Approx

Problème d'optimisation (déf haut-niveau)

Entrée: ensemble d'objets  $O$   
contraintes  $C$  sur la faisabilité d'une solution  
critère d'optimisation  $\alpha$

Sortie: solution faisable selon  $C$  qui maximise  
(ou minimise)  $\alpha$

ex: MAX-CLIQUE

Entrée: graphe  $G$

Sortie: clique  $X \subseteq V(G)$  de taille max

objets  $O = G$   
faisable =  $\{X \subseteq V(G); X \text{ est une clique}\}$   
critère:  $|X|$  à maximiser

ex: MIN-VERTEX-COVER

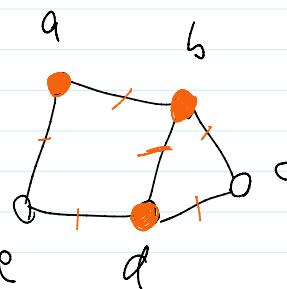
Entrée: graphe  $G$

Sortie: un ensemble  $X \subseteq V(G)$  tel que

$\forall uv \in E(G), \text{ on a } u \in X \text{ ou } v \in X$

de taille minimum.

On appelle  $X$  une couverture.



$\{a, b, d\}$  est une couverture

objet = G

faisable =  $\{X \subseteq V(G) : X \text{ est une couverture}\}$

critère =  $|X|$  à minimiser

---

Soit A un problème de minimisation.

Un algorithme M est une c-approximation pour A si

- M prend un temps polynomial
- $\forall I$  instance I avec valeur optimale  $OPT(I)$ ,

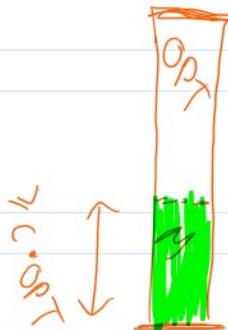
M retourne une sol. faisable de valeur au plus  $c \cdot OPT(I)$   
(avec  $c \geq 1$ )



Soit B un problème de maximisation

Un algo M est une c-approx si:

- M prend un temps polynomial
- $\forall I$ , M retourne une sol faisable de valeur au moins  $c \cdot OPT(I)$  ( $c \leq 1$ )



ex: MAX-SAT

$$\boxed{x_1 \vee \bar{x}_2 \vee x_3}$$

Entrée: formule CNF  $\varphi$

$$C_1 \wedge C_2 \wedge \dots \wedge C_m$$

Sortie: assignation qui maximise le #  
de clauses satisfaites

Voici une  $\frac{1}{2}$ -approx

maxsat( $\varphi$ )

Soit A l'assignation où  $x_1 = x_2 = \dots = x_n = \text{true}$

si A satisfait  $\geq$  la moitié des clauses

retourner A

sinon

soit  $\bar{A}$  où  $x_1 = x_2 = \dots = x_n = \text{false}$

retourner  $\bar{A}$

C'est une  $\frac{1}{2}$ -approx. car :

soit  $\text{OPT}(\varphi)$  le # max de clauses satisfaisables.

soit m le # de clauses de  $\varphi$ .

→ Il est évident  $\text{OPT}(\varphi) \leq m$ .

Si l'algorithme retourne A, alors A satisfait  $\geq \frac{1}{2}m$  clauses.

→ Dans ce cas,  $\frac{1}{2}m \geq \frac{1}{2}\text{OPT}(\varphi)$  [ $\Rightarrow \frac{1}{2}$ -approx]

Sinon, soit  $C_i$  une clause non-satisfait par A.

Alors  $C_i$  est satisfait par  $\bar{A}$ .

$\Rightarrow \bar{A}$  satisfait  $\geq \frac{1}{2}m$  clauses et  $\frac{1}{2}m \geq \frac{1}{2}\text{OPT}$  [ $\Rightarrow \frac{1}{2}$ -approx].

Q : est-ce qu'on peut faire mieux ?

$\frac{3}{4}$ -approx?  $\frac{19}{20}$ -approx?  $(1-\epsilon)$ -approx?

R :  $\exists \frac{7}{8}$ -approx si chaque clause a 3 variables

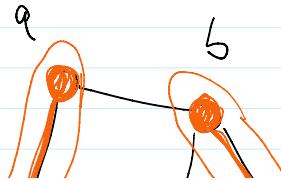
Conjecture : pas mieux que  $\frac{7}{8}$  (avec 3 vars) sauf si P=NP.

