Introduction à l'optimisation discrète

Laurent Lemarchand
LabSTICC/UBO
Laurent.Lemarchand@univ-brest.fr

Optimisation combinatoire

- **De nombreux choix**
  - Pas évident, nombreux aspects à prendre en compte ...

Problème : Comment trouver la meilleure **solution** par rapport à un **critère** donné ?

- On peut lister toutes les solutions (problème discret)

Optimisation combinatoire
introduction

- POC (problème d'optimisation combinatoire) : *prouver la meilleure solution d'un prblème donné, parmi l'ensemble (fini) des solutions réalisables du problème*

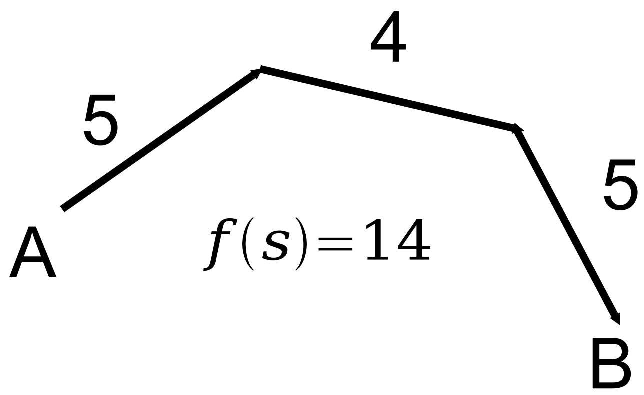
  - *S :* espace des solutions

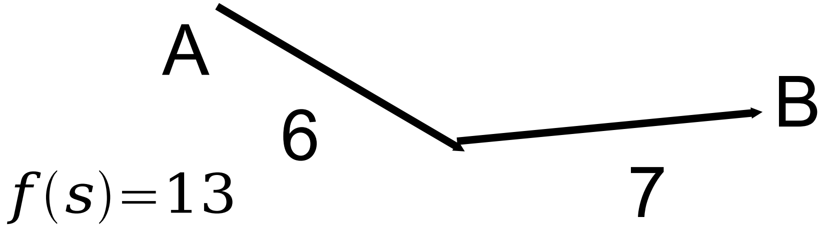
  - *f(S) : fonction objective* pour l'évaluation de la qualité des solutions – à maximiser ou minimiser

Aie, une seule place assise !

Optimisation combinatoire
exemples

- Recherche de chemins: *plus court chemin entre deux sommets d'un graphe*
  - *S :* tous les chemins possibles
  - *f(S) : longueur des chemins* (à minimiser !)



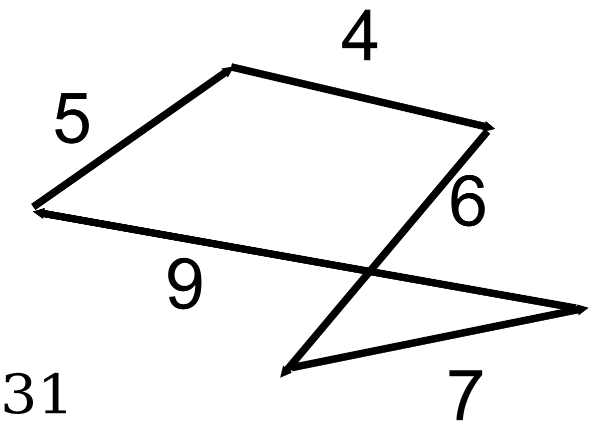


- Optimisation dans les graphes

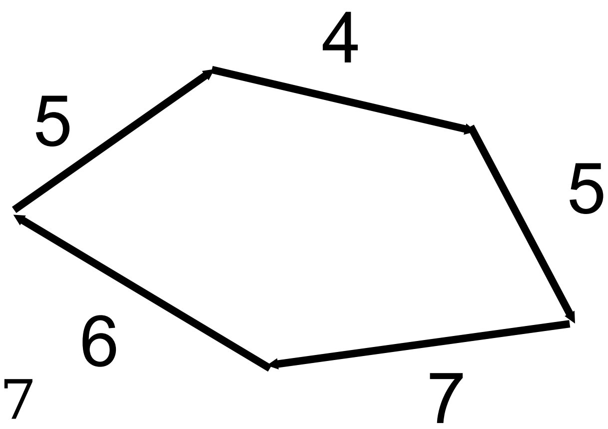
Optimisation combinatoire
exemples

- Voyageur de commerce : *visiter chaque ville une et une seule fois et revenir à son point de départ*
  - *S : toutes les tournées possibles* (tours)
  - *f(S) : longueur des tours* (à minimiser !)

