G, r)$
  $k = \frac{\log p}{\log r}$
  Compute $G^k$
  $D = \mathcal{A}(G^k)$
  **return** $D_{\max}$

  De acuerdo con el apartado anterior $\mathcal{EA}$ devuelve un clique en $G$.

  Primero veamos es una $r$-aproximación.

  Como $\mathcal{A}$ es una $p$-aproximación tenemos

  $$p \geq \frac{\mathsf{opt}(G^k)}{\mathcal{A}(G^k)} = \frac{\mathsf{opt}(G)^k}{\mathcal{A}(G^k)}$$

  Por el apartado anterior sabemos que $D^k_{\max} \geq \mathcal{A}(G^k)$. Por tanto $\mathcal{EA}(G)^k \geq \mathcal{A}(G^k)$. Combinando con la desigualdad anterior tenemos

  $$p \geq \frac{\mathsf{opt}(G^k)}{\mathcal{A}(G^k)} = \frac{\mathsf{opt}(G)^k}{\mathcal{A}(G^k)} \geq \frac{\mathsf{opt}(G)^k}{\mathcal{EA}(G)^k}.$$

  Lo que nos da, teniendo en cuenta la definición de $k$,

  $$\frac{\mathsf{opt}(G)}{\mathcal{EA}(G)} \leq \sqrt[k]{p} = r.$$

  Analicemos ahora el coste del algoritmo.

  $G^k$ tienen tamaño $O(n^{2k})$, ya que tiene $n^k$ vértices y como mucho$O(n^{2k})$ aristas. Para construir la matriz de adyacencia necesitamos $O(k)$ comprobaciones por elemento, esto nos da un coste $O(kn^{2k})$. Ejecutar $\mathcal{A}$ tiene coste polinómico en $n^{2k}$. Finalmente, calcular $D_{\max}$ nos cuesta como mucho $O(|D|) \leq O(n^k)$.

  En total nos queda coste $O(p(n^{2k}))$. Teniendo en cuenta que $p$ es una valor constante, $k = O(\frac{1}{\log r})$ y cuando $r$ tiende a 1, $\frac{1}{\log r} = \Theta(\frac{1}{r-1}) = O(\frac{r}{r-1})$. Cuando $r \to 1$ el coste es $O(p(n^{\frac{r}{r-1}})$ que es polinómico en $n$ para $r$ constante a 1, que es polinómico en $n$ pero exponencial en $\frac{r}{r-1}$. Por lo que tendríamos un PTAS.

3.8. Consider the MAXIMUM COVERAGE problem: Given sets $S_1, \ldots, S_m$ over a universe of elements $U = \{1, \ldots n\} = \cup_{i=1}^{n} S_i$ and a positive integer $k$. Choose $k$ sets that cover as many elements as possible, that is

$$opt(x) = \max\{|\cup_{i\in I} S_i| \mid |I| = k\}.$$

Provide a greedy approximation algorithm with rate $\frac{e}{e-1} \approx 1.58$.

**A solution** El algoritmo greedy, selecciona en cada paso el conjunto con más elementos de $U$ todavía sin cubrir. Hasta que ha seleccionado $k$ conjuntos.

El algoritmo calcula una solución al problema ($k$ conjuntos) en tiempo polinómico.

Para analizar el factor de aproximación analizaremos el número de elementos que se cubren en cada iteración de una solución óptima.

Notación:

- $C$ es el conjunto cubierto por una solución óptima, i.e. con tamaño $\mathsf{opt}$.
- $a_i$ es el número de nuevos elementos de cubiertos en la iteración $i$ del voraz.
- $b_i = \sum_{i=1}^{i} a_i$ números total de elementos cubiertos en la iteración $i$.
- $c_i = \mathsf{opt} - b_i$.

La propiedad básica que aplicaremos es la siguiente, si $X \subseteq U$ se puede cubrir con $k$ conjuntos, siempre existe un $i$ tak que $|X \cup S_i| \geq |X|/k$. Si, todos tuviese menos de $|X|/k$ elemento de $X$ es imposible cubrir $X$ con $k$ conjuntos.

Aplicaremos esta propiedad de forma iterativa lo largo del proceso. Si tenemos $c_i$ elementos sin cubrir, sabemos que queda un conjunto $S$ que cubre al menos $c_i/k$ elementos. Por lo tanto $a_i \geq c_i/k$, ya que elegimos el conjunto que cubre más elementos nuevos.

