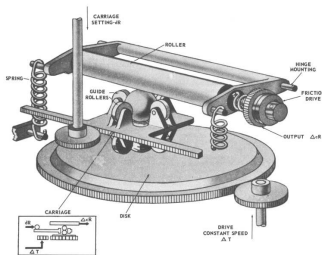


Cours 8: Complexité en temps. NP-complétude.

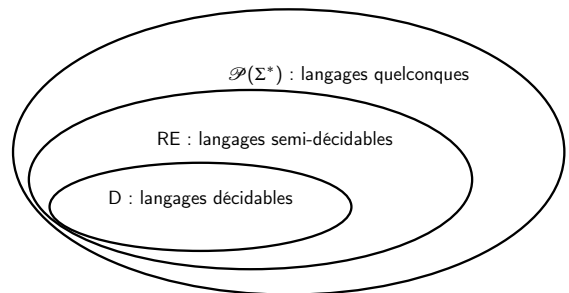


Olivier Bournez
bournez@lix.polytechnique.fr

Ecole Polytechnique
INF412

1

Le paysage de la calculabilité : vers de la complexité

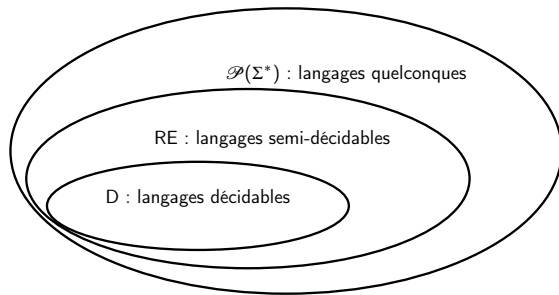


On ne considère dorénavant que les problèmes de D : les langages décidables. On souhaite une classification plus fine, qui prenne en compte les ressources utilisées.

2

La suite de ce cours

- On souhaite parler d'une ressource particulière : le **temps de calcul**.
- Objectif :
 - ▶ distinguer ce qui est raisonnable de ce qui n'est pas raisonnable en termes de temps de calcul.

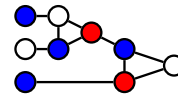


3

Exemple 1 : tester le coloriage d'un graphe

■ BON COLORIAGE :

- Donnée:**
1. Un graphe (fini) $G = (V, E)$ de n sommets et m arêtes.
 2. avec la donnée pour chaque sommet $v \in V$ d'une couleur.
- Réponse:** Décider si cela correspond à un coloriage de G : c'est-à-dire si il n'y a pas d'arêtes de G avec deux extrémités de la même couleur.



Algorithme :

- res := vrai.
- pour chaque arête (u, v)
 - ▶ si $\text{couleur}(u) = \text{couleur}(v)$ alors res := faux.
- retourner res.

■ Complexité de l'algorithme (en nombre d'instructions effectuées^a) :

- ▶ $\mathcal{O}(m)$.
- ▶ $\mathcal{O}(n^2)$ (car $m \leq n(n-1)/2 \leq n^2$).

a. On suppose qu'accéder à une arête se fait en temps constant

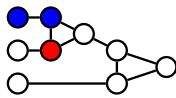
4

Exemple 2 : 3-COLORABILITE

3-COLORABILITE :

Donnée: Un graphe (fini) $G = (V, E)$.

Réponse: Décider s'il existe un coloriage du graphe utilisant au plus 3 couleurs.



Algorithme :

- res :=faux.
- pour les 3^n façons possibles d'affecter une couleur à chaque sommet
 - ▶ tester si cela est un coloriage de G .
 - ▶ si oui alors res :=vrai.
- retourner res.

