# Algorithmes approchés

VLADY RAVELOMANANA

IRIF – UMR CNRS 8243
Université de Paris 7
vlad@irif.fr

**– M1 Algorithmique avancée –**

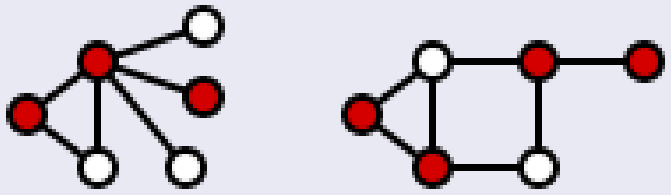
# Plan de ce cours

- Algorithmes approchés.
  - Approximation et schémas.
  - Limite de l'approximabilité.
  - Non approximabilité.
  - Complexité paramétrique.

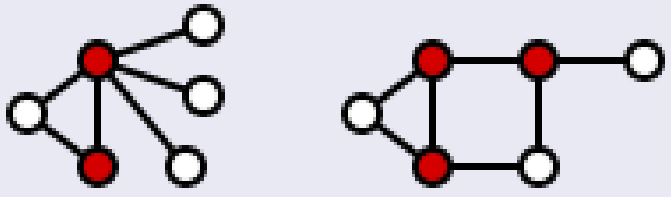
# Le problème de couverture par sommets ou “Vertex Cover”

## Définition.

Une **couverture par sommets**, ou **transversale** d’un graphe $G$ est un ensemble de sommets tel que chaque arête de $G = (V, E)$ est incidente à au moins un sommet de ce sous-ensemble (cf. les sommets en rouge dans le dessin):



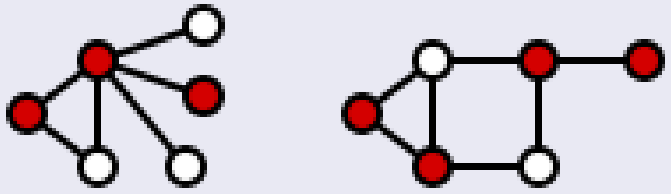
Un tel ensemble est dit **minimale** si la propriété de couverture est violée lorsqu’on lui enlève un sommet:



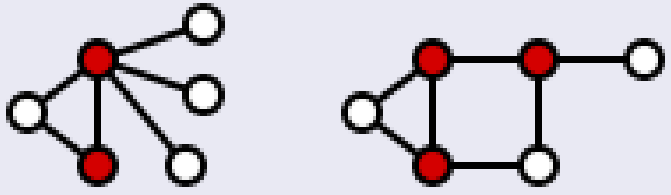
# Le problème de couverture par sommets ou “Vertex Cover”

## Définition.

Une **couverture par sommets**, ou **transversale** d'un graphe $G$ est un ensemble de sommets tel que chaque arête de $G = (V, E)$ est incidente à au moins un sommet de ce sous-ensemble (cf. les sommets en rouge dans le dessin):



Un tel ensemble est dit **minimale** si la propriété de couverture est violée lorsqu'on lui enlève un sommet:



## Le problème algorithmique.

**Input:** Un graphe $G = (V, E)$.
**Output:** Un ensemble (de cardinalité) `minimum` de sommets couvrant toutes les arêtes.

# Un glouton

## NP-complétude.

Difficile de trouver l'optimum efficacement.

# Un glouton

## NP-complétude.

Difficile de trouver l'optimum efficacement.

## Un algorithme glouton.

res $:= \{\}$;
$A := \{$tous les sommets de $G\} = V$;
**while** $A \neq \emptyset$ **do**
    Prendre un sommet $s$ dans $A$;
    res $:=$ res $\bigcup \{s\}$;
    Supprimer $s$ et tous ses voisins dans $A$;
**end**
**retourner**(res);

# Un glouton

## NP-complétude.

Difficile de trouver l'optimum efficacement.

## Un algorithme glouton.

res $:= \{\}$;
$A := \{$tous les sommets de $G\} = V$;
**while** $A \neq \emptyset$ **do**
    Prendre un sommet $s$ dans $A$;
    res $:=$ res $\bigcup \{s\}$;
    Supprimer $s$ et tous ses voisins dans $A$;
**end**
**retourner**(res);

## Pire instance.

Si le graphe est une étoile, cet algorithme peut renvoyer un ensemble de cardinalité $n - 1$ sommets.
On veut une **`certaine garantie`** par rapport à l'optimum.

