Résolution de Problèmes
Introduction

Marie Pelleau
marie.pelleau@unice.fr

Master 1 - Semestre 1

Remerciements

- Wikipedia
- Jean-Charles Régin
- Olivier Bournez, LIX
- Christine Solnon, Université Lyon
- Anne Benoit, ENS Lyon
- Roman Barták, Charles University

Plan du cours

Cours magistraux

1. Algorithmes Gloutons
2. Recherche Locale
3. Programmation Par Contraintes

Contrôle des connaissances

- Contrôle continu
- Contrôle terminal

Références

- T. Cormen, C. Leiserson, R. Rivest, **Introduction à l'algorithmique**, Dunod
- D. Knuth, **The Art of Computer Programming**
- M. Gondran et M. Minoux, **Graphes et Algorithmes**
- Autres livres selon votre goût : ne pas hésiter à en consulter plusieurs

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

Problème

- Certains problèmes sont faciles : Trier des nombres, Inverser une chaîne
- D'autres sont difficiles : le TSP

Traveling Salesman Problem (TSP)

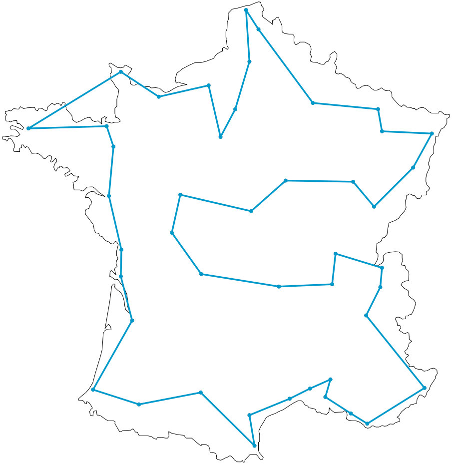
Description

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

Formulation mathématique

Étant donné un graphe complet pondéré, trouver un cycle hamiltonien de poids minimum

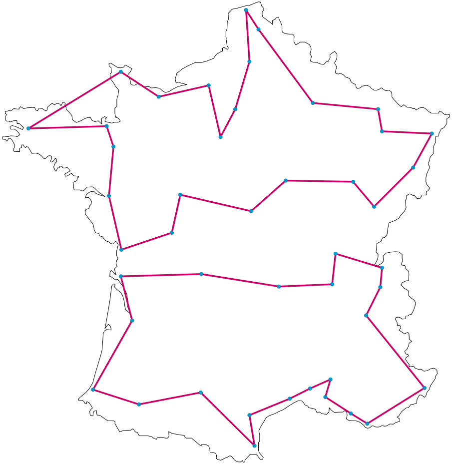
TSP



Description

- Chaque ville est visitée une et une seule fois

TSP



Description

- Chaque ville est visitée une et une seule fois
- Un seul tour (pas de sous-tour)

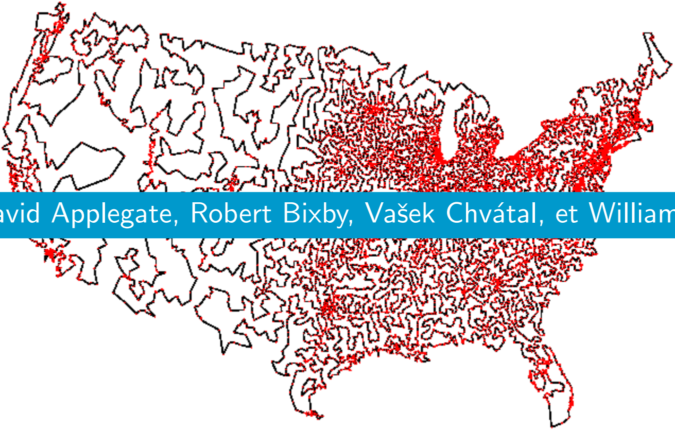
TSP

Description

- Certains problèmes sont équivalent au problème du TSP
  - problème d'ordonnancement : trouver l'ordre dans lequel on doit construire des objets
- La version "pure" du TSP n'est pas fréquente en pratique, on a souvent des problèmes qui sont
  - Non euclidiens
  - Asymétriques
- Ces variations ne rendent pas le problème plus facile
- Problème assez commun
  - Vehicle routing (time windows, pickup and delivery...)

