1.1

Résolution de problèmes

Jean-Charles Régin

Licence Informatique 2ème année

Remerciements

- Wikipédia
- Patrick Lester (images A*)
- Cours et TD de l'ENS Lyon (Yves Robert, Yves Caniou et Eric Thierry)

But du cours

- Vous faire comprendre que les ordinateurs peuvent aussi être utilisés pour résoudre des problèmes complexes
  - Il n'y a pas que le web et la video !
- Utilisation intelligente de la puissance de calcul

Organisation

- Des cours et des TDs et des TPs
- Horaires annoncés au fur et à mesure
- Un projet à faire en binome
  - Programmer le jeu d'Awalé. L'ordinateur doit jouer !
  - Regarder les règles sur wikipédia
- Concours entre projets !

Contrôle des connaissances

- Contrôle 40%
- Projet 40%
- Compétition 20%

Plan du cours

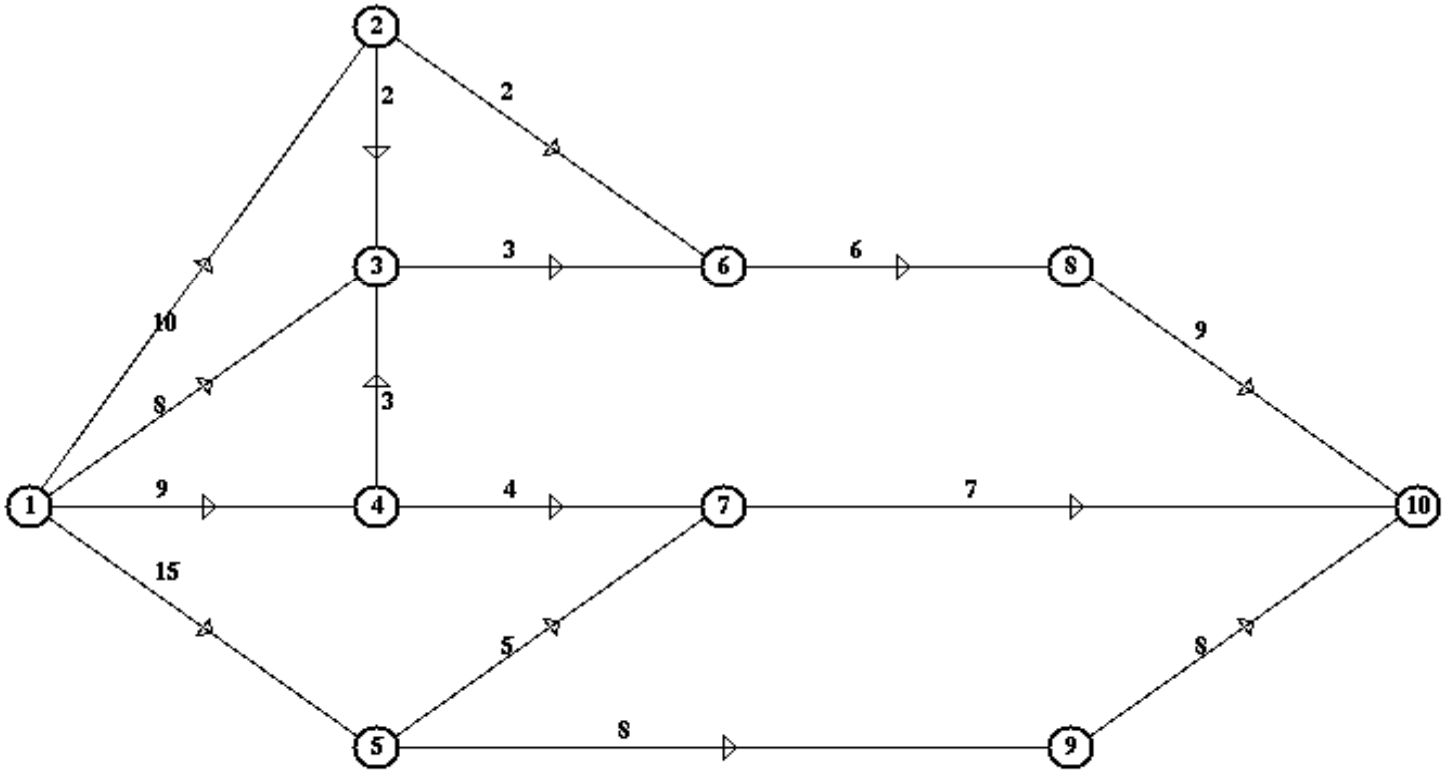
- Introduction
- Algorithmes gloutons
- Problèmes de cheminement (graphe d'états)
- Enumération exhaustive
- Problème à deux joueurs

1.7 Introduction

Graphe : définitions

- Un Graphe Orienté G=(X,U) est déterminé par la donnée :
  - d'un ensemble de **sommets** ou **nœuds** X
  - d'un ensemble ordonné U de couples de sommets appelés **arcs**.
- Si u=(i,j) est un arc de G alors i est l'extrémité initiale de de u et j l'extrémité terminale de u.
- Les arcs ont un sens. L'arc u=(i,j) va de i vers j.
- Ils peuvent être munit d'un coût, d'une capacité etc…

