Complexité algorithmique

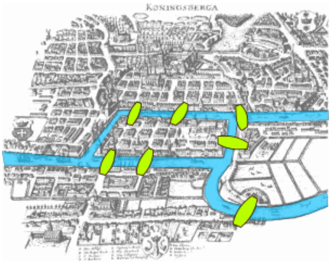
Timothée Pecatte

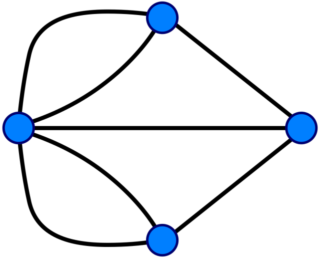
DIU Bloc 5
25/06/2020

Table des matières

Introduction

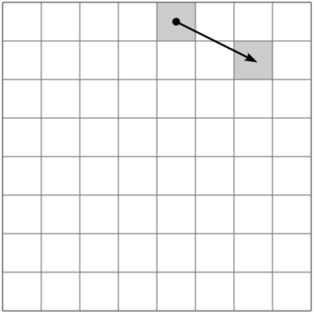
Introduction : chemin eulérien

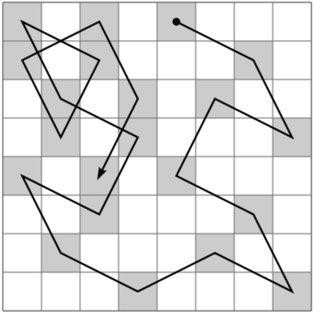




Existe-il un cycle passant par chaque arête exactement une fois ?

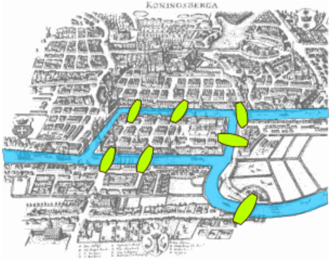
Introduction : chemin hamiltonien



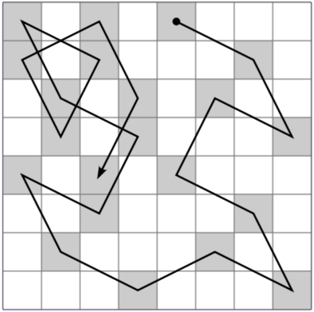


Existe-il un cycle passant par chaque sommet exactement une fois ?

Complexité relative



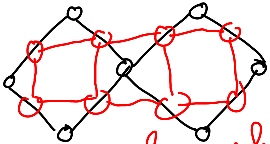
$\leq$



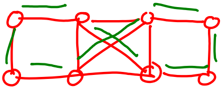
Objectif : classer les problèmes en fonction de leur complexité.

Rappels et premières définitions

G a un cycle eulérien $\iff$ G' a un cycle hamiltonien



en rouge le graphe dual obtenu en remplaçant une arête par un sommet et en reliant 2 sommets dans G' si les 2 arêtes dans G sont incidentes à un même sommet



— cycle hamiltonien qu'on convertit en cycle eulérien dans G

Pourquoi Hamiltonien plus dur que eulérien si on a une équivalence ? Parce qu'on construit le graphe hamiltonien

G a un cycle eulérien $\iff M(G')$ oui

G n'a pas de cycle eulérien $\iff M(G')$ non

On a même une bijection entre les cycles eulériens de G et les cycles hamiltoniens de G'

Hamiltonien est plus dur
car si je sais résoudre hamiltonien
alors je sais résoudre eulérien.

Rappels de complexité algorithmique

Definition

Un **problème** = une **entrée** et sa **sortie** correspondante.

Exemple (Problème du cycle Eulérien)

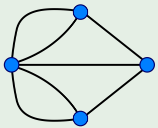
Entrée : un graphe G.

Sortie : un cycle Eulérien C de G s'il existe, "Impossible" sinon.

Definition

Une **instance** d'un problème A = une entrée spécifique.

Exemple (Instance du problème du cycle Eulérien)

Entrée : 

instance positive : il y a une solution.
instance négative : il n'y en a pas

Problèmes de décision

Definition

Un **problème de décision** = sortie **booléenne**.

Exemple (Problème de décision du cycle Hamiltonien)

Étant donné un graphe G, existe-t-il un cycle Hamiltonien dans G ?

Exemple

Étant donné un entier n, est-il premier ?