V Rodin et. all



$f(s) = 31$



$f(s) = 27$

Optimisation combinatoire
exemples

- Un meunuisier peut fabriquer au plus 6 chaises et 3 tables par journée de 8 heures maximum
  - Une table lui rapporte 90 euros (1h15 de travail)
  - Une chaise, 50 euros (45mn de travail)
- Maximiser son profit

$$f(s) = 90\mathrm{t} + 50\mathrm{c}$$
$$75\mathrm{t} + 45\mathrm{c} \leqslant 480$$
$$0 \leqslant t \leqslant 3$$
$$0 \leqslant c \leqslant 6$$

- *Programmation linéaire :* simplexe en $O(2^n)$

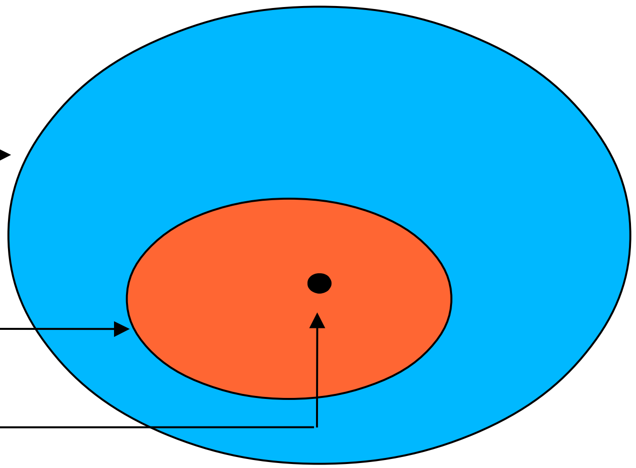
Optimisation combinatoire
cadre général

- Espace de solutions $S \subseteq X$

- Fonction objective $f : X \rightarrow \mathbb{R}$

- Trouver $s^* \in S$ t.q $\forall s \in S \;\; f(s^*) \leq f(s)$

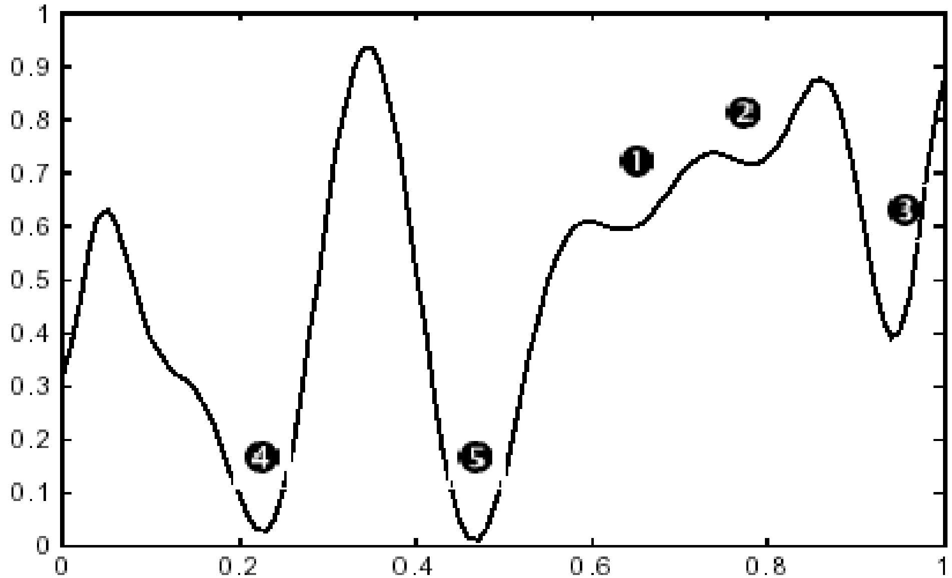
- X , espace des solutions
- S , espace des solutions **réalisables**
- s* , solution optimale



Optimisation combinatoire problème 1: solutions sous-optimales

- On peut tomber dans un minimum local 1, 2, 3, 4, 5 (5 est la meilleure solution :-)
- On doit explorer tout l'espace des solutions
- Pas simplement le voisinage
- Exemple: minimiser une var continue sur un interval donné

f(x)