Vamos a demostrar por inducción que $c_{i+1} \leq (1 - \frac{1}{k})^{i+1}\mathsf{opt}$.

El caso base es $i = 1$:

$$c_1 = \mathsf{opt} - b_1 = \mathsf{opt} - a_1 \leq \mathsf{opt} - \frac{\mathsf{opt}}{k} \leq (1 - \frac{1}{k})\mathsf{opt}.$$

La hipótesis de inducción: $c_i \leq (1 - \frac{1}{k})^i\mathsf{opt}$. Notemos que $c_i = \mathsf{opt} - b_i = \mathsf{opt} - \sum_{j=1}^{i} a_j$ y que por tanto $c_{i+1} = c_i - a_{i+1}$, así

$$c_{i+1} = c_i - a_{i+1} \leq c_i - \frac{c_i}{k} = (1 - \frac{1}{k})c_i \leq (1 - \frac{1}{k})^{i+1}\mathsf{opt}.$$

Tenemos que $c_k \leq (1 - \frac{1}{k})^k\mathsf{opt} \approx \frac{\mathsf{opt}}{e}$.

El número de elementos cubiertos por el voraz es $b_k$, pero

$$b_k = \mathsf{opt} - c_k \geq \mathsf{opt} - \frac{\mathsf{opt}}{e} = (1 - \frac{1}{e})\mathsf{opt}.$$

3.9. The Barcelona Diagonal University campus is shared among the UPC, UB, CSIC, and other public and private entities. As a result there are many surveillance cameras placed outside the buildings whose images can be accessed by different surveillance companies and in some cases by different surveillance teams inside the company. Each camera has a unique surveillance team that can visualize the images on the spot. The UPC Chancellor needs to establish a surveillance plan to be set during the next Chancellor elections. The aim of the plan is to guarantee that the voting is done without pressure.

To establish the plan the UPC Committee has set a collection of *surveillance locations* that require visual surveillance. For each of those locations, they have collected the surveillance teams that have cameras visualizing the location. Fortunately all the surveillance locations can be covered by an already existing camera, so the task can be performed without additional infrastructure.

Due to the economical situation, the UPC Chancellor wants to expend the minimum possible amount of money. The administrative office has negotiated and agreed the surveillance cost with each of the surveillance teams. Now they are seeking a solution that allows to keep surveillance on all the surveillance locations with minimum total cost. Assume, for this exercise, that a surveillance team receives a fix amount of money for looking to the images of all the cameras assigned to it.

Consider the algorithm that repeatedly chooses the team that cover the maximum number of still uncovered locations.

Is this algorithm a constant factor approximation for the problem? Justify its correctness and efficiency.

**A solution** El algoritmo es claramente correcto. Una solución es un conjunto de equipos de vigilancia que cubre todas las cámaras. Este algoritmo voraz selecciona cada vez el equipo que mayor número de cámaras puede controlar, y se acaba cuando todas las cámaras han sido asignadas, por lo tanto, siempre devuelve una solución correcta.

El coste de este algoritmo viene principalmente dado por el coste de ordenar los equipos en función del número de cámaras de las que pueden encargarse. Podemos implementarlo con una cola de prioridad para mantener el orden cada vez que eliminemos elementos del conjunto de cámaras que quedan por asignar. Por lo tanto el coste será de $n\log(n)$ por la ordenación inicial. El coste de la asignación es el de eliminar elementos del conjunto de cámaras que quedan. Esto es $O(nm)$, donde $n$ es el número de equipos y $m$ el de cámaras. Por lo tanto, el coste es $O(mn + nlog(n)) = O(nm)$.

Abstrayendo el problema que tenemos que resolver la entrada es un conjunto $C$ (el de cámaras) y una familia $F$ de subconjuntos de $C$ (un por cada equipo de vigilancia). Lo que hay que calcular es $E \subseteq F$ que cubra $C$ con tamaño mínimo. Este problema es SET COVER. De acuerdo con http://www.csc.kth.se/ viggo/wwwcompendium/node146.html el problema no es approximable en $c\log n$, para algún $c > 0$. El algoritmo voraz no nos proporciona tiene factor constante salvo que P = NP.