Graphe



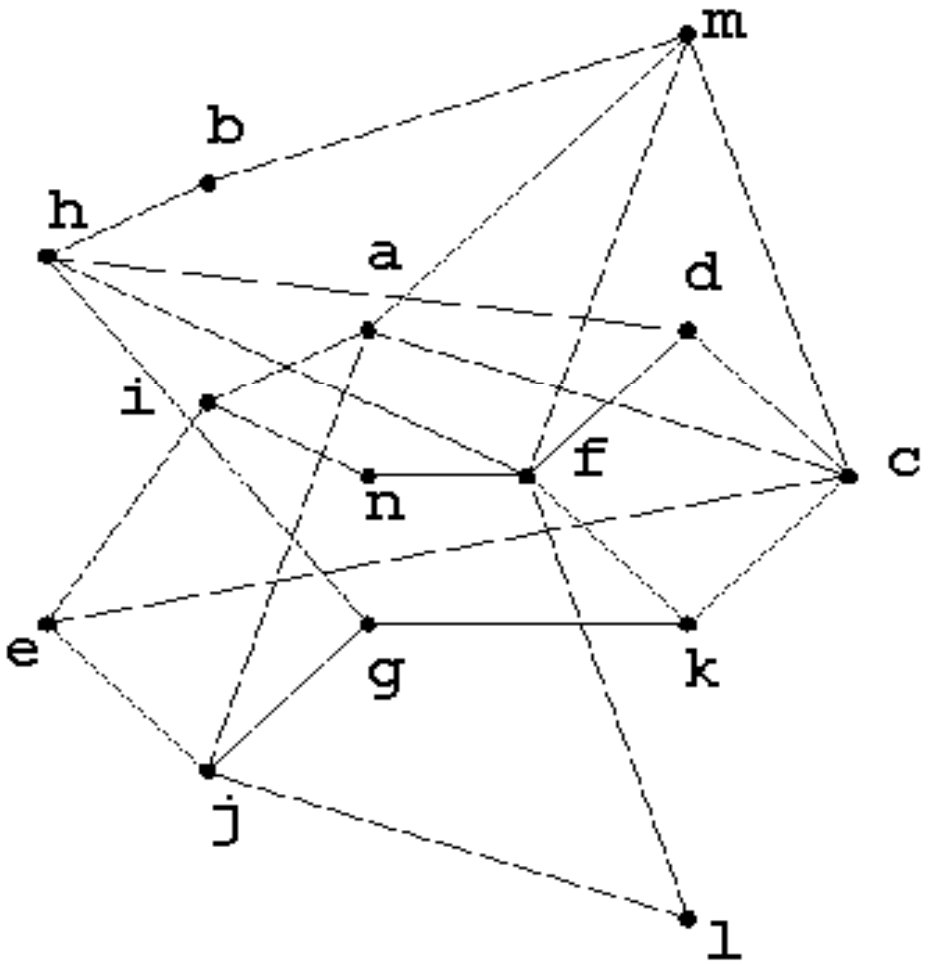
Graphe

- On note par $\omega(i)$: l'ensemble des arcs ayant i comme extrémité
- On note par $\omega^+(i)$: l'ensemble des arcs ayant i comme extrémité initiale = ensemble des arcs sortant de i
- On note par $\omega^-(i)$: l'ensemble des arcs ayant i comme extrémité terminale = ensemble des arcs entrant dans i
- N(i) : ensemble des voisins de i : ensemble des sommets j tels qu'il existe un arc contenant i et j

Graphe non orienté

- Un Graphe non orienté G=(X,E) est déterminé par la donnée :
  - d'un ensemble de sommets ou nœuds X
  - D'un ensemble E de paires de sommets appelées **arêtes**
- Les arêtes ne sont pas orientées

Graphe non orienté



Graphe : définitions

- Deux sommets sont voisins s'ils sont reliés par un arc ou une arête
- Degré entrant d'un sommet i : nombre d'arcs ayant i comme extrémité terminale = nombre d'arc entrant dans i
- Degré sortant d'un sommet i : nombre d'arcs ayant i comme extrémité initiale = nombre d'arcs sortant de i

Gestion de projet : Ordonnancement

- Gantt chart
- Graphe : définitions
- **Ordonnancement**
- Graphe Potentiel Tâches
  - Fonction rang d'un graphe
- Graphe Potentiel Etapes (PERT)

Graphe : définitions

- Chemin de longueur q : séquence de q arcs $\{u_1,u_2,\ldots,u_q\}$ telle que
  - $u_1=(i_0,i_1)$
  - $u_2 = (i_1,i_2)$
  - $u_q=(i_{q-1},i_q)$
- Chemin : tous les arcs orientés dans le même sens
- Circuit : chemin dont les extrémités coïncident

Plan

- Petite histoire du TSP
- Problèmes faciles et problèmes difficiles
- Problèmes d'optimisation et de décision
- Problèmes difficiles : limites à la résolution

Problème

- C'est une question générale : plus court chemin entre deux points, emploi du temps.
- Décrit par les données et une question
- Répondre à cette question c'est résoudre le problème.
- En informatique, on veut une réponse générale à ce problème, autrement dit un algorithme qui marche dans tous les cas.
- **Instance** = un jeu de données particulier. Par exemple plus court chemin entre Nice et Nantes

Introduction

- Certains problèmes sont faciles :
  - Trier des nombres
  - Inverser une chaine…
- D'autres sont difficiles
  - Exemple : le TSP

Traveling Salesman Problem (TSP)

- **Données :** une liste de villes et leurs distances deux à deux.
- **Question :** trouver le plus petit tour qui visite chaque ville exactement une fois.

- Reformulation plus mathématique :
  - Etant donné un graphe complet pondéré, trouver un cycle hamiltonien de poids minimum

TSP

Chaque ville est visitée une et une seule fois

Un seul tour (pas de sous-tour)

TSP

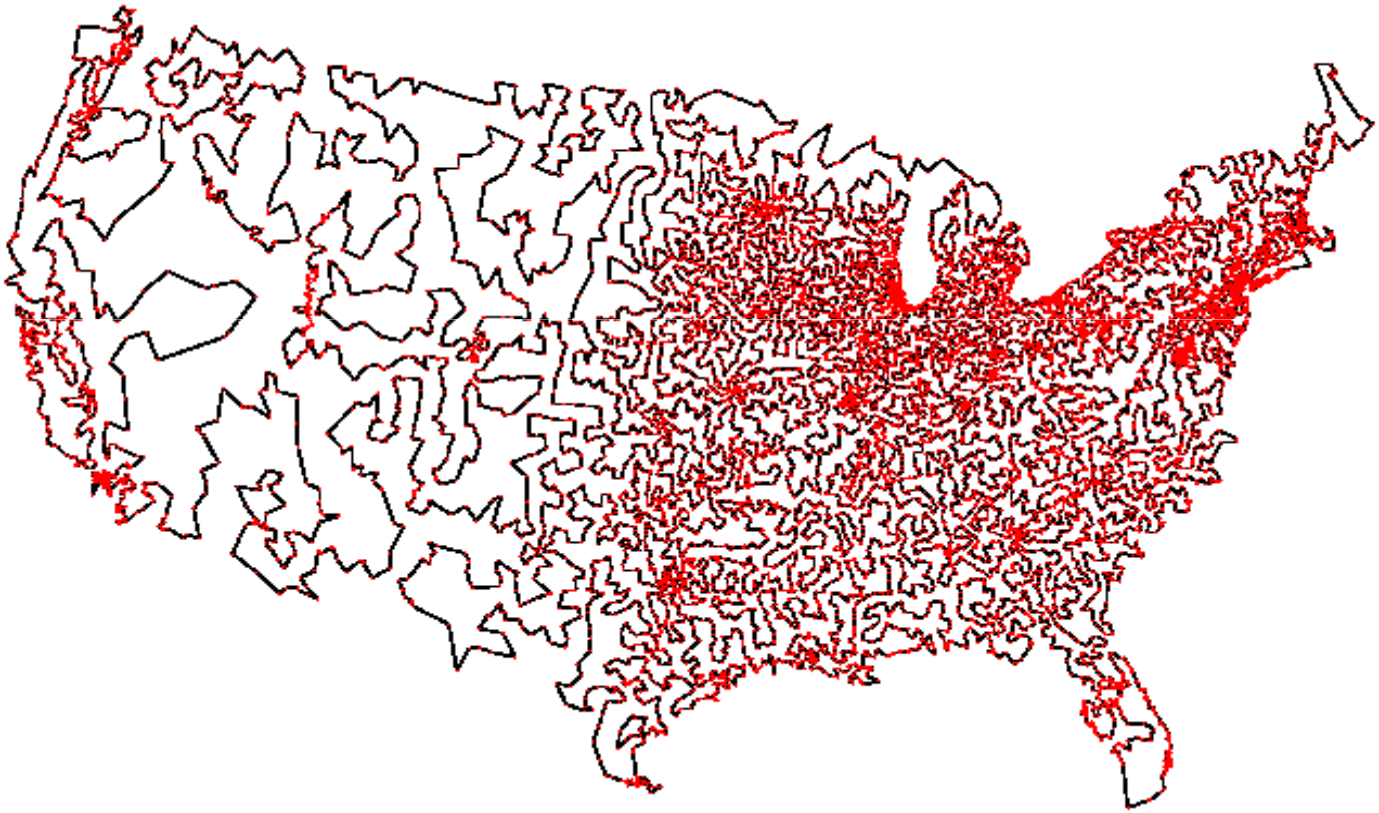
Chaque ville est visitée une et une seule fois

Un seul tour (pas de sous-tour)