MiN-VERTEX-COVER

$\text{minvc}(G)$

$$X = \bigcup_{v \in V} \{v\}$$

|

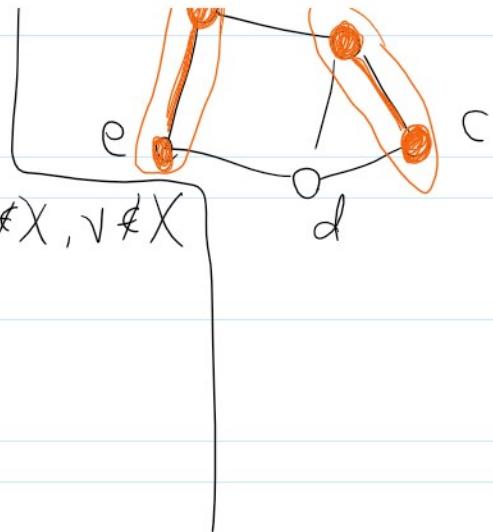


$X = \emptyset$   
done = false  
tant que not done

si  $\exists u \in E(G)$  telle que  $u \notin X, v \notin X$   
 $X \leftarrow X \cup \{u, v\}$

sinon  
done = true

return  $X$



- minvc est une 2-approx
- Soient  $u_1, v_1, u_2, v_2, \dots, u_k, v_k$  la séquence d'arêtes considérées par l'algorithme. On note qu'aucune de ces arêtes ne partagent de sommet. Toute solution (couverture) doit donc contenir au moins un de  $u_i$  ou  $v_i$ , et au moins un de  $u_j$  ou  $v_j$ , et...  
... un de  $u_k$  ou  $v_k$ .

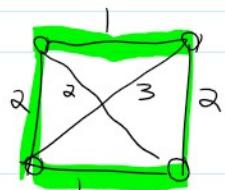
Dans  $\underline{\text{OPT}}(G) \geq k$ . Notre solution a  $2k$  sommets  
Puisque  $2k \leq 2\underline{\text{OPT}}(G)$ , on a une 2-approx.

MinTSP (traveling salesperson problem)

Entrée: graphe  $G$ , coût  $w: V \times V \rightarrow \mathbb{R}^{>0}$

Sortie: cycle passe 1 fois par sommet qui

minimise la somme des coûts des arêtes

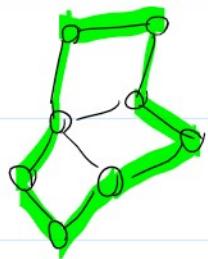


coût = 6

On va montrer que si  $P \neq NP$ , MinTSP n'a pas de c-approx  
 $\forall$  constante  $c \geq 1$

H constante  $c \geq 1$

Langage  $\text{HAMCYCLE} = \{G : \begin{array}{l} \text{il existe un cycle qui} \\ \text{est Hamiltonien} \end{array}\}$  passe 1 fois par sommet}

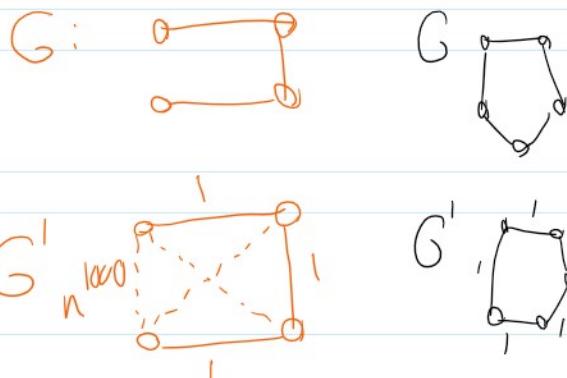


Thm: HAMCYCLE est NP-complet.

Thm: MinTSP n'a pas de  $c$ -approx. (s:  $P \neq NP$ )

Soit  $G$  une instance de HAMCYCLE. On crée  $\langle G', w \rangle$  une instance de MinTSP comme suit:

$$V(G') = V(G) \text{ et} \\ w(\{u, v\}) = \begin{cases} 1 & \text{si } uv \in E(G) \\ n^{1000} & \text{sinon} \end{cases}$$



S:  $G \in \text{HAMCYCLE}$ , alors  $\boxed{\text{OPT}(G') = n}$  car un cycle hamilt. dans  $G$  a un coût de  $n$ .

S:  $G \notin \text{HAMCYCLE}$ , tout cycle de  $G'$  doit emprunter une arête de coût  $n^{1000}$ . Donc  $\boxed{\text{OPT}(G') > n^{1000}}$

Supposons que MinTSP a une  $c$ -approx,  $c \geq 1$ .

