# Algorithmique
# M2 Data Science

## cours 5
## Algorithmes non-exacts

Vincent Runge


1. greedy algorithms (algorithmes gloutons)
2. Heuristic algorithms (algorithmes heuristiques)
3. Approximation algorithms (algorithmes d'approximation)
4. TP

# Algorithmes non-exacts

On connaît (avec ce cours) différentes stratégies algorithmiques **exactes** :

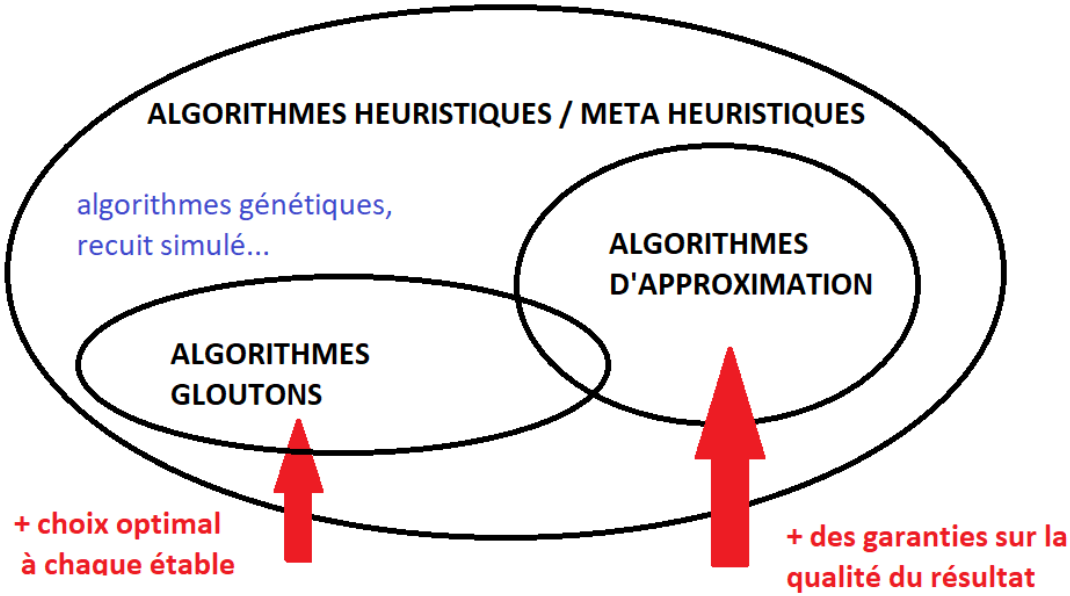
1) Récursivité (découper en sous-problèmes équivalents)
2) Programmation dynamique (résoudre des sous-problèmes emboîtés)
3) Algorithmes naïfs (donner une solution immédiate triviale)

**Pourquoi s'intéresser à des algorithmes non-exacts?**

- Pour les problèmes NP : si la taille du problème est trop importante, c'est l'unique solution de repli possible
- Pour des questions de temps : on souhaite obtenir rapidement ou en "temps réel" une bonne solution
- Parfois, obtenir la meilleure solution possible n'est pas l'objectif

# greedy / heuristic / approximation algorithms

- Les algorithmes **greedy / heuristic / approximation** ne garantissent pas de retourner la meilleure solution.
- Ils permettent de résoudre “convenablement” des problèmes de trop grande complexité (typiquement NP difficile)
- Les trois termes sont presque synonymes



Section 1

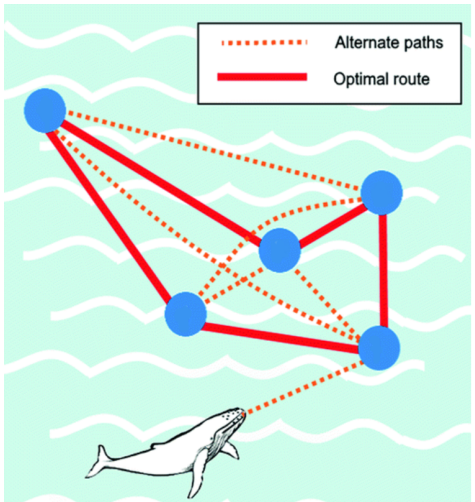
# greedy algorithms (algorithmes gloutons)