TSP

- Certains problème sont équivalent au problème du TSP. Exemple : problème d'ordonnancement : trouver l'ordre dans lequel on doit construire des objets avec des presses hydrauliques
- La version « pure » du TSP n'est pas fréquente en pratique. On a souvent des problèmes qui sont
  - Non euclidiens
  - Asymétriques
  - Recouvrement de sous-ens de nœuds et non pas de tous les nœuds
- Ces variations ne rendent pas le problème plus facile.
- Problème assez commun
  - Vehicle routing (time windows, pickup and delivery…)

Procter and Gamble Contest 1962

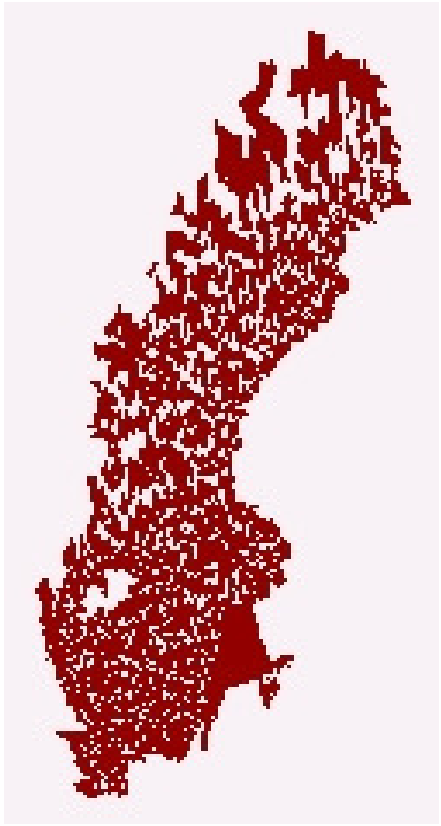


USA 13,509 cities. Solved in 1998

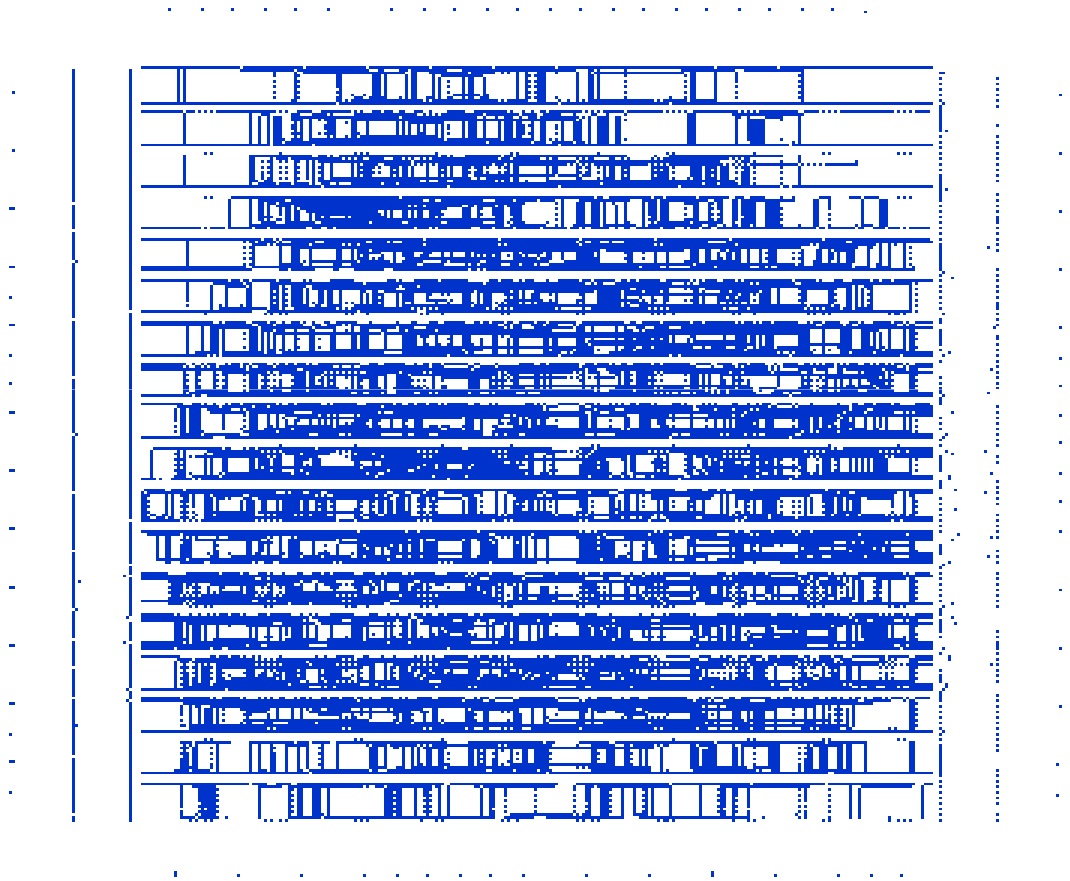
German Tour

- Le 20 Avril 2001, David Applegate, Robert Bixby, Vašek Chvátal, et William Cook ont anoncé la résolution du TSP pour les 15 112 villes d'Allemagne.
- Réseau de 110 processeurs (550 Mhz) à l'université de Rice et de Princeton.
- Le temps total d'utilisation des ordinateurs a été de **22.6 années.**

Germany 15,112 cities. Solved in 2001



Sweden 24,978 cities. Solved in 2004



VLSI 85,900. Solved in 2006

TSP

- Il existe plusieurs solveurs
- Le plus connu est Concorde de William Cook (gratuit)

TSP Solvers

- Les solveurs de TSP sont généralement dédiées à la résolution du problème pur
- Il est presque impossible de les utiliser si on change un tout petit peu le problème (asymétrique, contraintes annexes …)

Plan

- Petite histoire du TSP
- **Problèmes faciles et problèmes difficiles**
- Problèmes d'optimisation et de décision
- Problèmes difficiles : limites à la résolution

Algorithmes

- Tous les algorithmes ne sont pas équivalents. On les différencie selon 2 critères
  - Temps de calcul : lents vs rapides
  - Mémoire utilisée : peu vs beaucoup
- On parle de complexité en temps (vitesse) ou en espace (mémoire utilisée)

Pourquoi faire des algorithmes rapides ?

- Pourquoi faire des algo efficaces ? Les ordinateurs vont super vite !
- Je peux faire un algorithme en n^2 avec n=10000
- Je voudrais faire avec n=100000. Tous les 2 ans la puissance des ordinateurs est multipliée par 2 (optimiste). Quand est-ce que je pourrais faire avec n=100000 ?

Pourquoi faire des algorithmes rapides ?

- Pourquoi faire des algorithmes efficaces ? Les ordinateurs vont super vite !
- Je peux faire un algorithme en n^2 avec n=10000
- Je voudrais faire avec n=100000. Tous les 2 ans, la puissance des ordinateurs est multipliée par 2 (optimiste). Quand est-ce que je pourrais faire avec n=100000 ?

- Réponse:
  - p=100000 q=10000; p=10*q; donc $p^2 = 100*q^2$. Donc besoin de 100 fois plus de puissance
  - $2^6 = 64$, $2^7 = 128$ donc obtenue dans 7*2 ans=14 ans !!!
- Je trouve un algo en n log(n) pour p, je ferai 17*100000= 1 700 000 opérations donc 100/1.7 fois moins de temps !!!