Exemple

Étant donné un entier n sous forme de produit de nombres premiers, est-il premier ?

- Dans beaucoup de cas il est plus facile de déterminer si une solution existe que de la construire.
- Pour les deux derniers exemples, la spécification de l'entrée (son codage) est importante.

D'autres exemples

Exemple

Étant donné un graphe G, est-il 3-coloriable ?

Exemple

Étant donné un graphe planaire G, est-il 4-coloriable ?

Exemple

Étant donné un programme C, le programme s'arrête-il toujours ?

Exemple

Étant donné une formule logique (composée de OU,ET,NEG), existe-il une assignation des variables qui rend la formule vraie ?

Temps de calcul

Definition (Résolution)

A résout un problème de décision P : $\forall$ entrée I (instance) valide pour P, $A(I) =$ VRAI si et seulement si $I \in P$.

I instance positive de P

Definition (Complexité)

$t_A(I) =$ temps de calcul de A sur I (nombre d'instructions élémentaires).

complexité pire cas

$$T_A(n) = \max\{t_A(I) \mid \text{taille}(I) = n\}$$

$$T_P(n) = \min\{t_A(n) \mid A \text{ résout } P\}$$

Remarque

D'autres mesures de complexité possibles : mémoire, temps de calcul sur architecture parallèle, ...

En pratique

- Rarement accès à $T_P(n)$: utilisation de O
- Etude asymptotique : pas toujours utile sur des données réelles
- Modèles de calculs réels complexes à modéliser

- Problème Eulérien et Hamiltonien : vérification facile $O(n)$
- Énumération de tous les sous-ensembles d'arêtes : $O(n2^n)$
- Caractérisation des graphes Eulériens $\Rightarrow O(n)$
- Cycle Hamiltonien : ? ? ?

en nombre d'arêtes

on s'intéresse de pb de décision, pas de construction

Quelle granularité utiliser pour séparer les problèmes ?

Classes de complexité

Réduction polynomiale

Definition (Réduction - avec les mains)

Le problème P est plus facile que le problème Q si l'on peut se servir d'un algorithme pour le problème Q afin de résoudre le problème P.

Remarque

Si un algorithme A consiste en $O(f(n))$ appels à un algorithme B avec des entrées de taille $O(g(n))$, et que la complexité de B est $O(h(n))$ opérations élémentaires, alors la complexité de A est $O(f(n) \times (g \circ h)(n))$.

Algorithme linéaire ($O(n)$) + stabilité par réduction "raisonnable"
$\Rightarrow$ tous les algorithmes de complexité polynomiale.

Si un algo en $O(n)$ fait appel à chaque itération à un algo en $O(n)$ on a du $O(n^2)$

boîte noire

↳ On recherche des classes de complexité stables par réduc-tion

A $\longrightarrow$ B

entrée de taille n → entrée de taille $h(n)$

Classe polynomiale P : les problèmes "faciles"

Definition

Un problème A appartient à la classe P s'il existe un algorithme qui résout A en temps polynomial.

Example

- Eulérien
- 2-couleur
- Accessibilité : étant donnés un graphe G et deux sommets $s, t \in G$, existe-t-il un chemin de s à t ?
- Connexité : est-ce qu'un graphe donné est connexe ?
- PGCD
- Primalité
- Recherche de motif dans un texte

La classe P

Facile ou difficile ?

Problème dans P ou pas dans P ?

- Si le problème $\in$ P :
  - fournir un algorithme
  - montrer qu'il est correct
  - montrer qu'il est polynomial en la taille des données

voir le problème de programmation dynamique qui est exponentiel en $\log_2(W)$ la taille du sac

- Si le problème $\notin$ P :
  - montrer qu'**aucun** algorithme polynomial n'existe !
  - en général très compliqué

Example

Hamiltonien $\in$ P ? ?

Au-delà de P ?

- Hamiltonien $\in$ P ? ?
- Algorithme en $O(n2^n)$, Hamiltonien $\in$ EXP $\rightarrow$ pas suffisant ?
- Supposons qu'une opération prend $1\mu s = 10^{-6}$s :

n/f(n)	n	n^2	n^3	2^n	3^n	$n!$
10	$10\mu s$	$0.1ms$	$1ms$	$1ms$	$59ms$	$3.63s$
20	$20\mu s$	$0.4ms$	$8ms$	$1s$	58 min	77094 ans
40	$40\mu s$	$1.6ms$	$64ms$	12.73 j	385253 ans	$2.58 \cdot 10^{34}$ ans
60	$60\mu s$	$3.6ms$	$216ms$	36533 ans	$1.34 \cdot 10^{15}$ ans	$2.63 \cdot 10^{68}$ ans

- Et si on "boostait" ma machine ?
- Et si on utilisait un serveur de calcul ?
- Et si on parallélisait massivement les calculs ?