# Algorithme glouton : définition

- Faire un choix optimal à chaque sous-étape (dans l'espoir de réussir à construire une solution globablement optimale!)
- **Ne garantit pas** de trouver une solution optimale (exception : structure de Matroïde)
- Remplace une solution en programmation dynamique trop lente

# Exemple 1 : Problème du voyageur de commerce (again!)

- À chaque étape, on fait le choix (localement optimal) d'aller au point le plus proche. (The nearest neighbour (NN) algorithm)
- Mais, pour de nombreuses configurations de villes, cet algorithme retourne la pire route!




Un article scientifique pour les curieux

# Exemple 2 : Problème du rendu de monnaie (NP ou glouton!) (#1)

lien wikipédia

- Soit un système de monnaie (pièces et billets) donné
- Comment rendre une somme donnée de façon optimale, i.e. avec **un nombre minimal de pièces et billets**?
- Cas des systèmes canoniques $\rightarrow$ l'algorithme glouton est optimal


- système (1,3,4) : $6 = 4+1+1$ (algorithme glouton)

$6 = 3+3$ (programmation dynamique)

Section 2

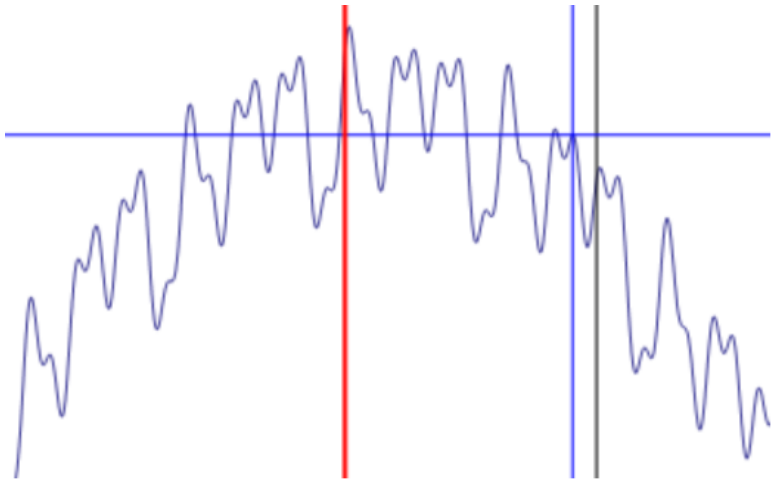
# Heuristic algorithms (algorithmes heuristiques)

# Algorithme heuristique : définition

**Classe d'algorithme qui engloble les algorithmes gloutons!**

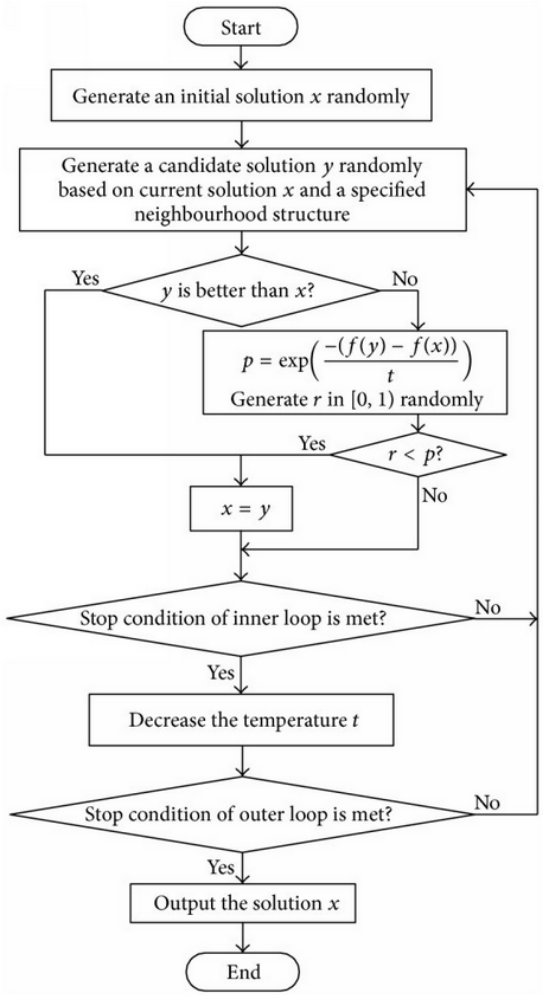
- On utilise souvent ce terme lorsqu'un **aspect aléatoire** est présent (2 exécutions de l'algo sur les mêmes données peuvent donner 2 résultats différents)
- Algorithmes **méta-heuristiques** = permettent de trouver/ générer / sélectionner une heuristique