TSP

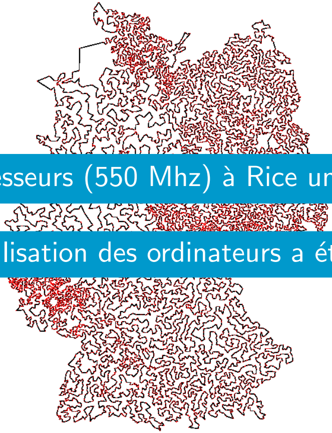
États-Unis 13 509 villes, résolu en 1998



Par David Applegate, Robert Bixby, Vašek Chvátal, et William Cook

TSP

Allemagne 15 112 villes, résolu en 2001



Réseau de 110 processeurs (550 Mhz) à Rice university et Princeton

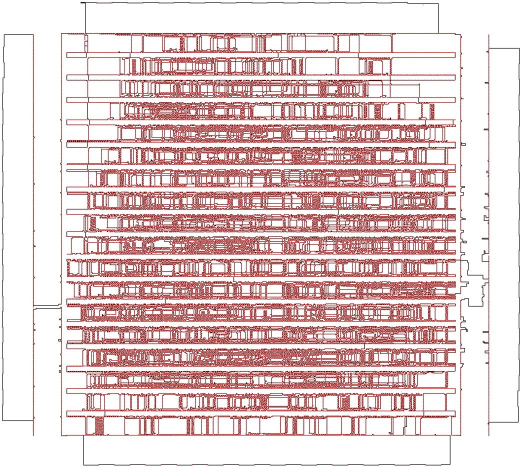
Temps total d'utilisation des ordinateurs a été de 22.6 années

TSP

Suède 24 978 villes, résolu en 2004

TSP

Puce informatique 85 900 “villes” résolu en 2006



TSP

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

- 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, ...)

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)

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
- À 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, $O(n\log(n))$ linéarithmique

NP-Complétude

LE problème majeur de l'informatique actuelle

P vs NP

- 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, on ne sait pas s'il existe un algorithme rapide.
On connaît des algorithmes exponentiels en temps : 2^n

21 problèmes NP-complets de Karp

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

Remarque

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

Hitting-set : Recouvrement (set cover)

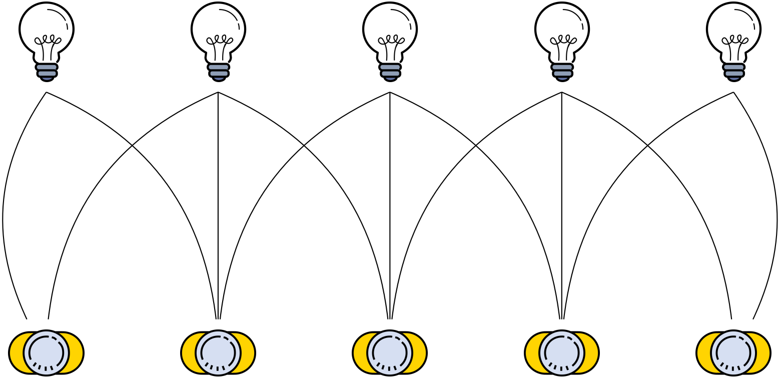
Description

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

Hitting-set : Recouvrement (set cover)

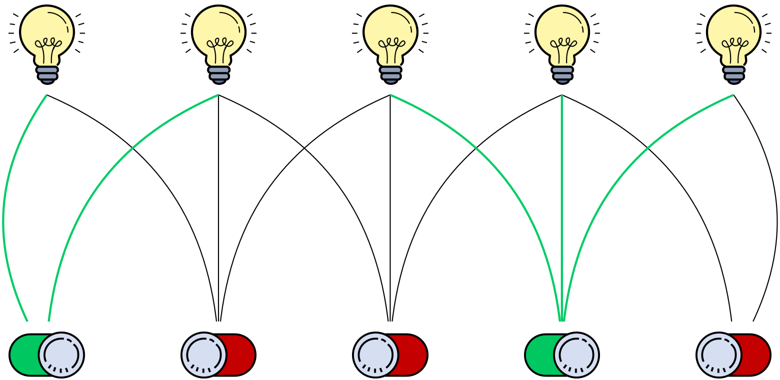
Exemple



n interrupteurs, 2 choix par interrupteur $\Rightarrow 2^n$

Hitting-set : Recouvrement (set cover)

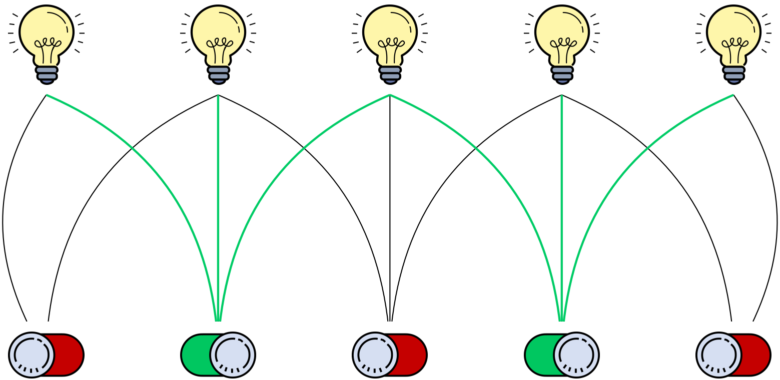
Exemple



n interrupteurs, 2 choix par interrupteur $\Rightarrow 2^n$

Hitting-set : Recouvrement (set cover)