# Un algorithme approché.

## Méthodologie générale.

Trouver une quantité comparable $Q$ par rapport à la solution optimale $S_{\mathrm{OPT}}$ et si possible montrer que:

- $c.Q \leq S_{\mathrm{OPT}} \leq C.Q$ pour **deux constantes absolues** $c$ **et** $C$.

# Un algorithme approché.

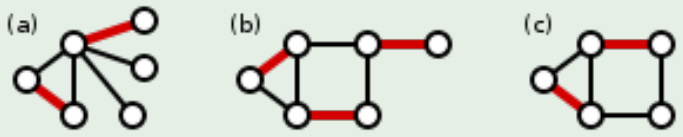
## Méthodologie générale.

Trouver une quantité comparable $Q$ par rapport à la solution optimale $S_{\text{OPT}}$ et si possible montrer que:

- $c.Q \leq S_{\text{OPT}} \leq C.Q$ pour **deux constantes absolues** $c$ **et** $C$.

## Couplage dans un graphe

Un **couplage** ou **appariement** (“**matching**”) d’un graphe est un ensemble d’arêtes de ce graphe qui n’ont pas de sommets en commun. Un couplage $M$ est dit **maximal** si toute arête du graphe possède une extrémité commune avec une arête de $M$. Un couplage **`maximum`** est un couplage contenant le plus grand nombre possible d’arêtes.




## Matching vs. $S_{\text{OPT}}$

- Une **borne inférieure**: si $M$ est un matching et $S$ un couvrant alors $|M| \leq |S|$ donc $|M| \leq |S_{\text{OPT}}|$.
- Une **borne supérieure**: si $M$ est un matching *maximal* alors ses sommets forment un couvrant. On a $|S_{\text{OPT}}| \leq 2|M|$.

# Un autre glouton

## GreedyMaximalMatching

res $:= \{\}$;
$A := \{$toutes les arêtes de $G\} = E$;
**while** $A \neq \emptyset$ **do**
    Prendre une arête $a$ dans $A$;
    res $:=$ res $\bigcup \{a\}$;
    Supprimer $a$ et toutes les arêtes couvertes par ses deux extrémités;
**end**
**retourner**(res);

# Un autre glouton

## GreedyMaximalMatching

res $:= \{\}$;
$A := \{$toutes les arêtes de $G\} = E$;
**while** $A \neq \emptyset$ **do**
    Prendre une arête $a$ dans $A$;
    res $:=$ res $\bigcup \{a\}$;
    Supprimer $a$ et toutes les arêtes couvertes par ses deux extrémités;
**end**
**retourner**(res);

**Théorème.**

L'algorithme `GreedyMaximalMatching` permet de trouver une solution approchée $s$ au problème du couverture par sommets telle que $|s| \leq 2|S_{OPT}|$

# Preuve

On peut exécuter l'algorithme suivant:

**(i)** trouver un *couplage maximal* avec `GreedyMaximalMatching`,

**(ii)** renvoyer les deux extrémités des arêtes comme sommets couvrant.

# Preuve

On peut exécuter l'algorithme suivant:

**(i)** trouver un *couplage maximal* avec `GreedyMaximalMatching`,

**(ii)** renvoyer les deux extrémités des arêtes comme sommets couvrant.

**Proposition 1.** Les appels (i) et (ii) renvoient bien des sommets couvrant.
**Preuve.** (i) renvoie un couplage maximal $M$ par construction. Pour (ii), on montre que si une arête $e$ n'est pas couverte par un sommet alors $M \bigcup \{e\}$ est un couplage qui contredit le fait que $M$ soit maximal.

# Preuve

On peut exécuter l'algorithme suivant:

**(i)** trouver un *couplage maximal* avec `GreedyMaximalMatching`,

**(ii)** renvoyer les deux extrémités des arêtes comme sommets couvrant.

**Proposition 1.** Les appels (i) et (ii) renvoient bien des sommets couvrant.
**Preuve.** (i) renvoie un couplage maximal $M$ par construction. Pour (ii), on montre que si une arête $e$ n'est pas couverte par un sommet alors $M \bigcup \{e\}$ est un couplage qui contredit le fait que $M$ soit maximal.
**Proposition 2.** Soit $M$ la solution renvoyée par `GreedyMaximalMatching`, on a $|M| \leq |S_{OPT}|$.
**Preuve.** Par définition, une solution au problème de couverture par sommets doit couvrir chaque arête dans $M$, en particulier ceci est valable pour la **solution optimale**.