Les ordinateurs vont super vite

- Idée reçue : « les ordinateurs vont tellement vite qu'ils sont capables de calculer toutes les solutions »
- jeu d'échecs :
  - un joueur a environ 10 coups a chaque fois,
  - je joue 10 coups, mon adversaire joue : 10 coups pour chacun de mes 10 coups.
  - je joue 2 fois 10(moi)x10(lui)x10(moi) combinaisons à envisagés.
  - je joue 3 fois : 10(moi)x10(lui)x10(moi)x10(lui)x10(moi).
- Je joue k fois 10^{k+2} combinaisons

Les échecs

- Je joue k fois 10^{k+2} combinaisons
- Je joue 5 fois 10 000 000 = 10 millions
- Il y a 20 ans (depuis 1993)
  - Le meilleur programme d'échec était du niveau d'un MI
  - Les ordinateurs allaient à 100Mhz
- Aujourd'hui
  - Le meilleur programme est le meilleur joueur du monde. Il ne coute rien ($50 ou $100)
  - Les ordinateurs vont à 3Ghz

Les échecs

- Je joue k fois 10^{k+2} combinaisons
- Accroissement de puissance des ordinateurs
  - fréquence : 3Ghz vs 100 Mhz
  - multicoeurs vs mono cœur
  - meilleur code engendré (compilateurs)
  - meilleur architecture des processeurs (fabricant CPU)
  - Dhrystone benchmark (date de 1984) : calcul uniquement sur des entiers
    - Pentium (100Mhz) Dhry2 opt = 122,
    - Core i7 930 (3Ghz) Dhry 2 opt = 8684
    - Rapport : 8684/122 = 71,18. (30 pour freq)

Les échecs

- Je joue k fois 10^{k+2} combinaisons
- 70 fois plus vite en 18 ans (30 pour la fréquence, 2,5 pour l'architecture)
  - 1 coup = 10 (moi) * 10 (lui) = 100
  - En gros en 20 ans on a gagné 1 coup pour un monocoeur
- 2 vrais coups (2 pour moi et 2 pour lui) = 100*100=10000.
  - Il faudrait 100 cœurs !
- **Donc on ne résout pas le problème uniquement en comptant sur l'augmentation de performance des ordinateurs**

Complexité des algorithmes

- **But:**
  - **avoir une idée de la difficulté des problèmes**
  - **donner une idée du temps de calcul ou de l'espace nécessaire pour résoudre un problème**
- Cela va permettre de comparer les algorithmes
- Exprimée en fonction du nombre de données et de leur taille.
- A défaut de pouvoir faire mieux :
  - On considère que les opérations sont toutes équivalentes
  - Seul l'ordre de grandeur nous intéresse.
  - On considère uniquement le pire des cas
- Pour n données on aura : O(n) linéaire, $O(n^2)$ quadratique O(n log(n)),…

Algorithmes : vitesse

- On peut qualifier de rapide un algorithme qui met un temps polynomial en n (nombre de données) pour être exécuté. Exemple n^2, n^8

NP-Complétude

- **LE problème** majeur de l'informatique actuelle :
  - P vs NP
- On ne rentre pas dans le détail
  - Facile = algorithme polynomial
  - Difficile = pas d'algorithme polynomial connu
  - Existe-t-il toujours un algorithme polynomial ? C'est **LA question**

NP-Complétude

- Pour certains problèmes : voyageur de commerce, remplissage de sac-à-dos de façon optimum. On ne sait pas s'il existe un algorithme rapide. On connaît des algorithmes exponentiels en temps : 2^n.
- 1 million de $, si vous résolvez la question!

Exemple de problèmes difficiles

- Hitting-set : Recouvrement (set cover)
  - Ampoules et Interrupteurs :
    - Un interrupteur est relié à certaines ampoules
    - Si on appuie sur l'interrupteur alors on allume toutes les ampoules reliées
    - Question : sur combien d'interrupteur au minimum doit-on appuyer pour allumer toutes les ampoules.
    - On veut une réponse générale qui marche pour toutes les instances

Exemple de problèmes difficiles

- Hitting-set : Recouvrement (set cover)
- Somme (Subset sum)
- Sac à dos (knapsack)
- Rangement (Bin packing)
- Coloriage (graph coloring)
- Clique maximale

Problèmes difficiles

- Les problèmes NP-Complets sont tous équivalents !
- Ils se ressemblent tous. On passe de l'un à l'autre

Coloriage des sommets d'un graphe

- Principe de reliements-contraction
- Pour colorier un graphe complet (une clique) contenant n sommets il faut n couleurs
- On prend 2 sommets a et b non reliés
  - Soit ils ont la même couleur
  - Soit ils ont une couleur différente
- Même couleur = contraction
- Couleur différente = ajout d'une arête

Coloriage des sommets d'un graphe

- Algorithme :
- On prend 2 sommets a et b non reliés
  - Soit ils ont la même couleur = on les contracte
  - Soit ils ont une couleur différente = on ajoute une arête
- On arrive à une clique : le nombre de sommets donne le nombre de couleurs
- Pb : m aretes potentielles. 2 choix par aretes (meme couleur ou couleur différente) = 2^m

Facile vs Difficile

- La différence peut être subtile. Attention à l'ajout de contraintes additionnelles
- On a une matrice. Pour chaque ligne et pour chaque colonne, on connait le nombre de 1. Définir précisément les 0 et les 1 de cette matrice.
  - Problème pur facile
  - On introduit la connexité : difficile
  - On introduit la convexité : difficile
  - On introduit la connexité et la convexité : facile !

Facile vs Difficile

- Coloriage des arêtes (deux arêtes ayant des sommets communs n'ont pas la même couleur) :
  - Nombre de couleur = degre max ou degre max +1
- Trouver la bonne valeur est un probleme NP-Complet !

Plan

- Petite histoire du TSP
- Problèmes faciles et problèmes difficiles
- **Problèmes d'optimisation et de décision**
- Problèmes difficiles : limites à la résolution

Problème de décision

- Un **problème de décision** est une question mathématiquement définie portant sur des paramètres donnés et demandant **une réponse par oui ou non**.
- Etant donné un ensemble de villes et une distance d, existe-t-il un chemin passant par toutes les villes et de longueur inférieure à d, est un problème de décision
- Peut-on colorier un graphe avec k couleurs ?

Problème d'optimisation

- Un **problème d'optimisation** est le problème qui consiste à trouver la **meilleure solution** d'un ensemble de solutions faisables
- Un problème d'optimisation A pour une instance x est défini par
  - f(x) : l'ensemble des solutions faisables
  - c(y) : la fonction de coût de d'une solution faisable y (coût de la solution).
  - Un objectif qui soit un min, soit un max
- Une solution optimale est une solution faisable qui respecte l'objectif, donc telle que son cout est le minimum (resp. maximum) de toutes les solutions faisables
- Si le but est de minimiser la fonction de coût alors on parle d'un problème de minimisation, sinon on parle d'un problème de maximisation
- Il est souvent possible de passer d'une problème de minimisation à un problème de maximisation en changeant le signe de la fonction de coût.

Problème d'optimisation

- DONNEES : Un graphe G
- QUESTION : Quel est le nombre de minimum de couleurs permettant de colorier G ?

- Plus court chemin entre deux points ?

Problème d'optimisation

- Il faut bien distinguer deux choses
  - Une solution optimale
  - La preuve qu'une solution est bien une solution optimale (preuve d'optimalité)

- Attention à ne pas trop généraliser
  - Trouver l'optimalité et la prouver peuvent etre lent ou rapide.
  - L'un peut être rapide et pas l'autre.