- Complexité de l'algorithme (en nombre d'instructions effectuées^a) :
 - ▶ $\mathcal{O}(3^n \times n^2) = \mathcal{O}(3^{(1+\epsilon)n})$
 où n est le nombre de sommets.

a. On suppose qu'accéder à une arête se fait en temps constant

5

Quelques considérations

- Pour un processeur capable d'effectuer un million d'instructions élémentaires par seconde.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n=10$	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	4 s
$n=30$	< 1 s	< 1 s	< 1 s	< 1 s	< 1 s	18 m	10^{25} a
$n=50$	< 1 s	< 1 s	< 1 s	< 1 s	11 m	36 a	∞
$n=10^2$	< 1 s	< 1 s	< 1 s	1s	12.9 a	10^{17} a	∞
$n=10^3$	< 1 s	< 1 s	1s	18 m	∞	∞	∞
$n=10^4$	< 1 s	< 1 s	2 m	12 h	∞	∞	∞
$n=10^5$	< 1 s	2 s	3 h	32 a	∞	∞	∞
$n=10^6$	1s	20s	12 j	31710 a	∞	∞	∞

- Notations : ∞ = le temps dépasse 10^{25} années, s= seconde, m= minute, h = heure, a = an.

6

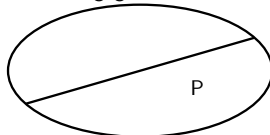
Une convention

- La **convention** suivante s'est imposée en informatique :

- ▶ Temps raisonnable = temps polynomial,
 - c'est-à-dire en $\mathcal{O}(n^k)$ pour un entier k .

- Graphiquement :

D : langages décidables.



P = problèmes avec un algorithme polynomial en temps.

- Exemples :
 - ▶ BON COLORIAGE est dans P.
 - ▶ on ne sait pas si 3-COLORABILITE est dans P.

7

Raison 1 : s'affranchir du modèle de calcul

- Thèse effective de Church :

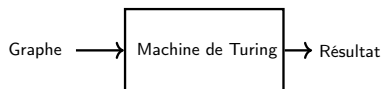
- ▶ Les modèles suivants se simulent deux à deux :
 - Les machines de Turing à un ruban.
 - Les machines de Turing à deux rubans.
 - Les machines à $k \geq 2$ piles
 - Les machines RAM¹
 - Les machines à $k \geq 2$ compteurs
 - Les programmes² JAVA, C, CAML, ...

- ▶ de telle sorte que : t instructions de l'un sont simulées par un nombre d'instructions polynomial en t par l'autre.

1. dont les registres restent bornés.
2. en définissant proprement les instructions autorisées.

8

Raison 1 : s'affranchir du modèle de calcul



Nombre d'instructions : T

- On peut donc parler d'algorithme **raisonnable** sans avoir à préciser dans quel langage / avec quel modèle l'algorithme est implémenté.
- Le temps correspond (à la composition par un polynôme près) au nombre d'instructions dans chacun de ces modèles.
 - ▶ on peut donc par exemple mesurer le temps comme le nombre d'étapes sur les machines de Turing.

9

Raison 2 : s'affranchir de problèmes du codage



Temps : Polynomial/Non polynomial

10

La classe P

- Soit $t : \mathbb{N} \rightarrow \mathbb{N}$ une fonction.
- On note $\text{TIME}(t(n))$ pour la classe des problèmes (langages) décidés par une machine de Turing en temps $\mathcal{O}(t(n))$, où n est la taille de l'entrée.
 - ▶ Si on préfère, $L \in \text{TIME}(t(n))$ s'il y a une machine de Turing M telle que
 - M décide L : pour tout mot w , M accepte w si et seulement si $w \in L$, et M refuse w si et seulement si $w \notin L$;
 - En un temps en $\mathcal{O}(t(n))$, où $n = |w|$ est la longueur de w : il y a des entiers n_0 et c tels que pour tout mot w , M accepte ou refuse en utilisant au plus $c \times t(n)$ étapes, où $n = |w|$, dès que $|w| \geq n_0$.
- La classe P est la classe des problèmes (langages) définie par :

$$P = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k).$$

- ▶ P caractérise les algorithmes raisonnables avec la convention précédente.

11

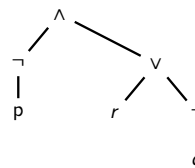
Exemples de problèmes de P

- BON COLORIAGE $\in P$.

- Évaluation en calcul propositionnel $\in P$.