# Recuit simulé (simulated annealing)



https://en.wikipedia.org/wiki/File:
Hill_Climbing_with_Simulated_Annealing.gif

# Recuit simulé (simulated annealing)




(source du graphe)

# Algorithme génétique

Quand les utiliser?

- Evaluation rapide de la fonction objectif
- Espace de recherche de grande dimension! ($R^p$ ou problème NP)
- Pas d'algorithme déterministe efficace

# Algorithme génétique (principe)

- vocabulaire emprunté à la biologie

Ses ingrédients :

1) Une fonction à optimiser $f$
2) Une population de chromosomes (qui évalue cette fonction). Un chromosome $= (x_1, ..., x_d) \in \{0, 1\}^d$ (classiquement) ou $\mathbb{R}^d$ ou $\{1, ..., D\}^d$
3) Une sélection de chromosomes pour reproduction
4) Des croisements
5) Des mutations aléatoires

# Algorithme génétique (principe) #2

Une itération :

1) Population $(x_1^1, ..., x_d^1), \ldots, (x_1^q, ..., x_d^q)$ évaluée par $f$ et classée de 1 à $q$ (meilleure solution à moins bonne)
2) On conserve $q'$ chromosomes (les $q'$ meilleurs). Souvent $q' = q/2$.
3) On tire $q''$ parmis ceux-ci pour croisement (tirage sans remise et probabilité plus grande si meilleure évaluation par $f$), puis le croisement est effectué par paires.
4) Certaines valeurs sont modifiées par mutation, i.e. modification aléatoire d'une valeur d'un chromosome par probabilité très faible ($p = 0.001$ par exemple)

# Algorithme génétique (en dessin)

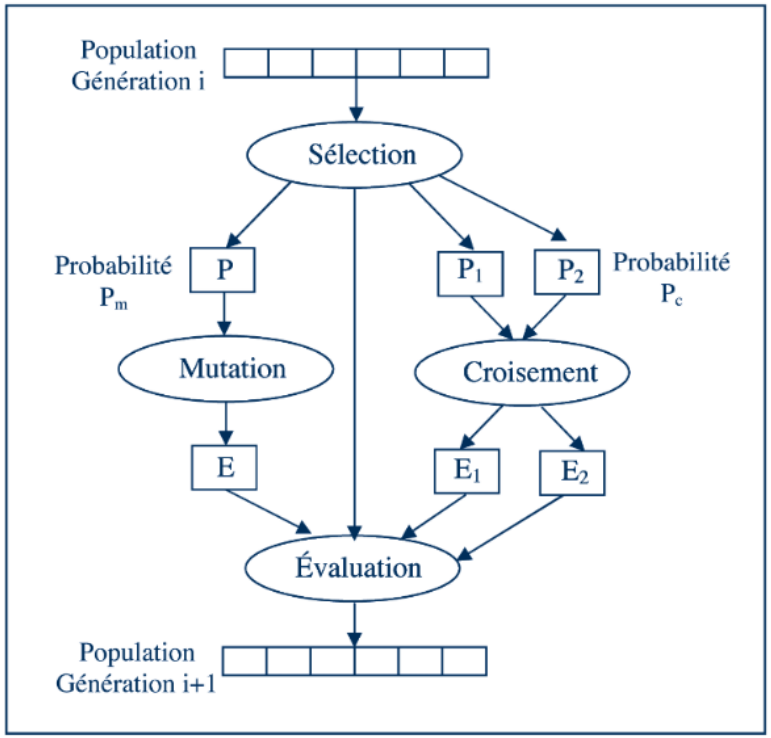



*Figure 1. Principe général d'un algorithme génétique.*

Liste d'applications

# Algorithme génétique + Data Science

Utilisé en sélection de modèle (chromosome = un vecteur binaire)

Exemple : TPOT is a Python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming.

