Métaheuristiques pour l'optimisation combinatoire

résolution de problèmes

***

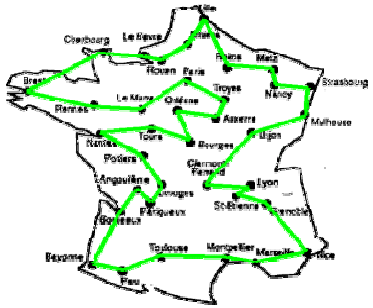
Grégory Thibault
UFR informatique
Université Lyon 1

Plan du cours

- Quelques problèmes classiques
- Quelques algorithmes classiques
  - un peu de vocabulaire
  - codage des solutions
  - taxinomie
  - méthodes complètes
  - méthodes incomplètes
    - recherche par voisinage
    - algorithmes à population
  - intensification versus diversification
  - design des méthodes
- notion de landscape
- transition de phase
- « no-free-lunch »

Quelques problèmes classiques (1/3)

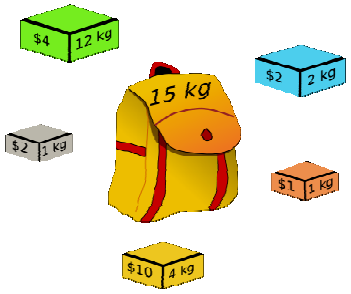
- Traveler Salesman's Problem
  - cycle hamiltonien de poids minimum dans un graph complet
  - nombre de solutions pour n « villes » : (n-1)! / 2



n = 30 :
+ de 4 x 10^{30} solutions !

Quelques problèmes classiques (2/3)

- KnapSack
  - optimisation de contraintes potentiellement contradictoires
  - minimiser le poids, maximiser le gain



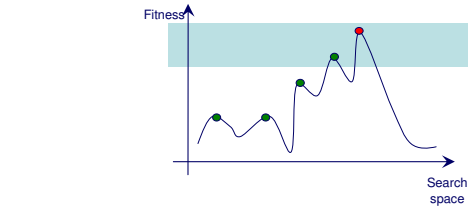
Problème NP-complet !

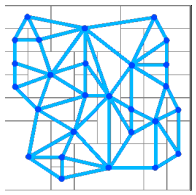
Quelques problèmes classiques (3/3)

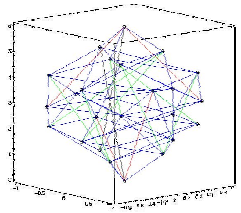
- Jeu d'échec,
- Jeu de GO (extrêmement difficile),
- Problèmes de flux,
- Emploi du temps (agencement),
- Coloration minimale propre de graphe,
- Rendu de monnaie,
- de nombreux problèmes de la vie courante…

Quelques algorithmes classiques (1/14)

- Un peu de vocabulaire :
  - mimétisme
  - heuristique
  - méta heuristique
  - minimum local / global
  - méthodes (in)complètes
  - codage solution
  - landscape
  - structure de voisinage
  - population / individu
  - instance / solution
  - modèle
  - etc.

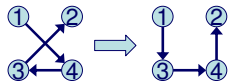






Quelques algorithmes classiques (2/14)

- Codage d'une solution
  - codage efficace = voisinage facilement calculable
    - e.g. TSP : suite de villes pour voisinage 2-opt
    - e.g. TSP : codage binaire pour algos génétiques



- vecteur binaire

0	1	1	1	0	1	0	0	1	0	1	1	0	1	1	...

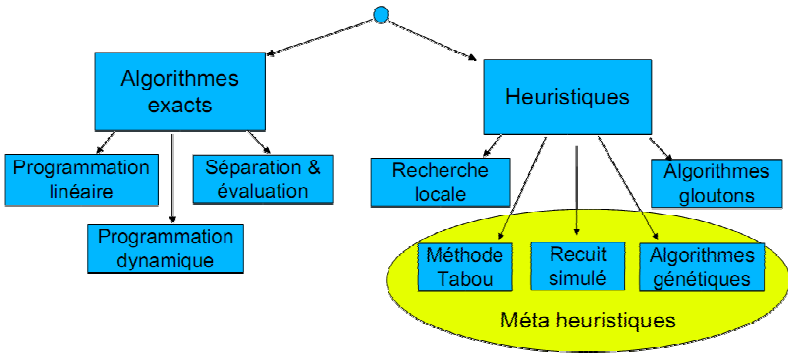
- liste

Marseille → Lyon → Dijon → Paris → Lille → …

- matrice
- …

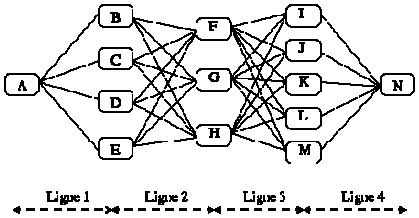
Quelques algorithmes classiques (3/14)