Optimisation et Décision

- A chaque problème d'optimisation correspond un problème de décision qui demande s'il existe une solution ayant une valeur particulière
- Example : On trouve un plus court chemin entre s et t. La valeur est c.
  - Probleme de décision :
    - Existe-t-il un chemin de cout c ?
  - **Preuve d'optimalité**
    - Existe-t-il un chemin de de cout < c ?

Plan

- Petite histoire du TSP
- Problèmes faciles et problèmes difficiles
- Problèmes d'optimisation et de décision
- **Problèmes difficiles : limites à la résolution**

Problèmes Difficiles

- On ne sait pas si les problèmes difficiles peuvent être résolus rapidement.
- Cela veut dire qu'actuellement, on ne sait pas les résoudre efficacement = en temps polynomial. Donc on a une exponentielle qui est toujours présente.

The problem

- n teams and n-1 weeks and n/2 periods
- every two teams play each other exactly once
- every team plays one game in each week
- no team plays more than twice in the same period

	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Period 1	0 vs 1	0 vs 2	4 vs 7	3 vs 6	3 vs 7	1 vs 5	2 vs 4
Period 2	2 vs 3	1 vs 7	0 vs 3	5 vs 7	1 vs 4	0 vs 6	5 vs 6
Period 3	4 vs 5	3 vs 5	1 vs 6	0 vs 4	2 vs 6	2 vs 7	0 vs 7
Period 4	6 vs 7	4 vs 6	2 vs 5	1 vs 2	0 vs 5	3 vs 4	1 vs 3

First model: results

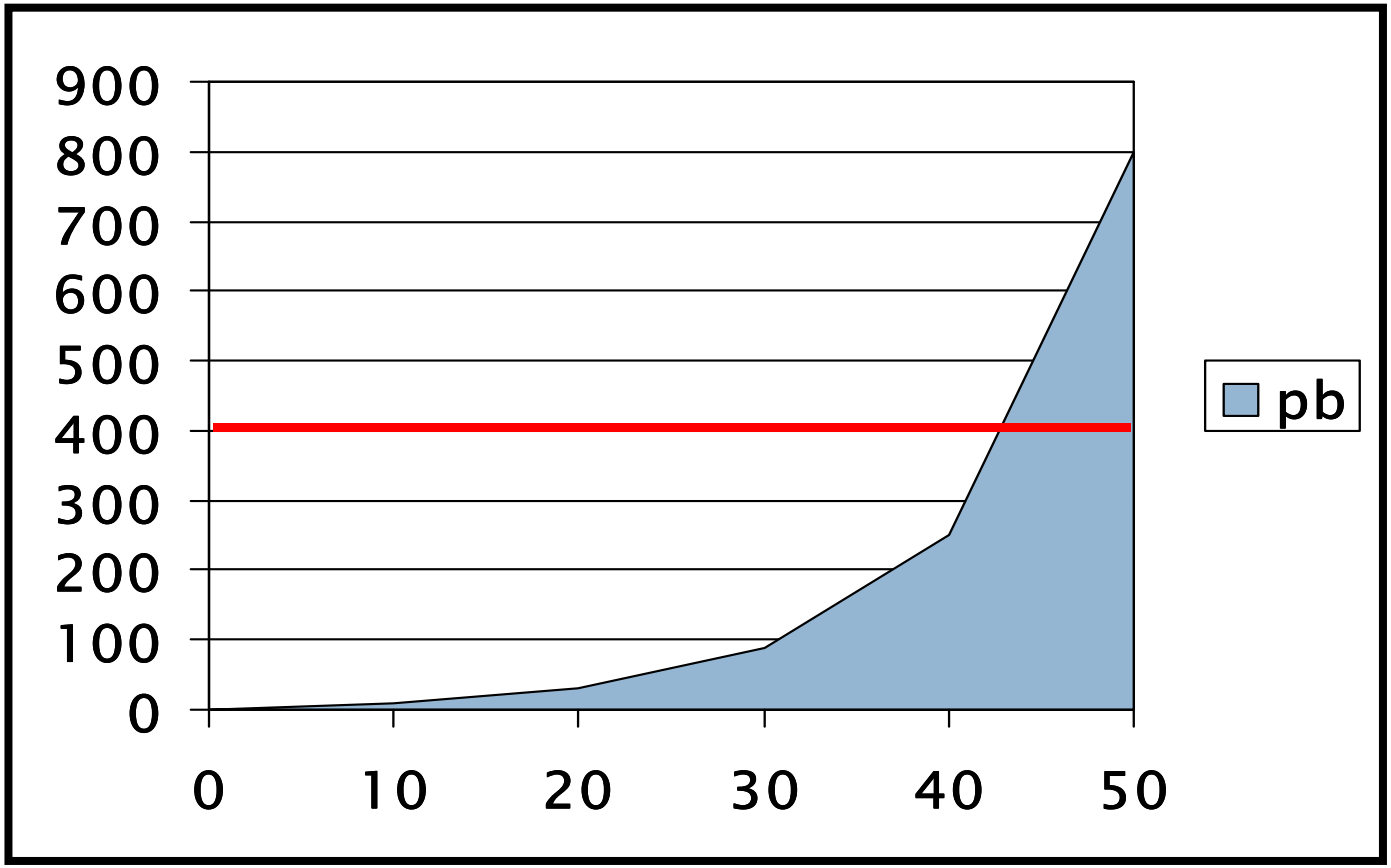
# teams	# fails	Time (in s)
4	2	0.01
6	12	0.03
8	32	0.08
10	417	0.8
12	41	0.2
14	3,514	9.2
16	1,112	4.2
18	8,756	36
20	72,095	338
22	6,172,672	10h
24	6,391,470	12h