# Preuve

On peut exécuter l'algorithme suivant:

**(i)** trouver un *couplage maximal* avec `GreedyMaximalMatching`,

**(ii)** renvoyer les deux extrémités des arêtes comme sommets couvrant.

**Proposition 1.** Les appels (i) et (ii) renvoient bien des sommets couvrant.
**Preuve.** (i) renvoie un couplage maximal $M$ par construction. Pour (ii), on montre que si une arête $e$ n'est pas couverte par un sommet alors $M \bigcup \{e\}$ est un couplage qui contredit le fait que $M$ soit maximal.
**Proposition 2.** Soit $M$ la solution renvoyée par `GreedyMaximalMatching`, on a $|M| \leq |S_{\mathrm{OPT}}|$.
**Preuve.** Par définition, une solution au problème de couverture par sommets doit couvrir chaque arête dans $M$, en particulier ceci est valable pour la **solution optimale**.

**Algorithme approché.**

On a un algorithme qui retourne un ensemble de sommets $s$ tel que $|s| = 2 \times |\{\text{arêtes renvoyées}\}| \leq 2|S_{\mathrm{OPT}}|$.

**Remarque.**

A ce jour, cet algorithme est le meilleur algorithme d'approximation pour le problème du couverture par sommets.

**Remarque.**

A ce jour, cet algorithme est le meilleur algorithme d'approximation pour le problème du couverture par sommets.

**Exercice:**

Pour le montrer, considérer le graphe biparti complet $K_{n,n}$.

**Remarque.**

A ce jour, cet algorithme est le meilleur algorithme d'approximation pour le problème du couverture par sommets.

**Exercice:**

Pour le montrer, considérer le graphe biparti complet $K_{n,n}$.

Considérons le graphe complet biparti $K_{n,n}$ avec $n$ sommets bleus et $n$ `sommets rouges`.
La taille de tout couplage maximal dans ce graphe est $n$: $|M| = n$.
Ainsi,

- l'appel `GreedyMaximalMatching`$(K_{n,n})$ retourne un ensemble de $n$ arêtes
- puis le fait de prendre les 2 extrémités renvoie $2n$ sommets comme sommets couvrant.
- Mais clairement, l'optimal est $n$.

# Plan de ce cours

- Algorithmes approchés.
- Approximation et schémas.
- Limite de l'approximabilité.
- Non approximabilité.
- Complexité paramétrique.

# Algorithmes approchés: définition

## Position du problème algorithmique.

Trouver parmi **toutes les solutions** d'un problème une qui `optimise` une fonction $f$.

# Algorithmes approchés: définition

## Position du problème algorithmique.

Trouver parmi **toutes les solutions** d'un problème une qui **optimise** une fonction $f$.

## Définition.

Un algorithme est dit **$\varepsilon$-approché** s'il donne une solution $S$ qui satisfait

$$\frac{f(S_{\text{opt}}) - f(S)}{f(S_{\text{opt}})} \leq \varepsilon \qquad \text{SI } S_{\text{OPT}} \text{ MAXIMISE } f$$

et

$$\frac{f(S) - f(S_{\text{opt}})}{f(S)} \leq \varepsilon \qquad \text{SI } S_{\text{OPT}} \text{ MINIMISE } f\,.$$

(**on veut un $\varepsilon$ aussi proche de 0 que possible!**)

On dit qu'un algorithme **APPROCHE L'OPTIMUM** à un **facteur** $\varphi$ si

$$\max\left(\frac{f(S)}{f(S_{\text{opt}})}, \frac{f(S_{\text{opt}})}{f(S)}\right) \leq \varphi\,.$$

(**on veut un $\varphi$ aussi proche de 1 que possible!**)

L'ensemble d'arêtes $M$ couvrant fournit par **`GreedyMaximalMatching`** vérifie $|M| \leq 2|S_{\text{opt}}|$ donc (minimisation ici)

$$\varepsilon = \frac{|M| - |S_{\text{opt}}|}{|M|} \leq \frac{|S_{\text{opt}}|}{|M|} \leq \frac{1}{2}$$

mais aussi

$$\varphi = \frac{|M|}{|S_{\text{opt}}|} \leq 2\,.$$

C'est un algorithme $\frac{1}{2}$**-approché** ou une **approximation à un facteur** 2.