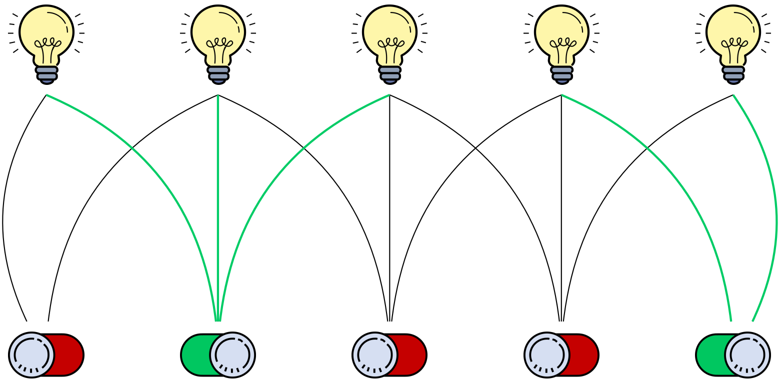
Exemple



n interrupteurs, 2 choix par interrupteur $\Rightarrow 2^n$

Hitting-set : Recouvrement (set cover)

Exemple



n interrupteurs, 2 choix par interrupteur $\Rightarrow 2^n$

Coloriage des sommets d'un graphe

Description

On colorie les sommets d'un graphe de façon à ce que 2 sommets reliés par une arête n'aient pas la même couleur

Principe de reliement-contraction

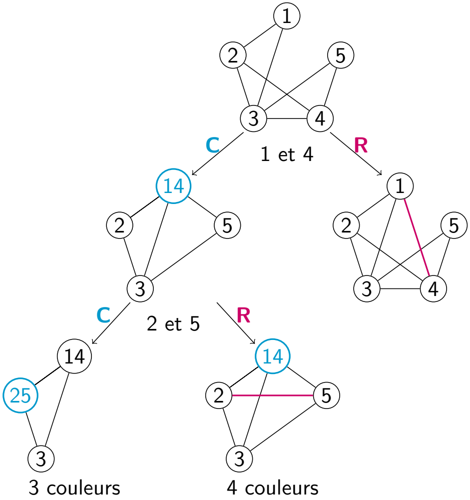
Pour colorier un graphe complet (une clique) contenant n sommets il faut n couleurs

- On prend 2 sommets a et b non reliés
- Soit ils ont la même couleur $\Rightarrow$ Contraction
- Soit ils ont une couleur différente $\Rightarrow$ Reliement
- On arrive à une clique $\Rightarrow$ le nombre de sommets donne le nombre de couleurs

p arêtes potentielles, 2 choix par arêtes (même couleur ou couleurs différente) $\Rightarrow 2^p$

Coloriage des sommets d'un graphe

Exemple



Facile vs Difficile

- 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

La différence peut être subtile

- Problème pur : facile
- On introduit la connexité : difficile
- On introduit la convexité : difficile
- On introduit la connexité et la convexité : facile

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**

Exemple

- Étant 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 ?
- 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 une fonction objectif (min ou max)
- Une solution **optimale** est une solution faisable, telle que son coût est le minimum (resp. maximum) de toutes les solutions faisables
- Si le but est de minimiser alors c'est un problème de minimisation, sinon c'est un problème de maximisation
- En changeant le signe de la fonction il est possible de passer d'un problème de minimisation à un problème de maximisation

Exemple

- Plus court chemin passant par toutes les villes ?
- Le nombre minimum de couleurs pour colorier un graphe ?

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 être lent ou rapide
- L'un peut être rapide et pas l'autre

Optimisation et Décision

À chaque problème d'optimisation correspond un problème de décision qui demande s'il existe une solution ayant une valeur particulière

Exemple

On trouve un plus court chemin entre s et t dont la valeur est c

- Problème de décision : Existe-t-il un chemin de coût c ?
- Preuve d'optimalité : Existe-t-il un chemin de de coût $< c$?

Optimisation et Décision

On résout souvent les problèmes d'optimisation en résolvant une suite de problèmes de décision

- On cherche une solution faisable, elle a un coût k
- On pose cherche s'il existe une solution de coût $< k$ et on répète le processus
- À la fin, on prouve bien l'optimalité, car la dernière résolution ne trouve pas de solution

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

Dans ce cours : on va voir comment proposer des solutions à ces problèmes

- avec des heuristiques (inexact mais rapide)
- avec de l'énumération complète des combinaisons (exact mais lent)