Section 3

# Approximation algorithms (algorithmes d'approximation)

# Algorithme d'approximation : définition

- Pour les problèmes d'optimisation de complexité NP
- Donne une **solution approchée** avec une borne sup sur la distance de cette solution à la solution optimale
- Permet de distinguer différents niveaux de difficulté parmi les problèmes NP

**Solution optimale $OPT$, solution $\rho$-approchée $f(x)$ si:**

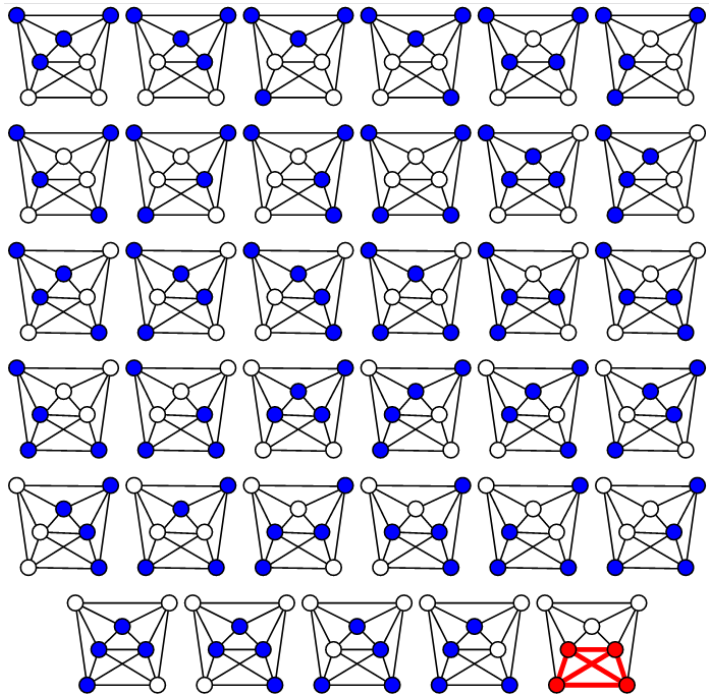
| | | |
|---|---|---|
| $OPT \leq f(x) \leq \rho \times OPT$ | problème de minimisation | $\rho > 1$ |
| $\rho \times OPT \leq f(x) \leq OPT$ | problème de maximisation | $\rho < 1$ |

Algorithme de Christofides pour le problème du voyageur de commerce facteur d'approximation de $\rho = 3/2$.