# Algorithmes approximable et nouvelles classes

## La classe APX.

La classe des algorithmes approximables à un facteur constant (une constante absolue) est noté **APX**.

Le couplage glouton fournit une approximation à un facteur 2 pour le problème de couverture des sommets (qui donc appartient à la classe **APX**).

# Algorithmes approximable et nouvelles classes

## La classe APX.

La classe des algorithmes approximables à un facteur constant (une constante absolue) est noté **APX**.

Le couplage glouton fournit une approximation à un facteur 2 pour le problème de couverture des sommets (qui donc appartient à la classe **APX**).

## Les classes PTAS, EPTAS et FPTAS.

Un problème admet un **schéma d'approximation polynomiale** si pour tout $\varepsilon$ il admet un algorithme $\varepsilon$-approché de complexité polynomial pour un $\varepsilon$ donné. La classe de ces problèmes est notée **PTAS** (pour " Polynomial-Time Approximation Scheme"). De plus, un schéma d'approximation est dit efficace si la complexité est POLYNOMIAL EN LA TAILLE DE L'ENTRÉE **sans dépendre d'**$\varepsilon$. La classe de ces problèmes est appelée **EPTAS**. Enfin, le schéma d'approximation est totalement polynomial si la complexité est POLYNOMIALE À LA FOIS EN LA TAILLE DE L'ENTRÉE ET DE $\varepsilon$ . La classe correspondante est **FPTAS**.

Par exemple, pour un problème $P$ sur une instance de taille $n$, on a un algorithme $A$ tel que pour tout $\varepsilon$, $A$ est un algorithme $\varepsilon$-approché de complexité $O(n^{\text{constante}/\varepsilon})$.

# Algorithmes approximable et nouvelles classes

## La classe APX.

La classe des algorithmes approximables à un facteur constant (une constante absolue) est noté **APX**.

Le couplage glouton fournit une approximation à un facteur 2 pour le problème de couverture des sommets (qui donc appartient à la classe **APX**).

## Les classes PTAS, EPTAS et FPTAS.

Un problème admet un **schéma d'approximation polynomiale** si pour tout $\varepsilon$ il admet un algorithme $\varepsilon$-approché de complexité polynomial pour un $\varepsilon$ donné. La classe de ces problèmes est notée **PTAS** (pour " Polynomial-Time Approximation Scheme"). De plus, un schéma d'approximation est dit efficace si la complexité est POLYNOMIAL EN LA TAILLE DE L'ENTRÉE **sans dépendre d'**$\varepsilon$. La classe de ces problèmes est appelée **EPTAS**. Enfin, le schéma d'approximation est totalement polynomial si la complexité est POLYNOMIALE À LA FOIS EN LA TAILLE DE L'ENTRÉE ET DE $\varepsilon$ . La classe correspondante est **FPTAS**.

Par exemple, pour un problème $P$ sur une instance de taille $n$, on a un algorithme $A$ tel que pour tout $\varepsilon$, $A$ est un algorithme $\varepsilon$-approché de complexité $O(n^{\text{constante}/\varepsilon})$.

## Ranking des classes.

On a par construction FPTAS $\subseteq$ EPTAS $\subseteq$ PTAS $\subseteq$ APX.

# Le problème du sac à dos

## Définition.

**Input/donnée:** $n$ pairs $(c_i, w_i)$ d'entiers et un entier $K$. On peut interpréter $c_i$ comme étant le coût d'un objet numéro $i$ et $w_i$ son volume/poids. $K$ est la capacité totale du sac à dos.
**Le problème d'optimisation:** trouver $\text{Sol} \subseteq \{1, 2, \cdots, n\}$ tel que $\sum_{i \in \text{Sol}} w_i \leq K$ et $\sum_{i \in \text{Sol}} c_i$ est maximum.