Quand on l'exécute sur  $G'$ , on a

- s:  $G \in \text{HAMCYCLE}$ , l'approx retourne une sol.  $\leq c \cdot \text{OPT} = c \cdot n$
- s:  $G \notin \text{HAMCYCLE}$ , l'approx retourne une sol.  $\geq n^{1000}$

On a  $n^{1000} > c \cdot n$  (pour  $n$  assez grand). Donc, on peut utiliser la  $c$ -approx pour distinguer si  $G \in \text{HAMCYCLE}$ , ce qui permet de décider HAMCYCLE. Donc si  $P \neq NP$ , cette  $c$ -approx ne

de décider HAMCYCLE. Donc si  $P \neq NP$ , cette c'approx ne peut pas exister.

Langage "gap": langage avec une promesse sur chaque instance  $I$

- |   |   |
|---|---|
| <ul style="list-style-type: none"> <li>• soit <math>\text{OPT}(I) \leq f(I)</math></li> <li>• ou bien <math>\text{OPT}(I) &gt; g(I)</math></li> </ul> | $\left. \begin{array}{l} \text{minimisation} \\ \text{maximisation} \end{array} \right\}$ |
| <ul style="list-style-type: none"> <li>• soit <math>\text{OPT}(I) \geq f(I)</math></li> <li>• soit <math>\text{OPT}(I) &lt; g(I)</math></li> </ul>    |   |

ex: les instances  $\langle G, w \rangle$  de MinTSP forment le langage gap

dans lequel soit  $\text{OPT}(\langle G, w \rangle) \leq n$

ou  $\text{OPT}(\langle G, w \rangle) > n^{1000}$

Intuition: si un problème d'optimisation a une version gap qui permet de résoudre un pb NP-complet, alors le problème n'a pas d'approx.

Un langage gap A (de maximisation) est difficile s'il existe un problème NP-complet L, et une fonction (langage)

$f: \sum^* \rightarrow A$  telle que (1)  $w \in L \Rightarrow \text{OPT}(f(w)) \geq \alpha(|I|)$   
(2)  $w \notin L \Rightarrow \text{OPT}(f(w)) \leq \beta(|I|)$

Le "gap" généré par f est  $\frac{\beta(|I|)}{\alpha(|I|)}$

- - - - -  $\alpha(111)$

$$\text{ex: MinTSP: gap} = \frac{n^{1000}}{n} = n^{999}$$

Idée: le gap détermine le meilleur facteur d'approx possible.  
Donc MinTSP n'a pas de  $(\underline{n^{999}})$ -approx.

---

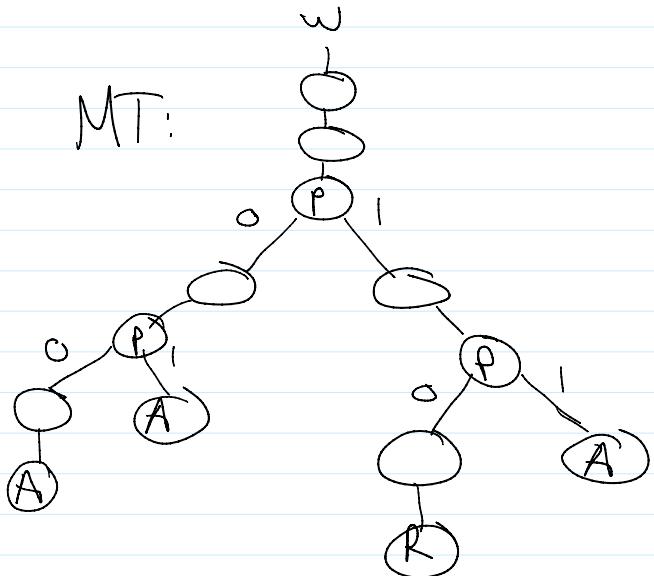
PCP: Probabilistically Checkable Proof

Vérificateur probabiliste en temps  $O(n^k), k \in \mathbb{N}$

MT probabiliste: peut lancer une pièce p t.q.

$$\Pr[p=0] = \Pr[p=1] = \frac{1}{2}$$

Fonctionnement: accès à un Oracle qui, en temps  $O(1)$ ,  
répond à une requête  $\text{rand}(i)$ , en retournant  
○ ou | sur l'emplacement courant.



Temps d'une MT probabiliste  
= # de configs du plus long  
chemin possible

$$\Pr[M \text{ accepte } w] = \frac{\# \text{ chemins acceptant}}{\# \text{ chemins au total}}$$

ex:  $\frac{3}{4}$

Vérificateur  $V$  probabiliste pour un langage  $L$ :

$$w \in L \Rightarrow \exists \text{ certificat } c \text{ t.q. } \Pr[V \text{ accepte } \langle w, c \rangle] = 1$$

$$w \notin L \Rightarrow \forall c \quad \Pr[V \text{ accepte } \langle w, c \rangle] \leq \frac{1}{2}$$

$c$  est "donné" d'avance, les choix aléatoires sont faits après