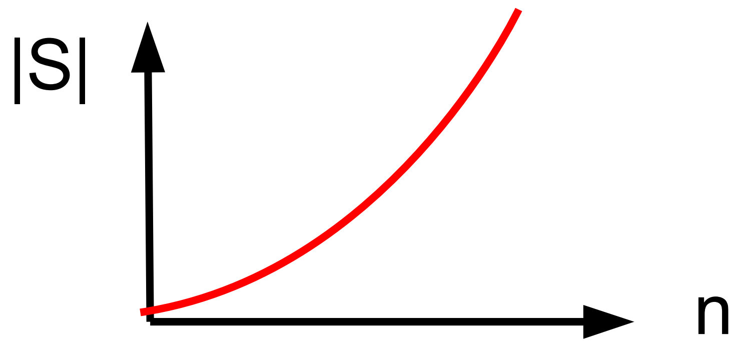


V Rodin et. all

Minimiser $f(x) : X \rightarrow [0..1]$

Optimisation combinatoire
problème 2 : explosion combinatoire

- Taille de S par rapport à la taille du problème
  - Voyageur de commerce (n-1)! / 2
  - Bi partitionnement 2^n
  - Programme en variable bivalentes 2^n



- La taille de S est exponentielle

Optimisation combinatoire
explosion combinatoire

- Enumérer toutes les solutions : souvent impossible

Complexité	*N = 1*	*N = 10*	*N = 100*	*N = 1000*	*N = 10000*
log N	0 ms	1 ms	2 ms	3 ms	4 ms
N	1 ms	10 ms	0.1 s	1 s	10 s
N^2	1 ms	0.1 s	10 s	17 min	28 hours
N^3	1 ms	1 s	17 min	12 days	32 years
e^N	3 ms	22 s	9 10^{32} years !	Long time	Very long time