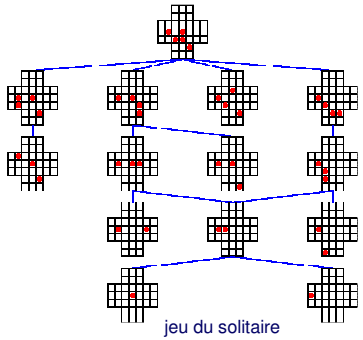
- Taxinomie (merci à Laurent Lemarchand de l'université de Bretagne Occidentale UBO)



Quelques algorithmes classiques (4/14)

- Méthodes complètes :
  - simplexe, programmation linéaire (PLNE, PL, etc.) (cf. Laure Gonnord)
  - programmation dynamique (Bellman) : diviser pour mieux régner
  - techniques « branch-and-bound » : séparation et évaluation
  - arbres, treillis, …





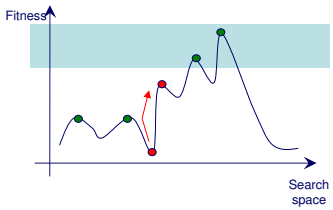
jeu du solitaire

Quelques algorithmes classiques (5/14)

Algorithm 1 Algorithme de recherche locale max

```
1: procédure LOCALE(P,VOISINAGE(.),ÉVALUATION(.))          ▷ Hill-climbing
2:     S ← choisir un point de départ pour P
3:     répéter
4:         V ← VOISINAGE(S)
5:         S ← argmax_{v∈V} {ÉVALUATION(V)}
6:     jusqu'à S n'est plus améliorable
7:     retourne S
8: fin procédure
```

- souvent un abus de langage !
  - algorithme hill-climbing : recherche
  - algorithme glouton : construction
- méthode incomplète
- « instance-based »

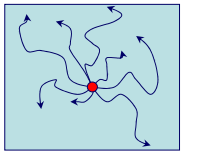


Quelques algorithmes classiques (6/14)

Algorithm 2 Algorithme de recherche aléatoire

```
1: procédure ALÉATOIRE(P,VOISINAGE(.),RANDOM(.))          ▷ Random-walk
2:     S ← choisir un point de départ pour P
3:     répéter
4:         V ← VOISINAGE(S)
5:         S ← RANDOM(v ∈ V)
6:     jusqu'à critère d'arrêt
7:     retourne S                                  ▷ ou bien la meilleure solution trouvée...
8: fin procédure
```

- Beaucoup d'exploration de l'espace…
- …aucune d'exploitation.
- heuristique très similaire : random
  - tirer solutions aléatoirement
  - garder la meilleure
- méthode incomplète
- « instance-based »



point de départ

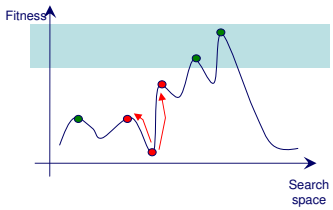
espace de recherche

Quelques algorithmes classiques (7/14)

Algorithm 3 Algorithme de recherche locale seuillée

```
1: procédure LOCALE-SEUIL(P,VOISINAGE(.),ÉVALUATION(.),Seuil)
                                                          ▷ Threesold-accepting
2:     S ← choisir un point de départ pour P
3:     répéter
4:         V ← VOISINAGE(S)
5:         V ← {v ∈ V : ÉVALUATION(v) ≥ Seuil}
6:         S ← RANDOM(v ∈ V)
7:     jusqu'à S n'est plus améliorable
8:     retourne S
9: fin procédure
```

- Presque un hill-climbing
  - mais moins « strict »
- encore une recherche locale
- ➔ méthode incomplète
- « instance-based »



Quelques algorithmes classiques (8/14)

Algorithm 4 Algorithme de recuit simulé

```
1:  procédure RECUIT-SIMULÉ(P,VOISINAGE(.),T)            ▷ Simulated-annealing
2:      S ← choisir un point de départ pour P
3:      répéter
4:          V ← VOISINAGE(S)
5:          v ← RANDOM(v ∈ V)
6:          Δf ← (ÉVALUATION(v) − ÉVALUATION(S))
7:          si Δf > 0 donc                               ▷ Si ça améliore la solution...
8:              S ← v
9:          sinon
10:             p ← RANDOM([0 : 1])
11:             si p ≤ e^(−Δf/T) donc                    ▷ ...ou bien avec une probabilité
12:                 S ← v
13:             sinon
14:                 continue
15:             fin si
16:         fin si
17:         T ← REFROIDISSEMENT(T)                       ▷ réduction de la température
18:     jusqu'à arrêt                                    ▷ critère d'arrêt à définir
19:     retourne S
20: fin procédure
```