Donnée: Une formule $F(x_1, x_2, \dots, x_n)$ du calcul propositionnel en forme normale conjonctive, des valeurs $x_1, \dots, x_n \in \{0, 1\}$ pour chacune des variables de la formule.

Réponse: Décider si la formule F s'évalue à vrai pour ces valeurs des variables.



$$p = 0, r = 1, q = 0.$$

12

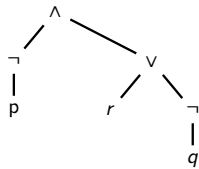
Dans P ?

- 3-COLORABILITE

- SAT :

Donnée: Une formule $F = (x_1, \dots, x_n)$ du calcul propositionnel en forme normale conjonctive.

Réponse: Décider si F est satisfiable : c'est-à-dire décider s'il existe $x_1, \dots, x_n \in \{0, 1\}$ tel que F s'évalue à vrai pour cette valeur de ses variables x_1, \dots, x_n .

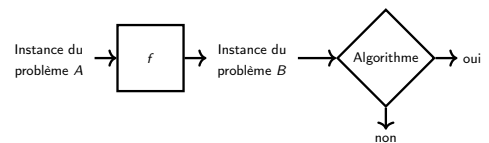


- Pour les deux problèmes, on ne sait pas s'ils sont dans P.
- Lequel de ces problèmes est le plus difficile ?

13

Comment comparer les problèmes

- Idée de \leq_M :



- On va utiliser exactement la même idée, mais en remplaçant **calculable** par **calculable en temps polynomial**,

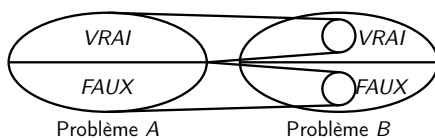
► pour obtenir \leq .

14

La notion de réduction

- Soient A et B deux problèmes d'alphabets respectifs M_A et M_B . Une **réduction de A vers B** est une fonction $f : M_A^* \rightarrow M_B^*$ calculable en **temps polynomial** telle que

$$w \in A \text{ ssi } f(w) \in B.$$



- On note $A \leq B$ lorsque A se réduit à B .
 - intuitivement : $A \leq B$ signifie que A est plus facile que B .

15

Principales propriétés

Théorème

\leq est un **préordre** (= est reflexive, transitive) :

- $L \leq L$;
- $L_1 \leq L_2, L_2 \leq L_3$ impliquent $L_1 \leq L_3$.

- intuitivement : un problème est aussi facile (et difficile) que lui-même, et la relation "être plus facile que" est transitive.

Théorème

Si $A \leq B$, et si B est dans P alors A est dans P

- intuitivement : si un problème est plus facile qu'un problème P, alors il est dans P.

Théorème

Si $A \leq B$, et si A n'est pas dans P, alors B n'est pas dans P.

- intuitivement : si un problème est plus difficile qu'un problème qui n'est pas P, alors il n'est pas dans P.

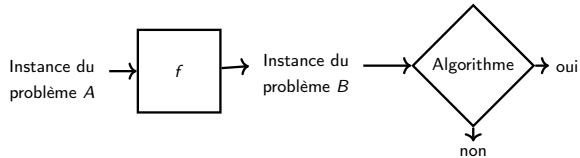
16

■ Preuve du premier théorème :

- ▶ Considérer la fonction identité pour f pour le premier point. Pour le second point, supposons $L_1 \leq L_2$ via la réduction f , et $L_2 \leq L_3$ via la réduction g . On a $x \in L_1$ ssi $g(f(x)) \in L_2$. La composée de deux fonctions calculables **en temps polynomial** est calculable.

■ Preuve du second théorème :

- ▶ A est décidé par la machine de Turing qui, sur une entrée w , calcule $f(w)$, puis simule la machine de Turing qui décide B sur l'entrée $f(w)$. Puisqu'on a $w \in A$ si et seulement si $f(w) \in B$, l'algorithme est correct. Il fonctionne **en temps polynomial avec les hypothèses**.



- Le troisième théorème est la contraposée du second.

17

Lequel est le plus difficile ?

■ 3-COLORABILITE \leq SAT :

- ▶ voir transparent 27 du cours 2.
- ▶ Théorème de Cook-Levin, à venir.