# Le problème du sac à dos

## Définition.

**Input/donnée:** $n$ pairs $(c_i, w_i)$ d'entiers et un entier $K$. On peut interpréter $c_i$ comme étant le coût d'un objet numéro $i$ et $w_i$ son volume/poids. $K$ est la capacité totale du sac à dos.
**Le problème d'optimisation:** trouver $\text{Sol} \subseteq \{1, 2, \cdots, n\}$ tel que $\sum_{i \in \text{Sol}} w_i \leq K$ et $\sum_{i \in \text{Sol}} c_i$ est maximum.

On définit une fonction $C$ tel que
$C(w, l) = J$ si $S \subseteq \{1, 2, \cdots, l\}$ et $J = \max_S \left(\sum_{i \in S} c_i\right)$ On remarque alors que $C(w, l+1)$ peut être défini à partir de $C(w, l)$ en considérant la solution maximum sur toutes les solutions incluant $l+1$ ou non. Donc, on a

$$\begin{cases} C(w, 0) = 0 \\ C(w, l+1) = \max\{C(w, l), c_{i+1} + C(w - w_{i+1}, l)\} \end{cases}$$

Donc en $O(n.K)$ étapes on a $\{C(1, n), \cdots, C(K, n)\}$.

# Le problème du sac à dos

## Définition.

**Input/donnée:** $n$ pairs $(c_i, w_i)$ d'entiers et un entier $K$. On peut interpréter $c_i$ comme étant le coût d'un objet numéro $i$ et $w_i$ son volume/poids. $K$ est la capacité totale du sac à dos.
**Le problème d'optimisation:** trouver $\text{Sol} \subseteq \{1, 2, \cdots, n\}$ tel que $\sum_{i \in \text{Sol}} w_i \leq K$ et $\sum_{i \in \text{Sol}} c_i$ est maximum.

On définit une fonction $C$ tel que
$C(w, I) = J$ si $S \subseteq \{1, 2, \cdots, I\}$ et $J = \max_S \left(\sum_{i \in S} c_i\right)$ On remarque alors que $C(w, I+1)$ peut être défini à partir de $C(w, I)$ en considérant la solution maximum sur toutes les solutions incluant $I+1$ ou non. Donc, on a

$$\begin{cases} C(w, 0) = 0 \\ C(w, I+1) = \max\{C(w, I), c_{i+1} + C(w - w_{i+1}, I)\} \end{cases}$$

Donc en $O(n.K)$ étapes on a $\{C(1, n), \cdots, C(K, n)\}$.

## Remarque (ce n'est pas polynomial!!!!!!).

La taille de l'entrée est

$$\sum_{i=1}^{n} \log c_i + \sum_{i=1}^{n} \log w_i + \log K\,.$$

# Le problème du sac à dos (suite)

On définit une fonction $W$ tel que $W(c, l) = J$ où $J$ est le plus petit volume $\sum_{i \in S} w_i$ d'un sous-ensemble $S \subseteq \{1, 2, \cdots, l\}$ tel que $\sum_{i \in S} w_i = c$
On peut calculer $W(c, l)$ pour tout $c \leq n. \max\{c_1, c_2, \cdots, c_n\}$ et $l \leq n$ car

$$\begin{cases} W(c, 1) = w_1 \text{ si } c = c_1 \\ W(0, l) = 0 \\ W(c, l+1) = \min\{W(c, l), w_{i+1} + W(c - c_{i+1}, l)\} \end{cases}$$

Ces équations réflètent le fait que $W(c, l+1)$ est le minimum des volumes des ensembles qui incluent ou non l'objet $l+1$. En choisissant le plus grand $c$ tel que $W(c, n) \leq K$ on a un algorithme en $O(n^2. \max\{c_i\})$.

# Le problème du sac à dos (suite)

On définit une fonction $W$ tel que $W(c, l) = J$ où $J$ est le plus petit volume $\sum_{i \in S} w_i$ d'un sous-ensemble $S \subseteq \{1, 2, \cdots, l\}$ tel que $\sum_{i \in S} w_i = c$
On peut calculer $W(c, l)$ pour tout $c \leq n.\max\{c_1, c_2, \cdots, c_n\}$ et $l \leq n$ car

$$\begin{cases} W(c, 1) = w_1 \text{ si } c = c_1 \\ W(0, l) = 0 \\ W(c, l+1) = \min\{W(c, l), w_{i+1} + W(c - c_{i+1}, l)\} \end{cases}$$

Ces équations réflètent le fait que $W(c, l+1)$ est le minimum des volumes des ensembles qui incluent ou non l'objet $l+1$. En choisissant le plus grand $c$ tel que $W(c, n) \leq K$ on a un algorithme en $O(n^2.\max\{c_i\})$.