- inspiré du phénomène de cristallisation par la chaleur
- gestion de la baisse de T°C
  - paliers,
  - linéaire,
  - etc.
- recherche locale avec parasitage par le bruit
  - bruit = exploration
  - locale = exploitation
- ➔ bon compromis exploration / exploitation

- méthode incomplète
- « instance-based »

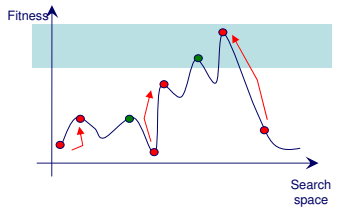
Quelques algorithmes classiques (9/14)

Algorithm 5 Algorithme de recherche locale avec recommencement

```
1: procédure LOCALE-RECOMMENCE(P,VOISINAGE(.),n)          ▷ Multistart
2:     pour i = 1 → n faire
3:         S_i ← choisir un point de départ aléatoire pour P
4:         S_i ← LOCALE(S_i)
5:     fin pour
6:     retourne argmax_{S_i} {ÉVALUATION(S_i)}
7: fin procédure
```

- puisque la recherche locale stagne dans les optima locaux…
  - on relance plusieurs fois à partir de solutions aléatoires

- méthode incomplète
- « instance-based »



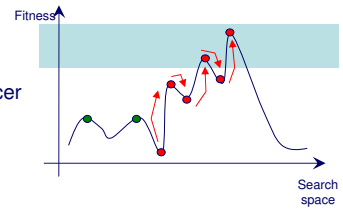
Quelques algorithmes classiques (10/14)

Algorithm 6 Algorithme de recherche locale itérative

```
1: procédure LOCALE-ITÉRATIVE(P,VOISINAGE(.),n)           ▷ Iterated-HC
2:     S ← choisir un point de départ pour P
3:     pour i = 1 → n faire
4:         S ← LOCALE(S)
5:         V ← VOISINAGE(S)
6:         S ← RANDOM(v ∈ V)                  ▷ aléatoire, ou avec du bruit !
7:     fin pour
8:     retourne S
9: fin procédure
```

- quand la recherche stagne…
  - on s'autorise des mouvements « pénalisants »
  - faire une faute pour mieux recommencer

- méthode incomplète
- « instance-based »



Quelques algorithmes classiques (11/14)

Algorithm 7 Algorithme de recherche taboue (une version spécifique)

```
1:  procédure RECHERCHE-TABOUE(P,VOISINAGE(.),OUBLI(.))   ▷ Tabu-search
2:      S ← choisir un point de départ pour P
3:      FIFO ← ∅
4:      répéter
5:          V ← VOISINAGE(S) \ FIFO
6:          M ← {v ∈ V : ÉVALUATION(v) ≥ ÉVALUATION(S)}
7:          si M == ∅ donc
8:              S ← RANDOM(v ∈ V)
9:          sinon
10:             S ← RANDOM(m ∈ M)
11:         fin si
12:         FIFO ← ADD-TO(FIFO, S)
13:         FIFO ← OUBLI(FIFO)              ▷ possibilité de jouer sur la taille
14:     jusqu'à arrêt                       ▷ toujours pareil : critère d'arrêt définissable
15:     retourne S
16: fin procédure
```

- utilisation d'une mémoire pour prohiber les derniers coups
  - dernières solutions
  - ou derniers mouvements
- utilisation d'une fonction d'oubli
- évite les phénomènes de cycles dans les recherches locales bruitées
- variante : recherche locale réactive
  - la taille de la mémoire s'adapte au cours du temps

- méthode incomplète
- « instance-based »

principe du « First-in First-out » ou file :

Quelques algorithmes classiques (12/14)

Algorithm 8 Algorithme génétique

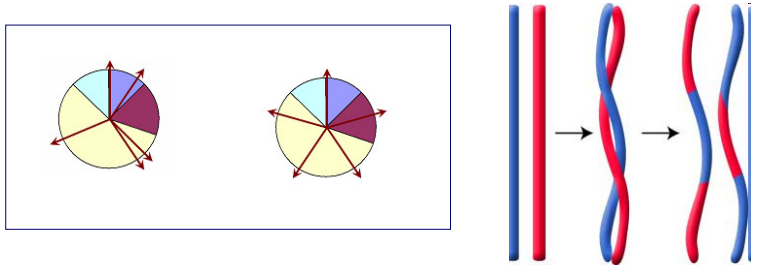
1: **procédure** GENETIQUE($\mathcal{P}$,*Opérateurs*, n, m) ▷ Genetic-algorithm
Opérateurs :
- FITNESS(.)
- SÉLECTION(.)
- CROISEMENT(.,.)
- MUTATION(.) ▷ un opérateur de voisinage sans contrainte de taille