Second model: results

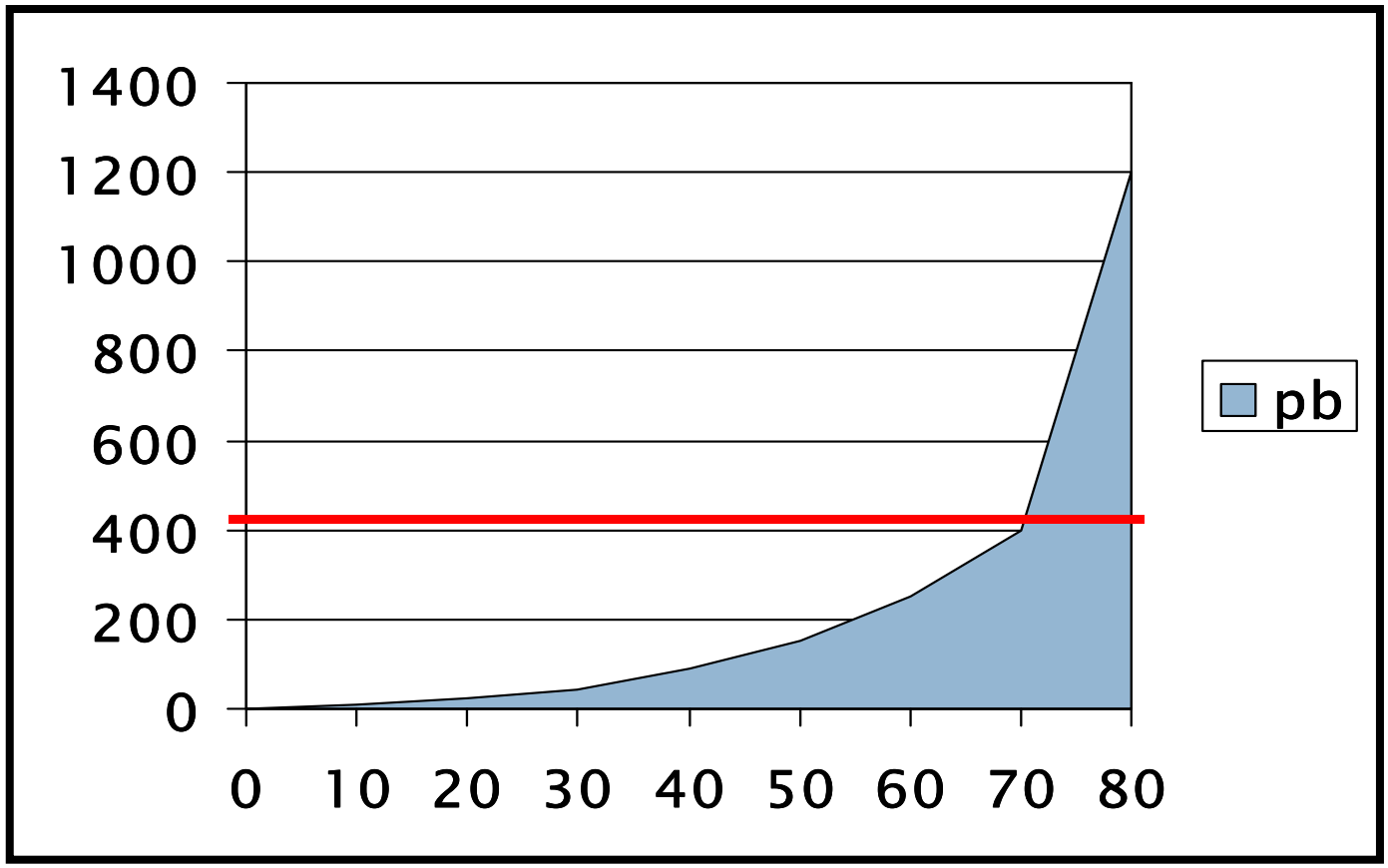
# teams	# fails	Time (in s)
8	10	0.01
10	24	0.06
12	58	0.2
14	21	0.2
16	182	0.6
18	263	0.9
20	226	1.2
24	2702	10.5
26	5,683	26.4
30	11,895	138
40	2,834,754	6h

First model limit

Shifting the exponential



Shifting the exponential



Problème Difficiles

- Dans ce cours : on va voir comment proposer des solutions à ces problemes
  - avec des heuristiques (inexact mais rapide)
  - avec de l'enumeration complete des combinaisons (exact mais lent)

1.64

Algorithmes gloutons

Plan

- Définition
- Le sac à dos
- Heuristique
- Quelques algorithmes gloutons
  - Choix d'activités
  - Coloriage de graphes
  - Couverture de sommets (transversal minimal)

Algorithme glouton

- Un **algorithme glouton** est un algorithme qui suit le principe de faire, étape par étape, un choix optimum local, dans l'espoir d'obtenir un résultat optimum global.
- Dans les cas où l'algorithme ne fournit pas systématiquement la solution optimale, il est appelé une **heuristique gloutonne**.

Algorithme glouton

- La facon dont on fait le choix est parfois appelée **stratégie gloutonne.**
- Exemple : rendre la monnaie avec le minimum de pièces
  - Stratégie gloutonne : on fait plusieurs étapes. A chaque étape on rend la pièce dont le montant est plus petit que ce qu'il reste à rendre et dont la valeur est la plus grande
  - 37 cents à rendre. La plus grande pièce est 20, il reste 17. La plus grande pièce est 10, in reste 7. La plus grande pièce est 5, il reste 2. On rend 1 et 1. On a donc rendu 20+10+5+1+1

Algorithme glouton

- Dans le système de pièces européen l'algorithme glouton donne toujours **une solution optimale.**
- Dans le système de pièces (1, 3, 4), l'algorithme glouton n'est pas optimal, comme le montre l'exemple simple suivant. Il donne pour 6 : 4+1+1, alors que 3+3 est optimal.

Algorithme glouton

- Principe : aller vite
- Difficulté : trouver la bonne stratégie, ou une stratégie efficace.

Plan

- Définition
- **Le sac à dos**
- Heuristique
- Quelques algorithmes gloutons
  - Choix d'activités
  - Coloriage de graphes
  - Couverture de sommets (transversal minimal)

Algorithme glouton : le sac-à-doc

- Le problème du sac-à-dos (knapsack)
  - DONNEES :
    - Un Sac dans lequel on peut mettre un poids limité
    - Un ensemble d'objets. Chaque objet o_i à
      - Un poids : p_i
      - Un coût : c_i
  - QUESTION :
    - Quels sont les objets que l'on doit prendre pour maximizer le coût transporté tout en respectant la contrainte de poids
    - La somme des coûts des objets pris est maximale
    - La somme des poids des objets pris est < poidsmax du sac

Le sac à doc : stratégie

- Une bonne stratégie ?
- Si au lieu d'avoir des objets dont le poids est p et donc que l'on doit prendre en entier ou ne pas prendre, on avait des sacs de poussiere d'or, de poussiere d'argent etc… Est-ce que cela vous aiderait ?

Le sac à doc : stratégie

- On a des sacs de métaux précieux.
- Stratégie:
  - pour chaque sac on calcule la valeur par gramme.
  - 1) On prend le sac ayant le plus grande valeur par gramme et on rempli mon sac à dos avec le plus possible de ce sac
  - 2) si j'ai atteint le poids limite de mon sac à dos alors j'arrête. Sinon, j'ai pris entièrement le contenu d'un sac, j'élimine ce sac et je retourne en 1)
- Cette stratégie est optimale !

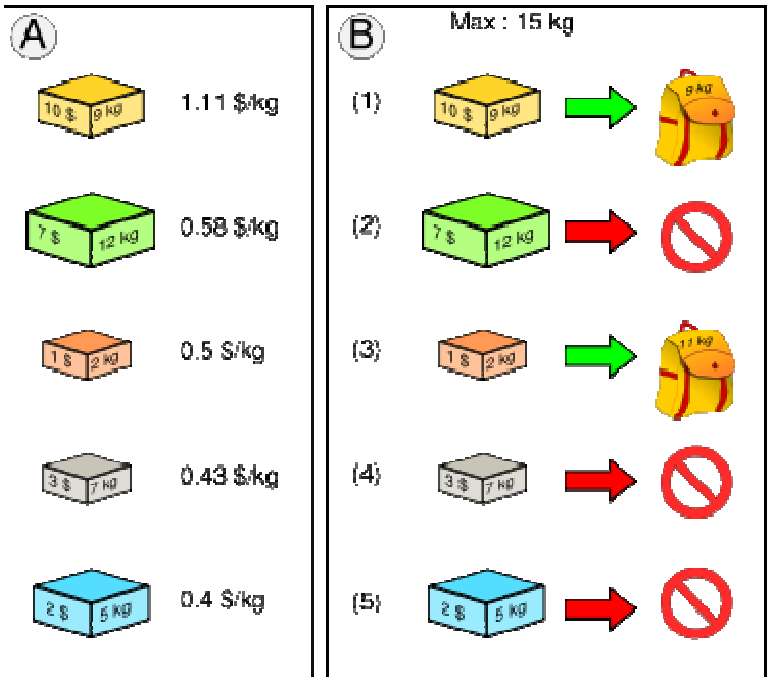
Le sac à dos : stratégie

- Preuve d'optimalité :
- On définit l'efficacité d'un objet o_i : c'est le coût rapporté par gramme.
- Supposons qu'il existe une solution optimale
  - qui ne prend pas un gramme d'un objet o_i et qui prend un gramme d'un objet o_k d'efficacité moindre que o_i.
- Dans ce cas en échangeant 1 g de o_k par 1 g o_i de on améliore la solution. Donc elle n'est pas optimale

Le sac à dos : stratégie

- Mettre des sacs de poudre de métaux précieux revient à accepter de couper des objets
- Si on ne peut pas couper des objets, comment fait-on ?
  - Le problème devient difficile
  - On fait « comme si »
    - On calcule l'efficacité et on prend les objets en fonctions de leur efficacité en respectant la contrainte de poids.