■ SAT \leq 3-COLORABILITE ? :

- ▶ voir transparent 35

18

Le transparent 27 du cours 2 :

- On se donne un graphe $G = (V, E)$ et k couleurs.
- On considère $\mathcal{P} = \{A_{u,i}, u \in V, 1 \leq i \leq k\}$ un ensemble de variables propositionnelles.
- Idée : $A_{u,i}$ vraie ssi le sommet u est colorié avec la couleur i .
- Contraintes :
 - ▶ Chaque sommet possède une couleur :

$$\Gamma_1 = \{A_{u,1} \vee \dots \vee A_{u,k} \mid u \in V\}.$$
 - ▶ Chaque sommet n'a pas plus qu'une couleur :

$$\Gamma_2 = \{\neg(A_{u,i} \wedge A_{u,j}) \mid u \in V, 1 \leq i, j \leq k, i \neq j\}.$$
 - ▶ Chaque arête n'a pas ses extrémités d'une même couleur :

$$\Gamma_3 = \{\neg(A_{u,i} \wedge A_{v,i}) \mid u \in V, 1 \leq i \leq k, (u, v) \in E\}.$$
- Un graphe est coloriable avec k couleurs si et seulement si on peut satisfaire toutes les formules de $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \Gamma_3$.

19

- On ne sait pas si 3-COLORABILITE et SAT admettent ou pas un algorithme polynomial.
- ▶ Cependant, il est très clair que 3-COLORABILITE et SAT admettent un **vérificateur polynomial**.

20

Vérificateur

- Un **vérificateur** pour un problème A est un algorithme V tel que A s'écrit

$$A = \{w \mid V \text{ accepte } (w, u) \text{ pour un certain mot } u\}.$$

- ▶ Autre façon de l'écrire :

$$w \in A \text{ ssi } \exists u, V \text{ accepte } (w, u).$$

- ▶ Autrement dit, un vérificateur utilise une information en plus, à savoir u pour vérifier que w est dans A .

- Le mot u est alors appelé un **certificat** (parfois aussi une **preuve**) pour w .

- Le vérificateur est polynomial si V décide en temps polynomial en la **longueur de w** .
- On dit qu'un langage est **polynomialement vérifiable** s'il admet un vérificateur polynomial.

21

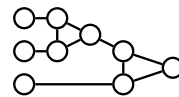
Exemple 1 : 3-COLORABILITE

- Certificat u :

- ▶ $u = c_1 c_2 \dots c_n$ donne les couleurs de chacun des sommets.

- Vérificateur :

- ▶ l'algorithme pour BON COLORIAGE.



- $G \in 3\text{-COLORABILITE}$ ssi $\exists u, (G, u) \in \text{BON COLORIAGE}$.

- Vérifier que u est un coloriage de G se fait bien en un temps polynomial en la taille de G .

22

Exemple 2 : SAT

- Certificat u :

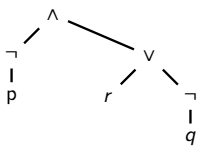
- ▶ $u = x_1 x_2 \dots x_n \in \{0, 1\}^n$ donne la liste de valeurs de chaque variable.

- Vérificateur :

- ▶ l'algorithme V pour évaluer une formule en calcul propositionnel à partir d'une formule F et de la valeur de ses variables.

- $F \in \text{SAT}$ ssi $\exists u, (F, u) \in V$.

- Vérifier que u rend la formule F vraie se fait bien en un temps polynomial en la taille de F .



23

La classe NP

- NP est la classe des problèmes (langages) qui possèdent un vérificateur polynomial.

- ▶ NP contient un nombre incroyable de problèmes d'intérêt pratique.

- ▶ Parenthèse/Attention : le N dans NP ne vient pas de "non", "not", mais de "non-déterministe".

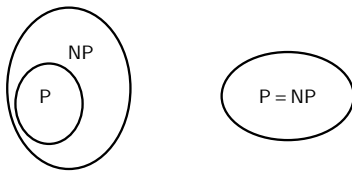
- On peut montrer que NP correspond aux problèmes décidés en temps polynomial par une machine de Turing non-déterministe.