Exponentielle à éviter

1.000 fois plus puissant : 10^9 opérations/s

n/f(n)	n	n^2	n^3	2^n	3^n	$n!$
10	$10ns$	$0.1\mu s$	$1\mu s$	$1\mu s$	$59\mu s$	$3.63ms$
20	$20ns$	$0.4\mu s$	$8\mu s$	$1ms$	3.48 s	$77,1$ ans
40	$40ns$	$1.6\mu s$	$64\mu s$	18.34 h	$385,25$ ans	$2.58 \cdot 10^{31}$ ans
60	$60ns$	$3.6\mu s$	$216\mu s$	$36,5$ ans	$1.34 \cdot 10^{12}$ ans	$2.63 \cdot 10^{65}$ ans

1.000.000 fois plus puissant : 10^{12} opérations/s

n/f(n)	n	n^2	n^3	2^n	3^n	$n!$
10	$10ps$	$0.1ns$	$1ns$	$1\mu s$	$59\mu s$	$3.63ms$
20	$20ps$	$0.4ns$	$8ns$	$1ms$	3.48 ms	28.16 j
40	$40ps$	$1.6ns$	$64ns$	1.1 s	140.71 j	$2.58 \cdot 10^{28}$ ans
60	$60ps$	$3.6\mu s$	$216ns$	13.34 j	$1.34 \cdot 10^{9}$ ans	$2.63 \cdot 10^{62}$ ans

P or not P ?

Quand on ne sait pas...

- Pour beaucoup de problème, on ne sait pas :
  - aucun algo polynomial connu $\Rightarrow$ tous sont exponentiels...
  - ...mais aucune preuve que le problème $\notin$ P !
- Idée : inventer une classe intermédiaire : P $\subseteq$ NP $\subseteq$ EXP

ATTENTION

NP veut dire Nondeterministic Polynomial

NP **ne veut pas dire NON POLYNOMIAL !**

Classe NP : définition avec certificats

Definition

$P \in$ NP si pour chaque **instance positive** I (réponse OUI), il existe un certificat $C(I)$ (de sa positivité) vérifiant :

1. taille de $C(I)$: **polynomiale** en la taille des données du problème
2. vérification à partir de $C(I)$: en temps **polynomial**

Example

Hamiltonien $\in$ NP , Eulérien $\in$ NP.

- NP : solution **facile à vérifier**
- P : solution **facile à trouver**
- On a bien P $\subseteq$ NP

NP– Intuitivement

Coloration de graphe

- j'ai un graphe G et un entier k (une instance I)
- je me demande si G peut être proprement colorié en $\leq k$ couleurs
- quelqu'un observe par-dessus mon épaule, réfléchit et répond "oui" (instance positive)
- je doute : je lui demande une "preuve" (certificat $C(I)$)
- je vérifie, sur la base de sa "preuve", qu'il dit vrai
- si la taille de $C(I)$ et l'algorithme de vérification sont polynomiaux (en la taille des données), le problème est dans NP

NP : définition non-déterministe

Definition

Un problème A appartient à la classe NP s'il existe un algorithme **non-déterministe** qui résout A en temps polynomial.

↳ un algorithme qui a des choix à faire qu'on ne précise pas

↳ Il existe au moins un choix qui résout le pb.

Algorithme 1 Hamiltonien(V, E)

1: choisir un sommet $s \in V$
2: $chemin \leftarrow \emptyset$
3: **tant que** $V \neq \emptyset$ **faire**
4: **si** s n'a pas de voisin **alors retourner** IMPOSSIBLE
5: choisir t un voisin de s } on ne dit pas comment choisir le voisin
6: $V \leftarrow V \setminus s$
7: $chemin \leftarrow (s, t) :: chemin$
8: $s \leftarrow t$
9: **fin tant que**

NP, et alors ?