2: $Pop \leftarrow$ choisir n points de départ pour $\mathcal{P}$
3: **répéter**
4: $E \leftarrow$ ÉVALUATION(Pop)
5: $F \leftarrow$ FITNESS(E)
6: $SubPop \leftarrow$ SÉLECTION(Pop, E, m)
7: **pour** $< I_1, I_2 > \in SubPop$ **faire** ▷ I_1&I_2 choisis aléatoirement
8: CROISEMENT(I_1, I_2) ▷ recombinaison
9: **fin pour**
10: **pour** $I_3 \in SubPop$ **faire** ▷ I_1 choisi aléatoirement
11: MUTATION(I_1)
12: **fin pour**
13: **jusqu'à** arrêt
14: **retourne** meilleure solution calculée
15: **fin procédure**

- méthode incomplète évolutive à population
- approche « instance-based » ou « model-based » selon le codage

- utilisation d'opérateurs comparables à ceux de la génétique (biomimétisme)
- sélection
  - élitisme
  - proportionnelle
  - tournois
  - roulette
  - stochastique uniforme
  - par les restes…
- croisement
  - en un point
  - en deux points
  - multipoints
  - uniforme…
- mutation
  - uniforme
  - en un point…
- fitness scaling
  - top
  - rank
  - proportionnel…

Quelques algorithmes classiques (13/14)



Quelques algorithmes classiques (14/14)

Algorithm 9 Colonies de fourmis – ACO

1: **procédure** FOURMIS($\mathcal{P}$, n, α, β, Q, ρ) ▷ Ant-system
2: initialisation phéromones : $\tau_{ij}(0)$
3: initialisation visibilité : η_{ij}
4: $t \leftarrow 0$
5: **répéter**
6: **pour** $k = 1 \rightarrow n$ **faire** ▷ pour chaque fourmi
7: $i \leftarrow$ sommet initial
8: $T^k(t) \leftarrow \emptyset$
9: **tant que** modèle $T^k(t)$ non construit **faire**
10: $J_i^k \leftarrow$ VOISINAGE(i)
11: $p_{ij}^k(t) = \begin{cases} \frac{\tau_{ij}(t)^\alpha \cdot \eta_{ij}^\beta}{\Sigma_{l \in J_i^k} \tau_{il}(t)^\alpha \cdot \eta_{il}^\beta} & \text{si } j \in J_i^k \\ 0 & \text{si } j \notin J_i^k \end{cases}$
12: $next \leftarrow$ RANDOM($p_{ij}^k(t)$) ▷ règle aléatoire de transition prop.
13: ajouter $next$ à la solution $T^k(t)$
14: **fin tant que**
15: $\Delta\tau_{ij}^k(t) = \begin{cases} \frac{Q}{L^k(t)} & \text{si } (i,j) \in T^k(t) \\ 0 & \text{si } (i,j) \notin T^k(t) \end{cases}$ ▷ dépôt de phéromones
16: **fin pour**
17: $\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k(t)$ ▷ évaporation
18: $t \leftarrow t+1$
19: **jusqu'à** arrêt
20: **retourne** meilleure solution modélisée
21: **fin procédure**

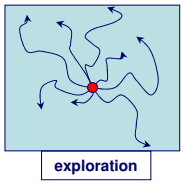
approche « model-based », évolutive, à population

- méta heuristique très complexe
- nombreuses variantes
  - Ant System
  - Min-Max Ant System
  - Ant Colony Optimization
  - …
- schéma phéromonal
  - arcs
  - sommets
- initialisation des phéromones ?
- choix Alpha et Beta
- visibilité
- taux d'évaporation
- taux de dépôt
- etc.