- ▶ Par définition $P \subset NP$.

24

La question $P = NP$?

- La question de savoir si l'inclusion est stricte est la question ouverte la plus connue et la plus célèbre de l'informatique.
 - ▶ Une des deux possibilités est correcte : mais laquelle ?



- ▶ Placée parmi la liste des questions les plus importantes pour les mathématiques et l'informatique pour le millénaire en 2000.
- ▶ Le Clay Mathematics Institute offre 1000000 de dollars à qui déterminerait la réponse à cette question.
- ▶ Cette question théorique a des implications très concrètes : par exemple pour la sûreté de nombreuses procédures cryptographiques.
- ▶ On conjecture généralement que l'inclusion est stricte.

25

Motivation

- On souhaite comprendre quels sont les problèmes les plus difficiles dans NP.
 - ▶ ou plus généralement dans une classe \mathcal{C} de problèmes de décision.
- Un problème A est dit NP-difficile si tout problème B de NP est tel que $B \leq A$.
 - ▶ plus généralement : Un problème A est dit \mathcal{C} -difficile si tout problème B de \mathcal{C} est tel que $B \leq A$.
 - ▶ Intuitivement : il est plus difficile que tous les problèmes dans la classe.
- Un problème A est dit NP-complet si en plus on a $A \in NP$.
 - ▶ plus généralement : Un problème A est dit \mathcal{C} -complet si en plus on a $A \in \mathcal{C}$.
 - ▶ Autrement dit : A est NP-complet signifie que A est un élément maximum dans NP pour \leq .

26

Théorème (Cook-Levin)

- ▶ Le problème SAT est NP-complet.

(on verra la preuve plus tard)

27

Conséquences

Corollaire

$P = NP$ si et seulement si $SAT \in P$.

- Preuve :
 - ▶ Puisque SAT est dans NP, si $P = NP$, alors $SAT \in P$.
 - ▶ Réciproquement, puisque SAT est complet, pour tout problème $B \in NP$, $B \leq SAT$ et donc $B \in P$ si $SAT \in P$.
- Se généralise à n'importe quel problème NP-complet :
 - ▶ Soit A un problème NP-complet.
 $P = NP$ si et seulement si $A \in P$.
 - ▶ D'où l'intérêt de produire de nombreux problèmes NP-complets.

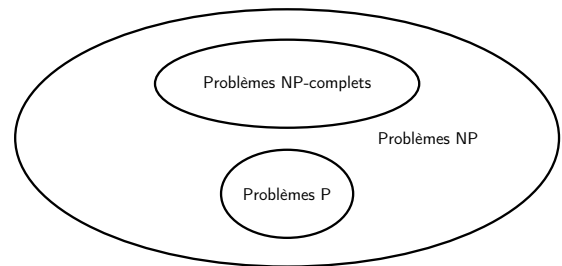
28

A quoi sert de prouver la NP-complétude d'un problème ?

- Arriver à prouver que $P = NP$...
 - ▶ Remarque : Si un problème A et un problème B sont NP-complets alors $A \leq B$ et $B \leq A$:
 - Tous les problèmes NP-complets sont donc de même difficulté.
- Surtout :
 - ▶ Supposons que l'on n'arrive pas à trouver un algorithme polynomial pour un problème.
 - ▶ Prouver sa NP-complétude permet de se convaincre que cela n'est pas possible,
 - sauf si $P = NP$.
 - ▶ et qu'il faut donc chercher une autre approche (voir cours à venir, exemple : approximations).

29

Situation avec l'hypothèse $P \neq NP$.



30

Stratégie pour prouver la NP-complétude

- Pour prouver la NP-complétude d'un problème A , il suffit :
 1. de prouver qu'il admet un vérificateur polynomial ;
 2. et de prouver que $B \leq A$ pour un problème B que l'on sait déjà NP-complet.
- Pourquoi ?
 - ▶ En effet, le point 1. permet de garantir que $A \in NP$,
 - ▶ et le point 2. que pour tout problème $C \in NP$ on a $C \leq A$:
 - en effet, par la NP-complétude de B on a $C \leq B$, et puisque $B \leq A$, on obtient $C \leq A$.
- **Attention, la NP-complétude d'un problème A s'obtient en prouvant qu'il est plus difficile qu'un autre problème NP-complet, et pas le contraire.**
 - ▶ C'est une erreur fréquente dans les raisonnements.