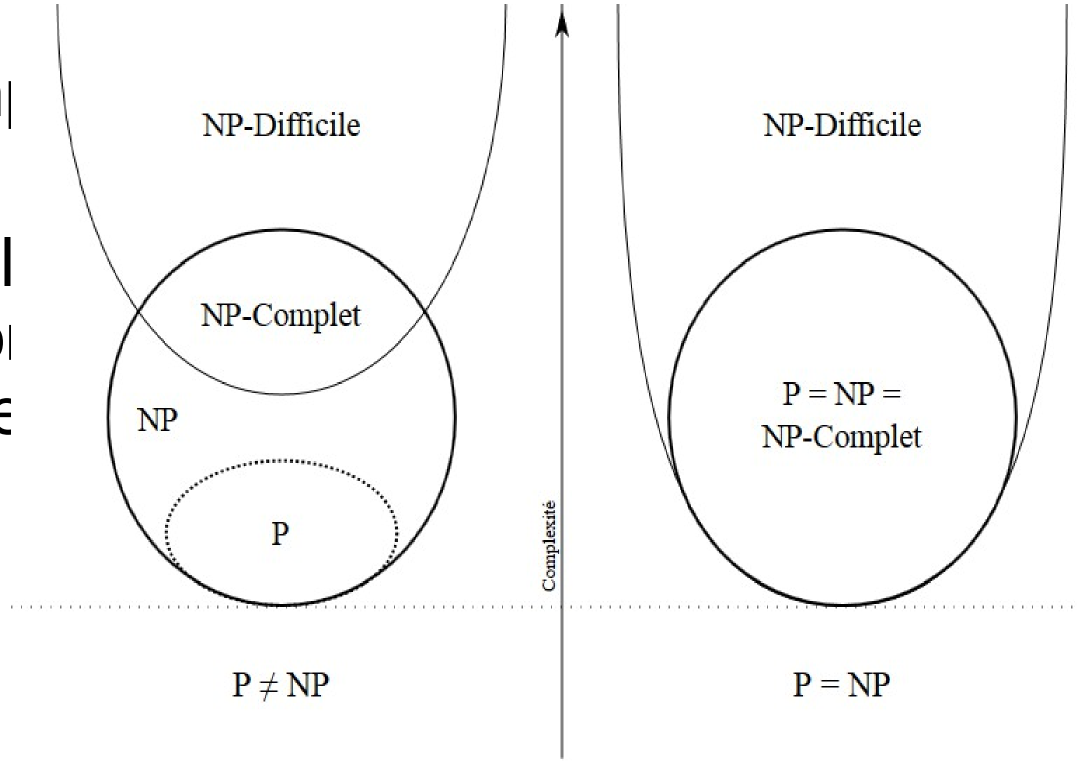
Si 1000 solutions evaluées par sec

V Rodin et. all

Classes de problèmes
Complexité

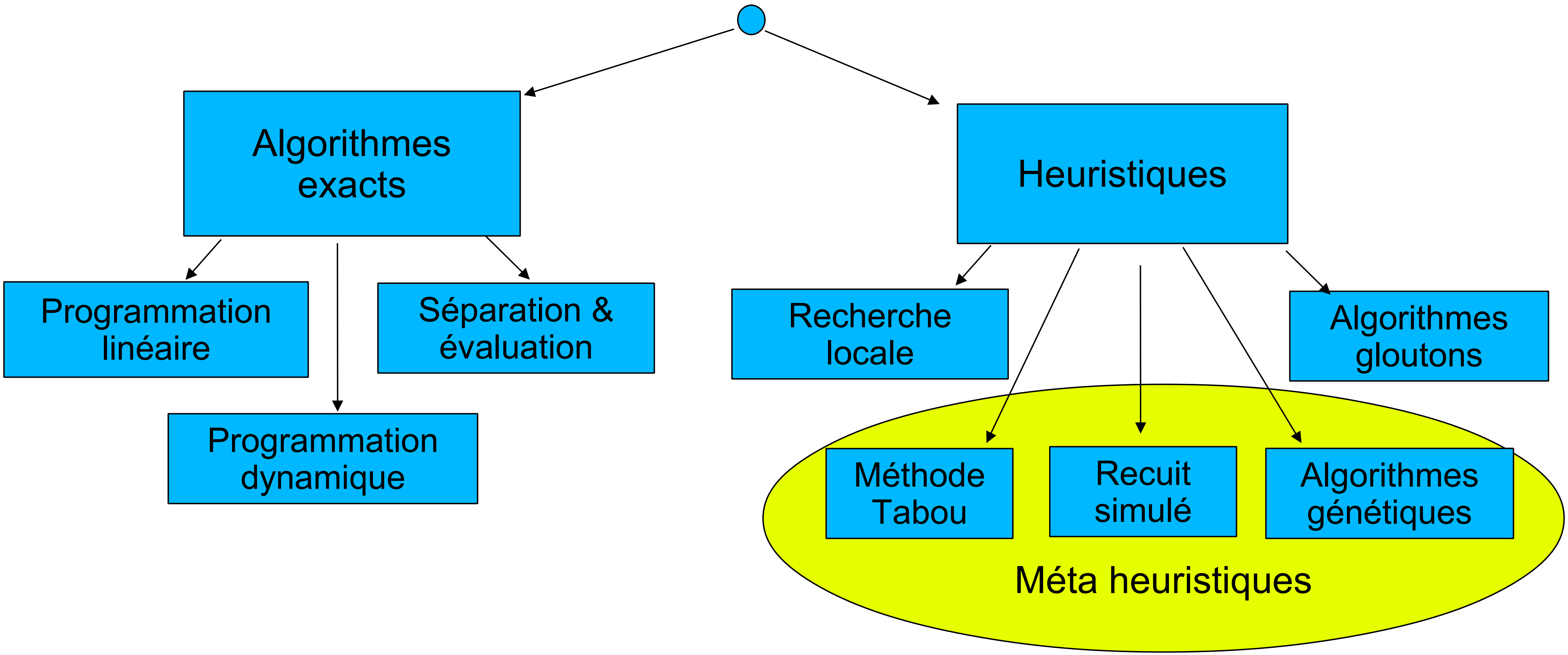
Problème de **décision** p :

- p ∈ classe P : algorithmes polynomiaux pour résoudre p en tem polynomial
- p ∈ classe NP : pas d'algo polynomial connu, mais vérification d'une solutio possible en temps polynomial. [est-ce que P = NP ?]
- Si tous les problèmes de NP peuvent être ramenés à p par une transformation en temps polynomial, alors
  - Si p ∈ NP, p ∈ classe NP-complet (⟶ Les pbs les plus durs de NP)
  - Si p ∉ NP, p ∈ classe NP-difficile