# PTAS or not PTAS

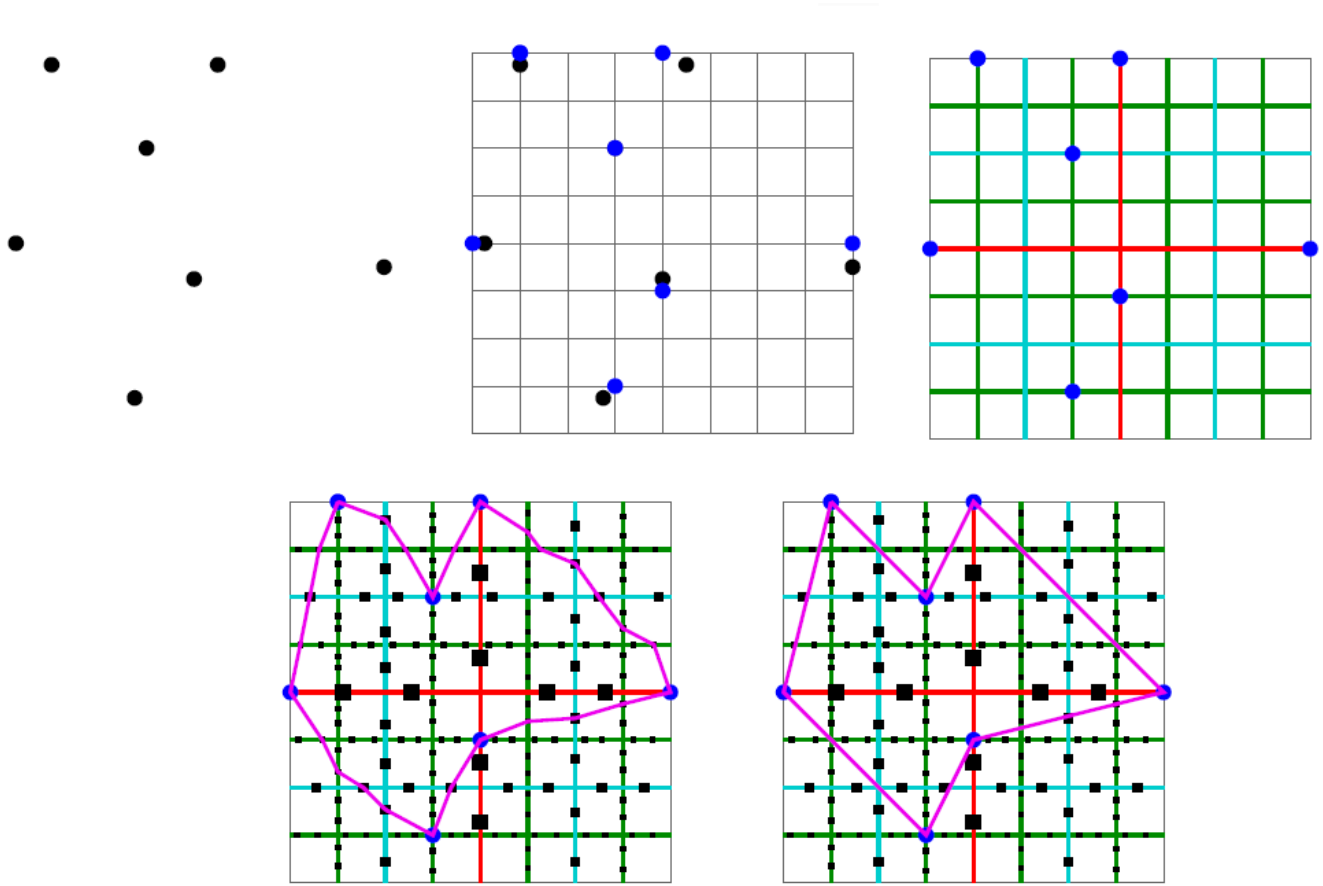
*polynomial-time approximation scheme (PTAS)* = il existe un algorithme polynomial $A_\epsilon$ en la taille $n$ du problème qui trouve le minimum à $1+\epsilon$ près pour tout $\epsilon > 0$

1) Notre problème du voyageur de commerce est PTAS (cas euclidien)
2) Le problème de la clique maximale n'est pas PTAS