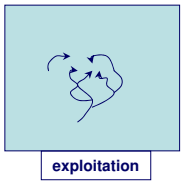
- nombreux paramètres
- mais un modèle mathématique formel très puissant

Dilemme intensification / diversification

- ou exploitation / exploration
  ➔ le point commun de tous ces algorithmes dits « de recherche »
- deux opérations *a priori* inconsistantes
- augmentation de l'exploitation au cours de la recherche



e.g. : random-walk



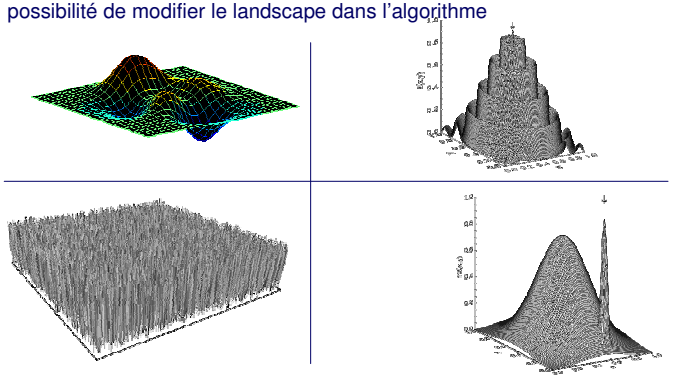
e.g. : hill-climbing

design des méthodes

- Le design est une phase très complexe de l'implémentation :
  - le problème
  - les données
  - paradigme d'évolution
  - type de modèle support
- paramétrage difficile
- possibilité d'hybrider ces méthodes
  - recherche locale pour améliorer des individus dans GA…

notion de landscape

- en français : paysage de recherche
  - façonné par le voisinage, mutation, etc.
  - pas tous de même difficulté
  - possibilité de modifier le landscape dans l'algorithme



notion de transition de phase

- certains problèmes NP-complets sont couramment résolus par des méthodes complètes en temps polynomial
  - Pourquoi ?

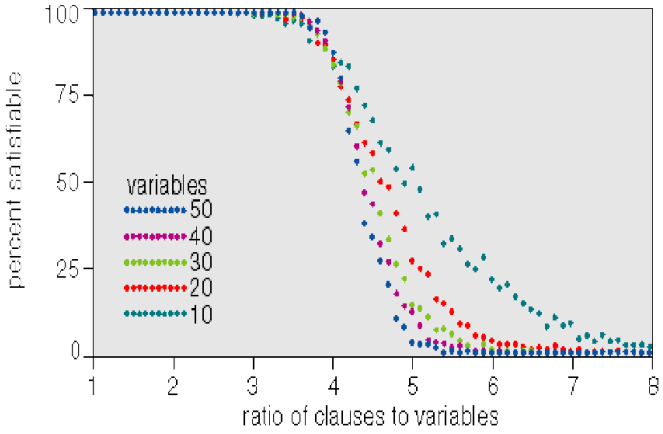
- toutes les instances d'un même problème ne sont pas difficiles
- existence d'un « pic de difficulté »

- certains problèmes appliqués à des données réelles ne connaissent pas ce « pic »

Transition de phases

merci à Christine Solnon

Pourcentage d'instances satisfiables en fct du ratio $\frac{\text{nb clauses}}{\text{nb variables}}$:

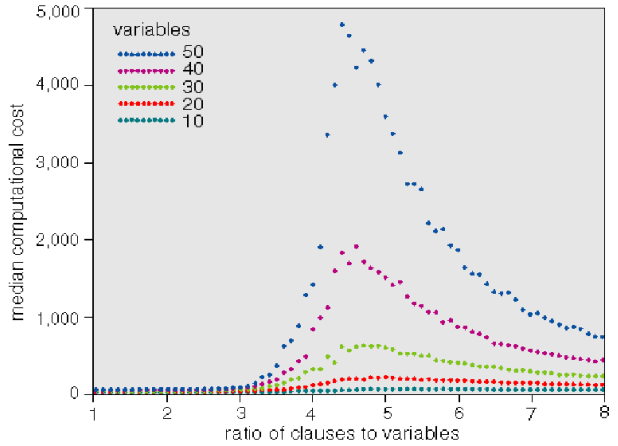


(image empruntée à des transparents de Toby Walsh)

Transition de phases

merci à Christine Solnon

Temps moyen pour décider de la satisfiabilité en fct du ratio $\frac{\text{nb clauses}}{\text{nb variables}}$:



(image empruntée à des transparents de Toby Walsh)

Théorème « no-free-lunch »

- Pas de dîner gratuit !
- Aucune méta ne bât une autre méta sur tous les problèmes
  - (au sens de instance de problème)
- énoncé :
- Wolpert, D.H., Macready, W.G. (1997), *No Free Lunch Theorems for Optimization*, IEEE Transactions on Evolutionary Computation **1**, 67.
- démonstration (simple) :
- Ho, Y.C., Pepyne, D.L. (2002), *Simple Explanation of the No-Free-Lunch Theorem and Its Implications*, Journal of Optimization Theory and Applications **115**, 549.