- LP ∈ P
- MILP ∈ NP-hard

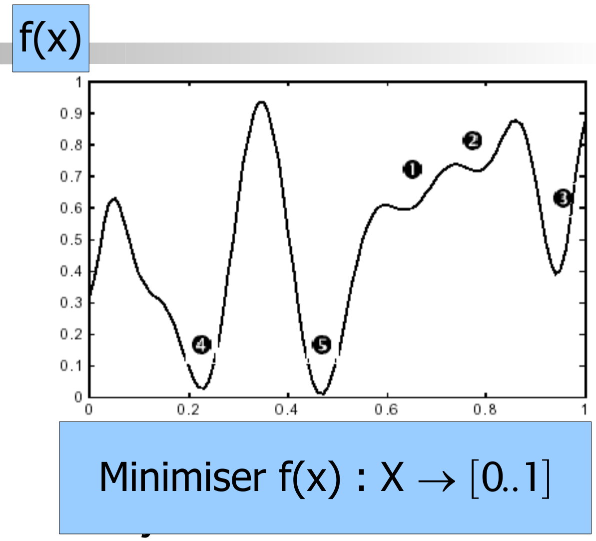
Optimisation combinatoire
algorithmes de recherche



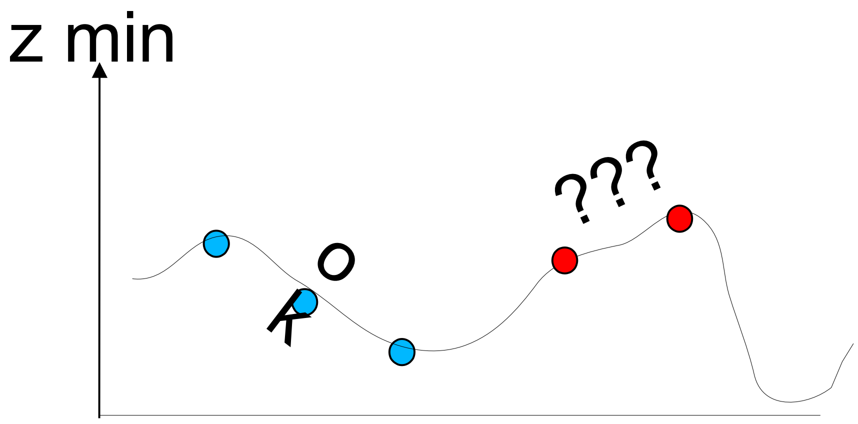
Combinatorial Optimisation combinatoire
techniques locales vs. techniques globales

- Rappel pb1 : minimum local
- Choix entre :
  - Améliorer/Affiner la solution courante
  - Explorer l'espace de recher complet

Exploitation ⟷ Exploration



- Politiques de compromis



Optimisation combinatoire
méthodes exactes vs. méthodes approchées

- En pratique :
  - *Pas toujours besoin de la meilleure* solution
  - *Mais* obtenir une *bonne* solution et éventuallement une garantie sur sa qualité (borne sur la différence avec la solution optimale)

- Si la solution exacte est voule, une méthode exacte est requise (parfois non implentable ou trop consommatrice de temps)
- Si solution appproximative
  - Heuristiques
    (découverte basée en partie sur de l'aléatoire)
  - Meta-heuristiques
    (algorithmes généraux à spécialiser pour un problème donné)

V Rodin et. all

Problèmes de grande taille
les heuristiques sont très utiles !

- Ok, résultat approché, mais :
  - Parfois, seule méthodes disponibles (*ex.* program optimization)
  - Ou méthodes exactes sur modèle approché (*ex.* test de circuits)
- Utilité
  - Gérer l'explosion combinatoire
  - Objectifs multiples, objectifs flous
  - Variabilité des entrées (robustesse des solutions)
  - Si temps de calcul plus important que précision de la solution

Problèmes de grande taille
Parallélisme

- Problèmes trop grands, besoin de réponses rapides
- Disponibilité de ressources en calcul parallèle (multicore, NOW)

- Parallelisation possible
  - Partitionnement de l'espace de recherche : anomalies favorables ou défavorables selon la stratégie de recherche et la fct obj.
  - Implementation centralisée ou distribuée
  - Pb de la mise à jour de la borne