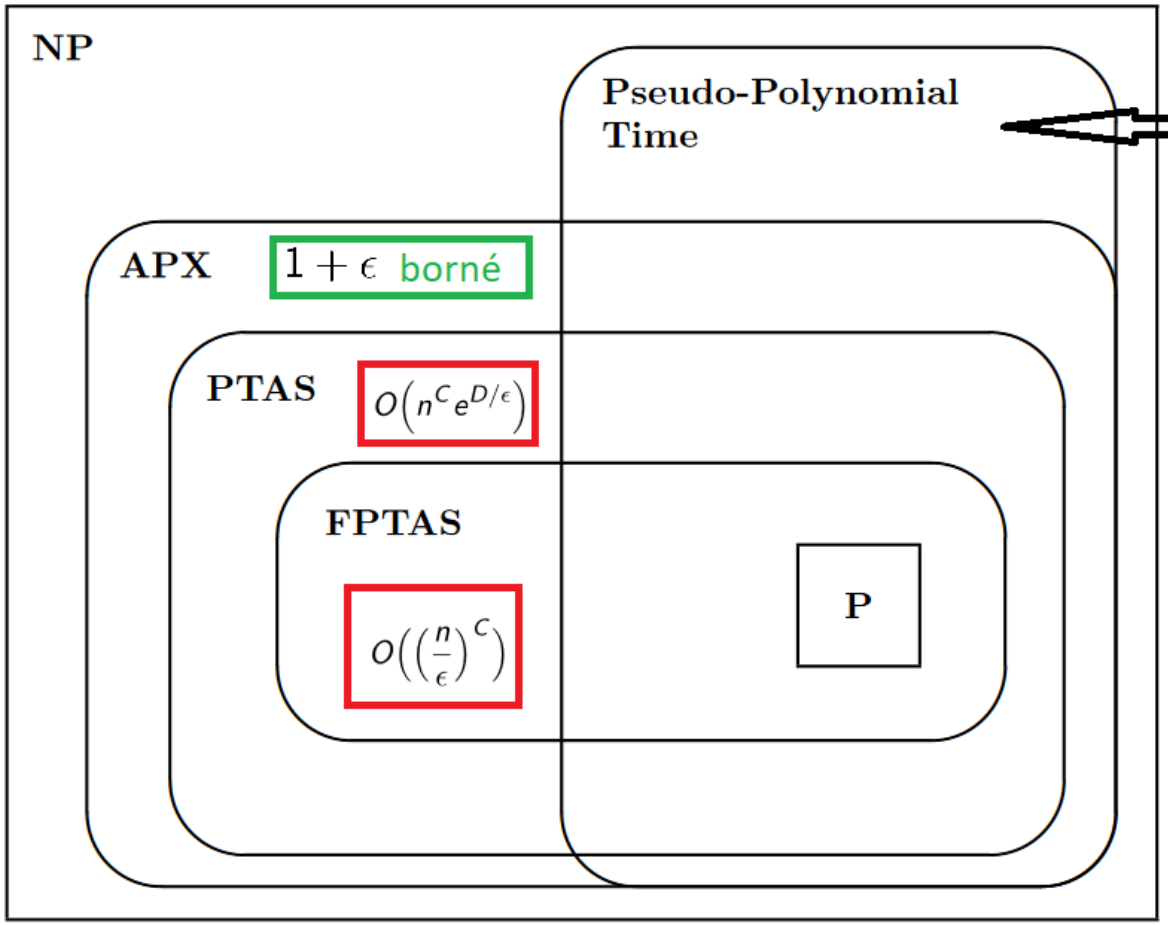


# PTAS or not PTAS

85 slides d'explication (stanford)



# PTAS or not PTAS




polynomial in the numeric value of the input (the largest integer present in the input) but not necessarily in the length of the input

Section 4

# TP

# Voyageur de commerce : Compétition !

1) Implémenter et tester l'algorithme génétique pour le voyageur de commerce. Vous pourrez prendre les fonctions codées ici pour trouver le meilleur chemin

**Objectifs:**

- tester la convergence (la qualité du résultat) pour différents paramètres d'initialisation.
- comment les paramètres "optimaux" évolue-t-il avec la taille des données?
- Le choix des paramètres initiaux est-il sensible à la position des villes?

# Voyageur de commerce : Compétition ! #2

2) Par groupes de 3 (les groupes des projets!) m'envoyer votre version de la fonction à tester sur les données ici (j'ai ferai un package test)
3) La compétition :

- Evaluer sur 20/30/40 villes (répartition uniforme dans $[0,1]^2$) avec $10^4/10^5/10^6$ chromosomes autorisés

Par exemple $10^4 = 100$ iterations avec initialisation de 10 chromosomes.

(on augmentera la limite de chromosomes selon les résultats préliminaires)

# Problème du sac à dos

Terminer la résolution du problème du sac à dos

**Objectifs:**

- Retrouver par des simulations la complexité quadratique recherchée
- Montrer que la solution exhaustive est rapidement impossible