Le sac à dos : stratégie



Solution : 1 + 3, cout $11

Meilleure solution 1 + 5, cout $12

Le sac à dos : modélisation

- Comment représente t'on ce problème ?
- Quelles sont les variables ?
- Comment exprime t'on les contraintes ?

- C'est la modélisation : représentation mathématique du problème. On définit un modèle

Le sac à dos : les variables

- On associe à chaque objet une variable 0-1 (elle ne prend que les valeurs 0 ou 1).
  - C'est une variable d'appartenance au sac à dos
  - Si l'objet est pris alors la variable vaut 1 sinon elle vaut 0
- Le coût d'un objet et son poids sont des données, donc pour l'objet o_i on a le coût ci et le poids w_i.
- La variable d'appartenance au sac est x_i
- Le poids maximum du sac est W

Le sac à dos : les contraintes

$$\max \sum_{i=1}^{n} c_i x_i$$

L'objectif

$$\sum_{i=1}^{n} w_i x_i \leq W$$

La somme des poids des objets pris doit être inférieure ou égal au poids maximal admissible

Plan

- Définition
- Le sac à dos
- **Heuristique**
- Quelques algorithmes gloutons
  - Choix d'activités
  - Coloriage de graphes
  - Couverture de sommets (transversal minimal)

Heuristique gloutonne

- Quand le glouton ne donne pas toujours une solution optimal c'est une **méthode heuristique**

Heuristique

- Vient du grec ancien eurisko « je trouve »
- Une **heuristique** est un algorithme qui fournit rapidement (en temps polynomial) une solution réalisable, pas nécessairement optimale, pour un problème d'optimisation NP-difficile.
- Une heuristique est **une méthode approximative** qui ne donne pas toujours la solution exacte.

Heuristique

- On parle bien souvent de **méthode heuristique**
- Généralement une heuristique est conçue pour un problème particulier, en s'appuyant sur sa structure propre.

Evaluation d'une Heuristique

- **Critère pratique**, ou **empirique** : on implémente l'algorithme approximatif et on évalue la qualité de ses solutions par rapport aux solutions optimales (ou aux meilleures solutions connues). Ceci passe par la mise en place d'un banc d'essai (en anglais *benchmark*, ensemble d'instances d'un même problème accessible à tous).
- **Critère mathématique** : il faut démontrer que l'heuristique garantit des performances. La garantie la plus solide est celle des algorithmes approchés, sinon il est intéressant de démontrer une garantie probabiliste, lorsque l'heuristique fournit souvent, mais pas toujours, de bonnes solutions.

Evaluation d'une Heuristique

- Les critères empiriques et mathématiques peuvent être contradictoire.
- Bien souvent la solution mathématique garantie n'est pas intéressante en pratique.

Heuristique

- Les algorithmes de résolution exacts étant de complexité exponentielle, il est parfois plus judicieux de faire appel à des méthodes heuristiques pour des problèmes difficiles.
- On peut combiner les deux types de méthodes
  - On utilise une heuristique est pour calculer une solution approchée d'un problème et ainsi accélérer le processus de résolution exacte.

Plan

- Définition
- Le sac à dos
- Heuristique
- **Quelques algorithmes gloutons**
  - Choix d'activités
  - Coloriage de graphes
  - Couverture de sommets (transversal minimal)

Choix d'activités

- On considère un gymnase dans lequel se déroulent de nombreuses épreuves : on souhaite en "caser" le plus possible, sachant que deux événements ne peuvent avoir lieu en même temps (il n'y a qu'un gymnase).
- Un événement i est caractérisé par une date de début di et une date de fin fi . On dit que deux événements sont compatibles si leurs intervalles de temps sont.

Choix d'activités

- Comment trouver « caser » le maximum d'événements ?

- Des idées ?

Choix d'activités

- Stratégie 1 :
  - On trie les événements par **durée**
  - On met les plus courts en premier s'ils sont compatibles avec ceux déjà placés.

Choix d'activités

- Stratégie 1 :
  - On trie les événements par **durée**
  - On met les plus courts en premier s'ils sont compatibles avec ceux déjà placés.

Choix d'activités

- Stratégie 2 :
  - On trie les événements par **date de commencement.**
  - On met les plus courts en premier s'ils sont compatibles avec ceux déjà placés.

Choix d'activités

- Stratégie 2 :
  - On trie les événements par **date de commencement.**
  - On met les plus courts en premier s'ils sont compatibles avec ceux déjà placés.

Choix d'activités

- Stratégie 3 :
  - On trie les événements par **date de fin croissante.**
  - On met les plus courts en premier s'ils sont compatibles avec ceux déjà placés.

Choix d'activités

- Stratégie 3 :
  - On trie les événements par **date de fin croissante.**
  - On met les plus courts en premier s'ils sont compatibles avec ceux déjà placés.
  - C'est optimal, on va le prouver !

Choix d'activités

- Stratégie 3 :
  - On trie les événements par **date de fin croissante**.
  - On met les plus courts en premier s'ils sont compatibles avec ceux déjà placés.
- Soit f1 l'élément finissant le plus tôt. On va montrer qu'il existe une solution optimale contenant cet événement. Soit donc une solution optimale arbitraire O, O = { fi1 , fi2 , ..., fik } avec k le maximum d'événements pouvant avoir lieu dans le gymnase. Il y a deux possibilités : soit fi1 = f1 soit fi1 ≠ f1.
  - Si fi1 = f1 alors on peut les interchanger sans problème
  - Si fi1 ≠ f1 alors on remplace fi1 par f1 . Comme f1 finit avant tout autre événement, alors comme fi2 n'intersectait pas avec fi1 , fi2 n'intersecte pas avec f1 . On peut donc bien trouver une solution optimale ayant comme premier événement l'événement finissant en premier, la classe des solutions ayant f1 en premier "domine".
- Ensuite, après avoir placé f1 , on ne considère que les événements n'intersectant pas avec f1 , et on réitère la procédure sur les événements restants, d'où la preuve par récurrence.

Coloriage de graphes

- Soit G un graphe, on veut colorier les sommets tel que 2 sommets adjacents (voisins) n'ont pas la même couleur. On recherche le nombre minimal de couleur.
- Problème difficile
- Très fréquent en pratique, graphe exprimant des relations entre objets.
- Arête = les objets doivent être gérés différemment
  - Allocation de registre (couleur = registre)
  - Allocation de porte d'embarquement (couleur = porte, nœud = avion, arête = présence simultanée)
  - Allocation de fréquence (arête = interférence entre objet, couleur = fréquence)

Coloriage de graphes

- Stratégie 1 :
  - prendre les sommets dans un ordre au hasard ;
  - **leur attribuer la plus petite valeur possible, i.e. la plus petite valeur qui n'a pas déjà été attribuée à un voisin.**
- Soit K le nombre de couleurs utilisées. Alors $K \leq \Delta(G) + 1$, où $\Delta(G)$ est le degré maximal d'un sommet, le degré d'un sommet étant le nombre d'arêtes auxquelles appartient ce sommet.
  - En effet, au moment où on traite un sommet donné, il a au plus $\Delta(G)$ voisins déjà coloriés, donc l'algorithme glouton n'est jamais forcé d'utiliser plus de $\Delta(G) + 1$ couleurs.
- Pour une clique (un graphe complet, avec toutes les arêtes possibles), il n'est pas possible de faire mieux.