**Remarque (ce n'est toujours pas polynomial!!!!!!).**

La taille de l'entrée est

$$\sum_{i=1}^{n} \log c_i + \sum_{i=1}^{n} \log w_i + \log K\,.$$

# Le problème du sac à dos (suite)

## Décomposer les $c_i$

On code les $c_i$ **en binaire** et on considère les **bits les plus significatifs**. On fixe un entier $b$ et on "élimine" les $b$ bits les moins significatifs des $c_i$ en définissant $c'_i = 2^b.d_i$ avec $d_i = \lfloor \frac{c_i}{2^b} \rfloor$. Si les données en entrée sont $w_1, w_2, \cdots, w_n, K$ **et** $d_1, \cdots, d_n$ alors la solution trouvée par l'algorithme est **la même** que si les données sont $w_1, w_2, \cdots, w_n, K$ **et** $c'_1, \cdots, c'_n$. La complexité est alors $O(n^2.\frac{\max c_i}{2^b})$ et on trouve une solution $S'$:

$$\sum_{i\in S} c_i \underbrace{\geq}_{\text{S est optimale}} \sum_{i\in S'} c_i \geq \sum_{i\in S'} c'_i \underbrace{\geq}_{S' \text{ est optimale}} \sum_{i\in S} c'_i \geq \sum_{i\in S} c_i - n.2^b .$$

(la dernière inégalité utilise le fait que $c'_i \geq c_i - 2^b$.) On a alors (via 2nde et 4ième expressions):

$$\sum_{i\in S} c_i - \sum_{i\in S'} c_i \leq n.2^b$$

et donc

$$\frac{\sum_{i\in S} c_i - \sum_{i\in S'} c_i}{\sum_{i\in S} c_i} \leq \frac{\sum_{i\in S} c_i - \sum_{i\in S'} c_i}{\max c_i} \leq \frac{n.2^b}{\max c_i}$$

On a donc un algorithme approché pour $\varepsilon = \frac{n.2^b}{\max c_i}$. On choisit $b = \log\left(\varepsilon \frac{\max c_i}{n}\right)$ pour avoir une complexité en $O(n^3/\varepsilon) \longrightarrow$ un **FPTAS**.

## Définition: avec très grande probabilité

On dit qu'un évènement $\mathcal{E}_n$ arrive avec **une très grande probabilité** si la probabilité de cet évènement est plus grande que $1 - O\left(\frac{1}{n^c}\right)$ pour une constante $c > 0$.

# Approximations randomisées et classes PRAS, EPRAS et FPRAS.

## Définition: avec très grande probabilité

On dit qu'un évènement $\mathcal{E}_n$ arrive avec **une très grande probabilité** si la probabilité de cet évènement est plus grande que $1 - O\left(\frac{1}{n^c}\right)$ pour une constante $c > 0$.

## Définitions: PRAS, EPRAS, FPRAS

Un algorithme randomisé fournit un **schéma randomisé d'approximation polynomiale** si **pour toute instance** d'un problème d'optimisation et pour tout $\varepsilon$ en un temps polynomial en $n$ l'algorithme fournit une solution qui est **à un facteur** $\varepsilon$ de l'optimal avec une **très grande probabilité** . La classe de ces problèmes est notée **PRAS** (pour " Polynomial-time Randomized Approximation Scheme").

De plus, un schéma d'approximation randomisé est dit efficace si la complexité est POLYNOMIALE EN LA TAILLE DE L'ENTRÉE **sans dépendre d'** $\varepsilon$. La classe de ces problèmes est appelée **EPRAS**.

Enfin, le schéma d'approximation randomisé est totalement polynomial si la complexité est POLYNOMIAL À LA FOIS EN LA TAILLE DE L'ENTRÉE ET DE $\varepsilon$ . La classe correspondante est **FPRAS**.