Proposition

$P \subseteq \mathrm{NP}$

- NP ne nous permet pas de distinguer entre des problèmes qu'on sait être dans P et des problèmes qui ont l'air plus durs.
- Exemple : Eulérien $\in$ NP et Hamiltonien $\in$ NP.
- **Idée :** se restreindre aux problèmes de NP les "plus durs"
- Pour rendre "plus durs" plus précis, on va maintenant formaliser la notion de **réduction**

→ On voudrait distinguer les pbs les + durs qui sont dans NP car $P \subseteq NP$

Réduction polynomiale "many-one"

Definition

$Q \leq^p_m P$ (Q se réduit à P) si et seulement si $\exists f$, calculable en temps polynomial, telle que : $\forall I$ entrée valide de Q,
$I \in Q \Leftrightarrow f(I) \in P$.

$Q \leq^p_m P$

$p \to$ polynomial

$m \to$ many one plusieurs entrées pour Q peuvent avoir le m̂ codage

Exemple

Eulérien $\leq^p_m$ Hamiltonien : réduction illustrée précédemment.

Remarque

Attention : si Q se réduit à P, c'est que Q est plus simple que P (d'où la notation $Q \leq^p_m P$)

Proposition

La classe P *est close par réduction polynomiale "many-one".*

La classe P est bien stable
par réduction polynomiale
Si on a un algo... poly-
-nomial et qu'on l'applique
à une réduction polynomiale
alors on a encore un algo
polynomial

Problèmes équivalents

Example

MAX-clique : étant donné un graphe G, trouver une **clique** maximale. *(une clique est un ensemble de sommets tous reliés deux-à-deux)*

Example

MAX-indépendant : étant donné un graphe G, trouver un **ensemble indépendant** maximal. *(un ensemble indépendant est un ensemble de sommets qui ne sont reliés par aucune arête)*

Proposition

MAX-clique $\leq_m^p$ *MAX-indépendant*
MAX-indépendant $\leq_m^p$ *MAX-clique.*

Classe NP-complet

Definition

Un problème est NP-complet si :

1. il est dans NP
2. chaque problème de NP peut se réduire vers lui

il est + dur que tt pb de NP

Proposition

Pour montrer qu'un problème P est NP-complet, il "suffit" de montrer :

1. $P \in \mathrm{NP}$
2. *il existe* **un** *problème NP-complet Q tel que $Q \leq_m^p P$*

- **une seule** réduction suffit
- mais il faut réduire depuis un problème NP-complet...

L'œuf et la poule

Montrer qu'un problème est NP-complet implique de réduire un problème NP-complet vers lui...mais il faut bien commencer quelque part !

Remarque

Il n'est a priori pas évident qu'il existe au moins un problème NP-complet

Theorem (Cook 1971)

Le problème **SAT** *est* NP*-complet*

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_4 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4)$$

SAT est NP-complet : preuve avec les mains

SAT $\in$ NP : étant donné les valeurs, il suffit d'évaluer la formule pour voir si celle ci est satisfaite.

Plus dur que tous les problèmes de NP :

- $P \in$ NP : soit A un algorithme non-déterministe qui le résout.
- Choix de A réalisés en "lançant une pièce" $\rightarrow$ variables booléenes $c_1, c_2, \ldots c_p$
- Soit I une instance de P.
- Formule $\varphi_A(0, 1, 1, \ldots)$ qui est satisfiable si et seulement si l'algorithme A répond OUI sur l'entrée I avec les résultat de lancés $0, 1, 1, \ldots$.
  Le modèle de calcul des machines de Turing permet d'écrire une telle formule.
- $f(I) = \varphi_A(c_1, \ldots, c_p)$ satisfiable si et seulement I est positive

Et après : c'est les soldes !

Richard Karp, 1972, *Reducibility Among Combinatorial Problems*

- CLIQUE : le problème de la clique (voir aussi le problème de l'ensemble indépendant)
    - SET PACKING : Set packing (empaquetage d'ensemble)
    - VERTEX COVER : le problème de couverture par sommets
        - SET COVERING : le problème de couverture par ensembles
        - FEEDBACK ARC SET : feedback arc set
        - FEEDBACK NODE SET : feedback vertex set
        - DIRECTED HAMILTONIAN CIRCUIT : voir graphe hamiltonien
        - UNDIRECTED HAMILTONIAN CIRCUIT : voir graphe hamiltonien

graphe hamiltonien NP-complet

- 0-1 INTEGER PROGRAMMING : voir optimisation linéaire en nombres entiers

Quand y'en a plus...