31

Une autre raison pour prouver la NP-complétude de différents problèmes

- Cette stratégie nécessite de posséder une base la plus large possible de problèmes connus pour être NP-complets.
 - ▶ Plus de 300 répertoriés en 1979 dans le livre de Garey & Johnson.

32

3-SAT est NP-complet

Problème 3-SAT:

Donnée: Une formule F sous la forme d'une conjonction de disjonction de 3 littéraux (variables ou leur négation).

- Formellement : une formule

$$F = C_1 \wedge C_2 \cdots \wedge C_\ell$$

avec

$$C_i = y_{i,1} \vee y_{i,2} \vee y_{i,3},$$

où pour tout i, j ,

$y_{i,j}$ est soit x_k , soit $\neg x_k$ pour l'un des x_k ,
pour des variables $\{x_1, \dots, x_n\}$.

Réponse: Décider si F est satisfiable.

Theorem

Le problème 3-SAT est NP-complet.

- On appelle souvent théorème de Cook-Levin ce théorème, plus facile à utiliser que l'autre.

33

Preuve

- 3-SAT est dans NP.

- La donnée d'une valeur dans $\{0,1\}$ pour chaque variable constitue un certificat vérifiable en temps polynomial.

- SAT \leq 3-SAT :

- Soit F une formule SAT.

- Soit C une clause de F , par exemple

$$C = x \vee y \vee \neg z \vee u \vee \neg v \vee w \vee t.$$

- On introduit de nouvelles variables a, b, c, d associées à cette clause, et on remplace C par la formule $R(C) = (x \vee y \vee a) \wedge (\neg a \vee \neg z \vee b) \wedge (\neg b \vee u \vee c) \wedge (\neg c \vee \neg v \vee d) \wedge (\neg d \vee w \vee t)$.
- Il est facile de vérifier qu'une assignation de x, y, z, u, v, w, t peut être complétée par une assignation de a, b, c, d de façon à rendre cette formule vraie si et seulement si C est vraie.

- A partir de $F = C_1 \wedge C_2 \wedge \cdots \wedge C_\ell$, on construit

$$F' = R(C_1) \wedge R(C_2) \wedge \cdots \wedge R(C_\ell).$$

- On a $F \in \text{SAT}$ ssi $F' \in 3\text{-SAT}$.

- La fonction qui à F associe F' se calcule bien en temps polynomial.

34

Théorème

3-COLORABILITE est NP-complet.

35

Preuve

- 3-COLORABILITE est dans NP : déjà vu.

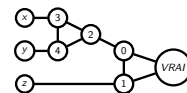
- 3-SAT \leq 3-COLORABILITE :

- On se donne donc une conjonction F de clauses à 3 littéraux, et il nous faut à partir de là construire un graphe.

- Il faut parvenir à traduire deux contraintes : une variable peut prendre la valeur 0 ou 1 d'une part, et les règles d'évaluation d'une clause d'autre part.

- On construit un graphe ayant $3 + 2n + 5m$ sommets, où n est le nombre de variables, m de clauses.

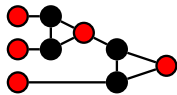
- Les trois premiers, notés VRAI, FAUX, NSP sont reliés deux à deux en triangle.
- On associe pour chaque variable x_j un sommet x_j et un sommet $\neg x_j$: on relie deux à deux en triangle les sommets x_j , $\neg x_j$, et NSP.
- On ajoute 5 sommets pour chaque clause $x \vee y \vee z$ selon le dessin suivant :



36

Le gadget

- Si l'on impose que les trois sommets à gauche sont soit bleus soit rouges,
 - ▶ alors, on peut colorier le sommet le plus à droite en bleu si et seulement si au moins un sommet à gauche est en bleu.