# Le problème MAX-3-SAT

C'est le problème d'optimisation correspondant au **problème de décision 3-SAT**.

## Définition de MAX-3-SAT

**Input/donnée:** Une formule 3-SAT construite sur $n$ variables et $m$ clauses

$$\begin{cases} C_1 \wedge C_2 \wedge \cdots \wedge C_m \\ \text{avec } C_i \text{ de la forme } C_i = x_r \vee x_s \vee x_t \\ \text{où } (x_r, x_s, x_t) \in \{x_1, x_2, \cdots, x_n, \bar{x}_1, \bar{x}_2, \cdots, \bar{x}_n\}^3 \end{cases}$$

**Output/Sortie:** trouver une affectation des booléens $x_i$ qui maximise le nombre de clauses satisfaites.

# Le problème MAX-3-SAT

C'est le problème d'optimisation correspondant au **problème de décision 3-SAT**.

## Définition de MAX-3-SAT

**Input/donnée:** Une formule 3-SAT construite sur $n$ variables et $m$ clauses

$$\begin{cases} C_1 \wedge C_2 \wedge \cdots \wedge C_m \\ \text{avec } C_i \text{ de la forme } C_i = x_r \vee x_s \vee x_t \\ \text{où } (x_r, x_s, x_t) \in \{x_1, x_2, \cdots, x_n, \bar{x}_1, \bar{x}_2, \cdots, \bar{x}_n\}^3 \end{cases}$$

**Output/Sortie:** trouver une affectation des booléens $x_i$ qui maximise le nombre de clauses satisfaites.

## MAX-3-SAT est NP-difficile

Si MAX-3-SAT peut être résolu en **temps polynomial** alors ce sera le cas aussi de 3-SAT!

# Le premier glouton venu

## GreedyApproxMax3SAT

```
for i = 1 to n do
    x_i = 0;
    Faire Pile ou Face;
    if Pile then
        x_i = 1;
    end
end
```

# Le premier glouton venu

## GreedyApproxMax3SAT

**for** $i = 1$ *to* $n$ **do**
    $x_i = 0$;
    Faire Pile ou Face;
    **if** *Pile* **then**
        $x_i = 1$;
    **end**
**end**

## Observations.

- L'algorithme **`GreedyApproxMax3SAT`** tourne en temps linéaire $O(n)$.
- L'**espérance** du nombre de clauses satisfaites est $\frac{7}{8}m$.
- La **`méthode probabiliste`** nous dicte donc que pour toute instance de 3-SAT, il existe une affectation qui va satisfaire **au moins** $\frac{7}{8}$ **des clauses**.
- Une instance de 3-SAT avec moins de 7 clauses peut toujours être satisfaite.

# L'amplification de Johnson (1974)

## Un autre glouton

L'algorithme de Johnson consiste à **répéter** gloutonnement `GreedyApproxMax3SAT` **jusqu'à** ce qu'`au moins` $\frac{7}{8}$ `des clauses soient satisfaites`.

# L'amplification de Johnson (1974)

## Un autre glouton

L'algorithme de Johnson consiste à **répéter** gloutonnement `GreedyApproxMax3SAT` **jusqu'à** ce qu'`au moins` $\frac{7}{8}$ `des clauses soient satisfaites`.

## Théorème.

L'algorithme de Johnson tourne en **moyenne** en `temps polynomial` et fournit une solution approchée avec un facteur $8/7$.

# L'amplification de Johnson (1974)

## Un autre glouton

L'algorithme de Johnson consiste à **répéter** gloutonnement `GreedyApproxMax3SAT` **jusqu'à** ce qu'`au moins` $\frac{7}{8}$ `des clauses soient satisfaites`.

## Théorème.

L'algorithme de Johnson tourne en **moyenne** en `temps polynomial` et fournit une solution approchée avec un facteur $8/7$.