3-SAT : satisfaction avec clause comportant 3 littéraux

- CHROMATIC NUMBER : coloration de graphe
  - CLIQUE COVER : partition en cliques
  - EXACT COVER : couverture exacte
    - MATCHING à 3 dimensions : appariement à 3 dimensions
    - STEINER TREE : voir arbre de Steiner
    - HITTING SET : ensemble intersectant
    - KNAPSACK : problème du sac à dos
    - JOB SEQUENCING : séquençage de tâches
    - PARTITION : problème de partition
    - MAX-CUT : problème de la coupe maximum

"Computers and Intractability : A Guide to the Theory of NP-Completeness" de Garey et Johson, 1979 $\Rightarrow \geq 300$ problèmes NP-Completenes

A quoi ça sert ?

Proposition

Les problèmes NP*-complet sont de complexité "équivalente". En particulier :*

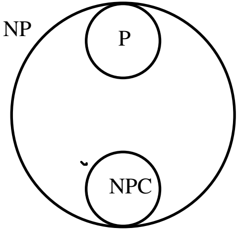
1. *Si* **un seul** *problème* NP*-complet est polynomial* $\Rightarrow$ NP $=$ P *!*
2. *Inversement, si un seul problème* NP*-complet n'est pas polynomial* $\Rightarrow$ *tous les problèmes* NP*-complet aussi !*

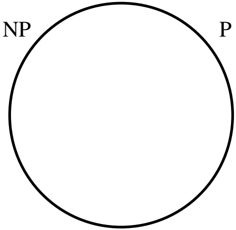
Actuellement :

- Aucun algorithme polynomial n'a été trouvé pour un problème NP-complet
- L'impossibilité de trouver des algorithmes polynomiaux n'a pas été prouvée non plus.

P vs NP

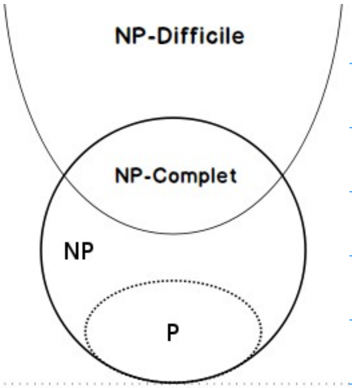
Quel est le bon schéma ?





Conjecture

$P \neq NP$



$P \neq NP$

sous les hypothèses "A <= B et A est NP"
on ne peut pas déduire grand chose :
on peut avoir A est en fait dans P et B aussi
on peut avoir B est EXP

ce qu'on peut dire c'est "A<= B
et A est NP-dur" alors B est NP-dur

si A est NP-complet et A <= B alors B est NP-dur
(cas particulier du précédent)
mais pas forcément B NP-complet par B pourrait
ne pas être NP (dans EXP sans être NP par exemple)

P => je peux trouver facilement la solution
NP => je peux vérifier facilement la solution
NP-dur => plus dur que tous les problèmes de NP
NP + NP-dur = NP-complet
NP complet =>NP dur,   réciproque fausse.
NEXP-dur => NP-dur mais pas NP-complet

Si on a une réduction de A vers B, notée A≤B
alors :   si B est P, A aussi
    si A est NP, B aussi
" A se réduit à B"
 par "résoudre A revient à résoudre B
dans un cas particulier"

La grande question

P=NP ?

- Recherches innombrables sur le sujet depuis des dizaines d'années
- Fait partie des 7 problèmes du millénaire du Clay Mathematics (1 million à la clé)
- Les implications sont multiples et réelles ! Exemple : transactions bancaires cryptées sur le web (codage RSA)
- Gerhard J. Woeginger : liste avec 62 preuves de $\mathrm{P} = \mathrm{NP}$, 50 preuves de $\mathrm{P} \neq \mathrm{NP}$, 2 preuves que le résultat n'est pas prouvable, et une preuve qu'il n'est pas décidable.
- Meilleure borner inférieure pour SAT : $T \cdot S \geq \Omega(n^{2-o(1)})$

Concrètement

Que faire face à un problème inconnu ?

On observe notre problème P

- soit on pense que le problème est **facile** $\Rightarrow$ on cherche un algorithme correct et polynomial (avec le meilleur temps possible !) qui le résout.
- soit on pense que le problèmeest **difficile** $\Rightarrow$ on cherche à montrer qu'il est NP-complet, càd
  - toute solution proposée peut être polynomialement **vérifiable** (appartenance à NP)
  - prendre un problème NP-complet et le **réduire** polynomialement à P

Que faire face à un problème NP-complet ?