Coloriage de graphes

- Stratégie 2 :
  - trier les sommets par degré décroissant ;
  - **prendre les sommets dans cet ordre et leur attribuer la plus petite valeur possible.**
- Soit n = | V | le nombre de sommets, et di le degré du sommet vi . On montre alors que $K \leq \max_{1 \leq i \leq n} (\min(di + 1, i))$.
  - En effet, au moment où on colorie le i-ème sommet vi , il a au plus min(di, i – 1) voisins qui ont déjà été coloriés, et donc sa propre couleur est au plus 1 + min(di, i – 1) = min(di + 1, i). En prenant le maximum sur i de ces valeurs, on obtient la borne demandée.
- Cette borne suggère de se débarrasser d'abord des plus gros degrés, qui disparaissent ainsi de la complexité tant que min(di +1, i) = i. Comme on ne sait pas a priori jusqu'à quand privilégier les grands degrés, on a trié l'ensemble des sommets par degrés décroissants dans l'algorithme glouton.

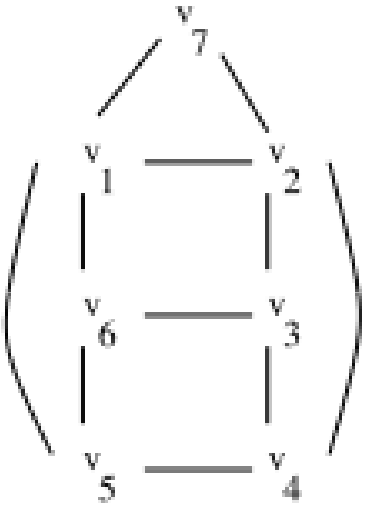
Coloriage de graphes

- L'idée intuitive est de colorier en priorité les sommets ayant beaucoup de sommets déjà coloriés.
  - On définit le degré-couleur d'un sommet comme son nombre de voisins déjà coloriés.
  - Le degré-couleur, qui va évoluer au cours de l'algorithme, est initialisé à 0 pour tout sommet :
- Stratégie 3 (**algorithme de Brelaz: DSATUR**) :
  - **prendre parmi les sommets de degré-couleur maximal un sommet de degré maximal, et lui attribuer la plus petite valeur possible.**

Stratégie de Brelaz

- Cette stratégie est assez générale : on prend l'objet a qui il reste le moins de liberté

Stratégie de Brelaz



1) On prend au départ un sommet de degré maximal, par exemple v1 , et on lui donne la couleur 1.

2) Le degré-couleur de v2 , v5 , v6 et v7 passe à 1, on choisit v2 qui est de degré maximal parmi ces trois sommets, et on lui donne la couleur 2.

3) Maintenant, v7 est le seul sommet de degré-couleur 2, on le choisit et on lui attribue la couleur 3.

4) Tous les sommets restants ont le même degré-couleur 1 et le même degré 3, on choisit v3 au hasard, on lui donne la couleur 1.

5) Puis v4 , de degré-couleur 2, reçoit la couleur 3.

6) Enfin v5 reçoit la couleur 2 et v6 la couleur 3.
Le graphe est 3-colorié, c'est optimal.

Couverture de sommets

- Un ensemble **transverval** (ou **couverture de sommets**) est un ensemble de sommets tel que chaque arête a une de ses extrémités dans cet ensemble.

- La recherche d'un ensemble **transversal minimum** (**minimum vertex cover problem**) est le problème d'optimisation :
  - DONNEES : Graphe G
  - QUESTION : Le plus petit nombre k tel que G a un transversal de taille k
- Problème de décision correspondant (**vertex cover problem**) :
  - DONNEES : Graphe G et un entier positif k
  - QUESTION : Est-ce que G contient un transversal de taille k ?

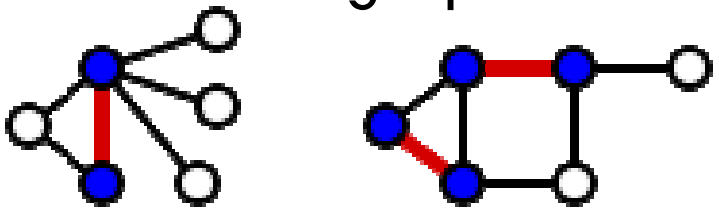
Couverture de sommets

- Si un ensemble de sommets S est un transversal, son complément est un stable (ou *ensemble indépendant*).
- Un graphe à *n* sommets a un transversal de taille *k* si et seulement s'il a un stable de taille *n-k*.

- Hitting set = vertex cover dans un hypergraphe
- Problème très fréquent : on veut un représentant de chaque groupe et minimiser le nombre de représentants

Stratégie gloutonne

- Il existe un algorithme qui trouve une 2-approximation du problème (la solution est au mieux 2 fois plus petite). On répète les 2 opérations suivantes tant qu'il reste des arêtes:
  - On prend une arête du graphe, on met ses deux extrémités dans l'ensemble transversal
  - On supprime ces deux sommets du graphe



En bleu le transversal

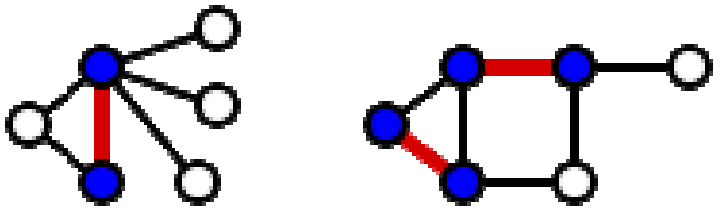
Stratégie gloutonne

- On peut montrer que l'on fait au plus une erreur d'un facteur 2.
- Montrons que le glouton fonctionne.
  - Il est clair qu'il ne reste pas d'arête à la fin donc toutes les arêtes sont bien couvertes.
- Montrons que l'on a au plus un facteur 2
  - Quand on sélectionne une arête, l'une des ses extrémités au moins doit appartenir au transversal, sinon l'arête ne sera pas couverte. C'est vrai pour tous les transversaux, y compris l'optimal. Donc, on doit prendre au moins 1 des deux extrémités des arêtes que l'on a sélectionné. Le transveral a donc au plus une taille deux fois inférieure à nombre d'arêtes prise puisque le glouton prend les deux extrémités de chaque arête.

Autre stratégie

- Heuristique des plus hauts degrés.
  - forme une solution réalisable en sélectionnant à chaque itération le sommet couvrant un maximum de sommets.

- En pratique elle donne souvent de meilleurs résultats



Autre stratégie

- Cette heuristique peut fournir des solutions aussi mauvaises que l'on veut, dans le sens que pour tout $\rho > 1$ on peut construire une instance pour laquelle l'heuristique donne une solution dont la valeur est supérieure à ρ fois celle de l'optimum.
- En effet cette stratégie peut être très mauvaise si le graphe est biparti (le problème est polynomial dans ce cas)