37

- Si F est satisfiable, on peut colorier le graphe $G(F)$ avec 3-couleurs :
 - ▶ mettre les variables vraies avec la couleur du sommet *VRAI*, les variables fausses avec la couleur du sommet *FAUX*, et compléter.
- Si le graphe $G(F)$ est coloriable avec trois couleurs :
 - ▶ Le triangle 1. assure que *VRAI* et *FAUX* ne sont pas de la même couleur.
 - ▶ Le triangle 2. assure que x_i et $\neg x_i$ ne sont pas de la même couleur pour chaque variable x_i , et soit de la couleur de *VRAI* ou de la couleur de *FAUX*.
 - ▶ Le gadget 3. assure que pour chaque clause C au moins x ou y ou z est de la couleur de *VRAI*.
 - ▶ En prenant pour vraies toutes les variables de la couleur de *VRAI*, on obtient une affectation des variables qui satisfait la formule F .
- Bref : $F \in 3\text{-SAT}$ ssi $G(F) \in 3\text{-COLORABILITE}$.
- La fonction qui à F associe $G(F)$ se calcule bien en temps polynomial.

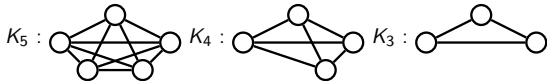
38

Sur les graphes

■ CLIQUE

Donnée: Un graphe $G = (V, E)$ non-orienté et un entier k .

Réponse: Décider s'il existe $V' \subset V$, avec $|V'| = k$, tel que $u, v \in V' \Rightarrow (u, v) \in E$.



En passant au complémentaire sur les arêtes :

■ STABLE

Donnée: Un graphe $G = (V, E)$ non-orienté et un entier k .

Réponse: Décider s'il existe $V' \subset V$, avec $|V'| = k$, tel que $u, v \in V' \Rightarrow (u, v) \notin E$.

En passant au complémentaire sur les sommets :

■ RECOUVREMENT DE SOMMETS

Donnée: Un graphe $G = (V, E)$ non-orienté et un entier k .

Réponse: Décider s'il existe $V' \subset V$, avec $|V'| \leq k$, tel que toute arête de G ait au moins une extrémité dans V' .

39

Sur les graphes

■ CIRCUIT HAMILTONIEN

Donnée: Un graphe $G = (V, E)$ (non-orienté).

Réponse: Décider s'il existe un **circuit hamiltonien**, c'est-à-dire un chemin de G passant une fois et une seule par chacun des sommets et revenant à son point de départ.

■ VOYAGEUR DE COMMERCE

Donnée: Un couple (n, M) , où M est une matrice $n \times n$ d'entiers et un entier k .

Réponse: Décider s'il existe une permutation π de $[1, 2, \dots, n]$ telle que

$$\sum_{1 \leq i \leq n} M_{\pi(i)\pi(i+1)} \leq k.$$

- ▶ Ce problème porte ce nom, car on peut voir cela comme l'établissement de la tournée d'un voyageur de commerce devant visiter n villes, dont les distances sont données par la matrice M de façon à faire moins de k kilomètres.

40

Sur les entiers

■ SOMME DE SOUS ENSEMBLE

Donnée: Une suite finie d'entiers x_1, x_2, \dots, x_n et un entier t .

Réponse: Décider s'il existe $E \subset \{1, 2, \dots, n\}$ tel que $\sum_{i \in E} x_i = t$.

■ PARTITION

Donnée: Une suite finie d'entiers x_1, x_2, \dots, x_n .

Réponse: Décider s'il existe $E \subset \{1, 2, \dots, n\}$ tel que $\sum_{i \in E} x_i = \sum_{i \notin E} x_i$.

■ SAC A DOS

Donnée: Un ensemble de poids a_1, \dots, a_n , un ensemble de valeurs v_1, \dots, v_n , un poids limite A , et un entier V .

Réponse: Décider s'il existe $E' \subset \{1, 2, \dots, n\}$ tel que $\sum_{i \in E'} a_i \leq A$ et $\sum_{i \in E'} v_i \geq V$.