Si *Pb* est NP-complet

- Premier constat : ne pas s'acharner à trouver un algorithme exact et rapide qui fonctionne sur toutes les instances
- Baisser ses exigences :
  a. soit sur la **rapidité** d'exécution :
     Je veux la réponse exacte, je suis prêt à attendre (si la taille est petite, ça ira)
  b. soit sur l'**exactitude** de la réponse :
     Je veux une réponse rapide, tant pis si elle n'est pas tout à fait exacte
  c. soit sur l'**ensemble des instances** autorisées :
     Je peux avoir un algorithme rapide et exact si mes données d'entrée sont "gentilles"

→ la coloration est NP complet mais si on se restreint à des arbres ou à des graphes de d° ≤ 2 c'est polynomial.

Que faire face à un problème NPC ?

Retour sur le cas (c)

Je peux avoir un algorithme rapide et exact si mes données d'entrée sont "gentilles"

- NP-complet signifie qu'au moins une instance est "difficile"...
- ...mais pas forcément toutes !
- Pour certaines instances, le problème (pourtant NP-complet) pourrait être résolu en temps polynomial

Example

- **MIN-COL** limité aux graphes de degré maximum 2
- **MIN-COL** limité aux arbres

MIN-COL limité aux graphes de degré maximum 2

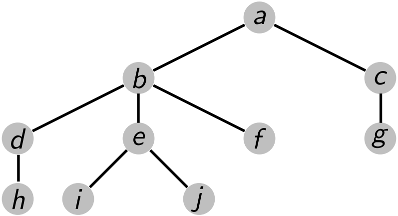
MIN-COL limité aux graphes de degré maximum 2 :
Instance : un graphe G de degré maximum 2
Question : quel est le nombre minimum de couleurs nécessaires pour colorier G de façon propre ?

Exercice

- Si G est connexe et de degré max. 2, à quoi ressemble G ?
- Si G n'est pas (forcément) connexe, à quoi ressemble-t-il ?
- Montrer que le problème **MIN-COL** limité aux graphes de degré max. 2 est dans P

MIN-COL limité aux arbres

MIN-COL :
Instance : un arbre G
Question : quel est le nombre minimum de couleurs nécessaires pour colorier G de façon propre ?



Exercice

Montrer que le problème **MIN-COL** limité aux arbres est dans P

Que faire face à un problème NPC ?

Retour sur le cas (b)

Je veux une réponse rapide, tant pis si elle n'est pas tout à fait exacte

- S'applique surtout aux problèmes d'optimisation
- Temps d'exécution exigé : polynomial
- Une possibilité : algorithmes d'approximation
  - algorithme polynomial
  - garantissant un résultat $\leq r \cdot c_{opt}$ (maximisation) ou $\geq r \cdot c_{opt}$ (minimisation)
  - pour toutes les instances
  - r est appelé le ratio d'approximation

mais le pb du sac à dos non entier

Conclusion

Pour l'examen : questions de cours

- Tout une théorie existe (seulement effleurée ici)
- Permet d'estimer la difficulté des problèmes (P vs NP)
- Si le problème est NP-complet, on adapte sa stratégie de résolution

Slides basés en partie sur ceux du DIU à Nantes.

Retour sur MAX-CLIQUE

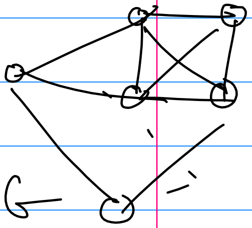
. Une clique est ensemble de sommets sont 2 à 2 reliés /

. Réseau social => plus gde communauté tous connectés 2 à 2

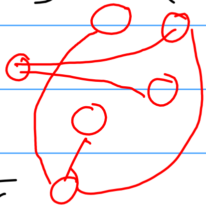
MAX-INDEPENDANT

. Pb complémentaire : plus gd ensemble tel que 2 éléments ne sont pas connectés => allocation de bandes de fréquence

MAX-CLIQUE sont équi-valents



G



$\bar{G}$

on relie deux sommets dans $\bar{G}$ qui ne sont pas reliés dans G

J'ai une arête dans G ssi je n'ai pas une arête dans $\bar{G}$