## Les observations clefs

- On a vu qu'on peut toujours satisfaire au moins $7/8$ des clauses.
- En répétant à l'infini `GreedyApproxMax3SAT`, on va finir par trouver l'optimum.
- Combien de répétitions pour trouver une affectation satisfaisant au moins $7/8$ des clauses?
- L'idée est donc de quantifier

$$\mathbb{P}\,[\text{un appel à } \texttt{GreedyApproxMax3SAT} \text{ satisfait au moins } 7/8 \text{ des clauses}]$$

# Un appel à GreedyApproxMax3SAT

## Lemme

$\mathbb{P}$ [un appel à `GreedyApproxMax3SAT` satisfait au moins $7/8$ des clauses] $\geq \frac{1}{8m}$ .

# Un appel à GreedyApproxMax3SAT

## Lemme

$\mathbb{P}$ [un appel à **`GreedyApproxMax3SAT`** satisfait au moins $7/8$ des clauses] $\geq \frac{1}{8m}$ .

**Preuve.** Soit $p_j$ la probabilité qu'**`exactement` $j$ `clauses`** soient satisfaites par un appel à **`GreedyApproxMax3SAT`** et $p$ la probabilité qu'**`au moins` $7/8$ `des clauses`** soient satisfaies. Si $Z$ est la v.a. des clauses satisfaites par une affectation gloutonne, on a

$$\frac{7}{8}m = \mathbb{E}[Z] = \sum_{j=0}^{m} jp_j$$

et donc

$$\frac{7}{8}m = \sum_{j=0}^{7m/8-1} jp_j + \sum_{7m/8}^{m} jp_j \leq \left(\frac{7m-1}{8}\right) \sum_{j=0}^{7m/8-1} p_j + m \sum_{7m/8}^{m} p_j \leq \left(\frac{7m-1}{8}\right).1 + m.p$$

On résout pour trouver $p \geq \frac{1}{8m}$.

# Un appel à GreedyApproxMax3SAT

## Lemme

$\mathbb{P}$ [un appel à **`GreedyApproxMax3SAT`** satisfait au moins $7/8$ des clauses] $\geq \frac{1}{8m}$ .

**Preuve.** Soit $p_j$ la probabilité qu'**`exactement` $j$ `clauses`** soient satisfaites par un appel à **`GreedyApproxMax3SAT`** et $p$ la probabilité qu'**`au moins` $7/8$ `des clauses`** soient satisfaies. Si $Z$ est la v.a. des clauses satisfaites par une affectation gloutonne, on a

$$\frac{7}{8}m = \mathbb{E}[Z] = \sum_{j=0}^{m} j p_j$$

et donc

$$\frac{7}{8}m = \sum_{j=0}^{7m/8-1} j p_j + \sum_{7m/8}^{m} j p_j \leq \left(\frac{7m-1}{8}\right) \sum_{j=0}^{7m/8-1} p_j + m \sum_{7m/8}^{m} p_j \leq \left(\frac{7m-1}{8}\right).1 + m.p$$

On résout pour trouver $p \geq \frac{1}{8m}$.

**En conséquence**, en répétant en moyenne au plus $8m$ fois on trouve une affectation satisfaisant au moins $7/8$ des clauses.

# Un appel à GreedyApproxMax3SAT

## Lemme

$\mathbb{P}$ [un appel à **GreedyApproxMax3SAT** satisfait au moins $7/8$ des clauses] $\geq \frac{1}{8m}$ .

**Preuve.** Soit $p_j$ la probabilité qu'**exactement $j$ clauses** soient satisfaites par un appel à **GreedyApproxMax3SAT** et $p$ la probabilité qu'**au moins $7/8$ des clauses** soient satisfaies. Si $Z$ est la v.a. des clauses satisfaites par une affectation gloutonne, on a

$$\frac{7}{8}m = \mathbb{E}[Z] = \sum_{j=0}^{m} j p_j$$

et donc

$$\frac{7}{8}m = \sum_{j=0}^{7m/8-1} j p_j + \sum_{7m/8}^{m} j p_j \leq \left(\frac{7m-1}{8}\right) \sum_{j=0}^{7m/8-1} p_j + m \sum_{7m/8}^{m} p_j \leq \left(\frac{7m-1}{8}\right).1 + m.p$$

On résout pour trouver $p \geq \frac{1}{8m}$.

**En conséquence**, en répétant en moyenne au plus $8m$ fois on trouve une affectation satisfaisant au moins $7/8$ des clauses.

## Exercice:

combien de répétitions pour avoir avec une très grande probabilité une affectation satisfaisant au moins $7/8$ des clauses?