3.10. The association for the promotion of the European Identity is planning a workshop formed by a series of debates over different European topics. They have a *participant's* list formed by the persons that have committed to participate in the workshop provided the association issues them a formal invitation.

The organizing committee in view of the planned topics and previous experiences has selected for each debate two disjoint lists of people: the *success* list and the *failure* list. The committee is considering those list in an optimistic perspective. The presence of at least one of the persons in the success list or the absence of at least one of the persons in the failure list of a particular debate guarantees that this debate will be successful.

The organizing committee faces the problem of selecting the subset of people to whom the association will send a formal invitation. The association wish to invite a set of people that guarantees that the number of successful debates (according to the previous rule) is maximized.

Design a 2-approximation algorithm for the problem. Justify its correctness and efficiency.

**A solution** Vamos a formalizar el problema. Para cada persona tenemos que decidir si la invitamos o no. Podemos representar a una persona con una variable lógica, a cierto – invitar, a falso – no invitar. Y la selección de invitados con una asignación lógica a las variables

Un debate lo podemos ver como una clausula (disyunción) en las que las variables de las personas en la success list aparecen "afirmadas" y las variables en la "failure list" aparecen negadas. Con esta interpretación un debate es succesful si la asignación satisface la fórmula.

Así maximizar el número de debates es lo mismo que maximizar el número de clausulas satisfechas. Podemos usar la 2-aproximación a MaxSat que ya hemos visto.

3.11. Considereu una formula booleana sobre un conjunt de $n$ variables en 3-CNF, totes les clausules tenen 3 literals diferents. Una 3-MCNF és una formula en 3-CNF on, a més, cap literal és la negació d'una variable.

Per suposat una 3-MCNF sempre admet una asignació de valors a les variables que la satisfà. Per això considerem el problema min-3-MCNF què donada una formula booleana $F$ en 3-MCNF ha de trobar una asignació de valor a les variables que satisfaci la formula amb el nombre més petit possible de variables a 1.

(a) Considereu l'algorisme següent:

```
1: procedure G-3-MCNF(F)
2:     F en 3-MCNF amb clausules C = {C_1, ..., C_m} over variables x_1, ..., x_n
3:     x_i = false, per 1 ≤ i ≤ n
4:     while C ≠ ∅ do
5:         Extreu una clausula C de C
6:         Sigui x_i une de les variables a C
7:         x_i = true
8:         Treu de C totes les clausules que continguin x_i
9:     Return x
```

És G-3-MCNF un algorisme d'aproximació constant per min-3-MCNF?

(b) Doneu una 3-aproximació per min-3-MCNF.

**A solution**

(a) El algoritmo no proporciona un factor de aproximación constante. Basta considerar una entrada en la que tenemos $n+1$ variables $x_0, \ldots, x_n$ y $n$ cláusulas $C_1, \ldots, C_n$, dónde $C_i = (x_i, x_{i+1\%n}, x_0$. Si el algoritmo al procesar la cláusula $i$ elige la variable $x_i$, acabará con una asignación con $n$ 1's. Por otra parte, todas las clausulas se pueden satisfacer asignando 1 a $x_0$ y 0 al resto de variables. La asignación óptima tiene solo un 1.

(b) Para conseguir un algoritmo de aproximación con factor 3 utilizo una idea similar a la de uno de los algoritmo voraces de aproximación para vertex cover.

```
1: procedure G-3-MCNF-2(F)
2:     F en 3-MCNF amb clausules C = {C_1, ..., C_m} over variables x_1, ..., x_n
3:     x_i = false, per 1 ≤ i ≤ n
4:     while C ≠ ∅ do
5:         Extreu una clausula C de C
6:         Assignem 1 a totes les variables a C
7:         Treu de C totes les clausules que continguin una de les variables a C
8:     Return x
```

El algoritmo propuesto claramente tiene coste polinómico. El conjunto de cláusulas seleccionadas por el algoritmo no comparten ninguna variable entre ellas. Por tanto en la solución óptima al menos una de sus variables tiene que estár asignada a 1. Como el algoritmo asigna las 3 a uno, tenemos que el número de 1's en la asignación obtenida por nuestro algoritmo es como mucho 3 veces el optimo. Por lo que tenemos una 3-aproximación.