

Méta-Heuristiques

M.-J. Huguet



<https://homepages.laas.fr/huguet>
2018-2019



Plan

- 1. Introduction**
- 2. Optimisation Combinatoire**
 1. Quel est le problème ?
 2. Démarche de résolution
 3. Familles de méthodes (exactes, approchées)
- 3. Méthodes approchées**
 1. Heuristiques gloutonnes
 2. Méthodes de recherche locale
 3. Méthodes à population
- 4. Conclusions et ouvertures**

Section 1. Introduction

- Contexte
- Exemples : SAT, TSP
- Rappels sur les classes de complexité
- Lien problèmes de décision et d'optimisation combinatoire

3

Contexte

- **Résolution de problèmes**
 - Intelligence Artificielle & Optimisation Combinatoire
 - Déterminer une séquence d'actions pour atteindre un but
 - Formalisation du problème / du but
 - Méthodes de recherche (informées, non informées) dans un graphe d'états
- **Performances des méthodes**
 - **Complétude** : est-ce que la méthode garantit l'obtention d'une solution si elle existe
 - **Optimalité** : est-ce que la méthode permet de déterminer une solution optimale ?
 - **Complexité temporelle** et spatiale : combien de temps et de mémoire nécessite la méthode ?

4

Exemple : Problème de Satisfiabilité (SAT)

- **Satisfaire une formule booléenne écrite sous forme de clauses**

- Ex : $(a \vee \neg b) \wedge (\neg a \vee c \vee d) \wedge (\neg b \vee \neg c \vee \neg e) \wedge (a \vee b \vee d \vee e)$
 - Conjonction de clauses

- **Problème de décision**

- Peut-on satisfaire toutes les clauses ?
- Problème NP-complet dans le cas général
 - Mais problème polynomial si la taille des clauses est au max de 2 éléments

- **Problème d'optimisation**

- Satisfaire le plus grand nombre de clauses
- Problème NP-difficile dans le cas général

- **Nombre de solutions**

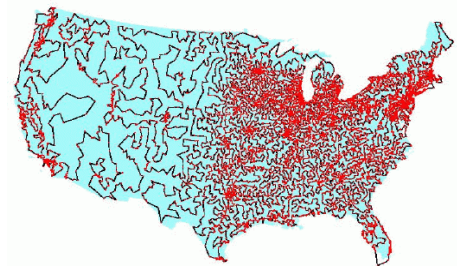
- Si n variables : 2^n solutions

5

Exemple : Voyageur de Commerce (1)

- **ou Traveling Salesman Problem (TSP)**

- Graphe $G(X, A)$: (supposé non orienté)
 - X : ensemble des villes
 - A : ensemble des arêtes
 - Chaque arête est évaluée par la longueur de la route



- **Problème de décision**

- Trouver un cycle hamiltonien de longueur $\leq K$
- Problème NP-complet

- **Problème d'optimisation**

- Trouver le cycle hamiltonien de plus petite longueur
- Problème NP-difficile

- **Nombre de solutions**

- Si n sommets : $(n - 1)!$ solutions (nombre de permutations)

6

Exemple : Voyageur de Commerce (2)

Bref rappel de Complexité (2)

- **La classe P**

- Problèmes pour lesquels il existe un algorithme polynomial
 - Complexité problème $A \leq O(n^k)$
 - n : taille des données en entrée de A ; k : constante indépendante de n
 - Exemple : rechercher un élément dans un tableau, plus court chemin , ...

- **La classe NP**

- Problèmes pour lesquels il est possible de **vérifier une solution** en temps polynomial (certificat)
 - Problème pour lesquels il existe un algorithme polynomial sur une machine de Turing non déterministe

- **Propriété**

- $P \subseteq NP$
- Conjecture : $P \neq NP$

9

Bref rappel de Complexité (3)

- **Problèmes NP-Difficiles**

- Les problèmes au moins aussi difficile que tous ceux de la classe NP

- **Problèmes NP-Complets**

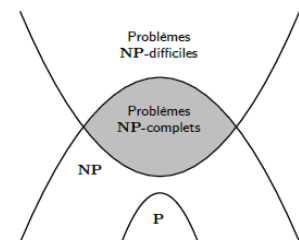
- Un problème est NP-Complet s'il est NP-difficile et qu'il est dans la classe NP

- **Résultats**

- Cook, 1971 : SAT est NP-Complet
- NP-complet \subseteq NP-Difficile

- **Démonstration de la NP-Complétude d'un problème A**

- Montrer que le problème A appartient à NP (certificat polynomial)
- Trouver une procédure polynomiale pour transformer un problème B connu comme NP-Complet vers le problème A (réduction polynomiale de B vers A)



10

Lien décision et optimisation

- **Problème de décision**

- Existe-t-il une solution satisfaisant une certaine propriété ?
- Résultat : OUI / NON
 - Ex : le graphe G donné est-il coloriable en 4 couleurs ?

- **Problème d'optimisation**

- Parmi les solutions satisfaisant une certaine propriété, quelle est celle qui optimise une fonction objectif (fonction de coût)
- Résultat : une solution de coût optimal
 - Ex : Quel est le nombre minimum de couleur pour colorier le graphe G donné ?

- **Résoudre un problème d'optimisation**

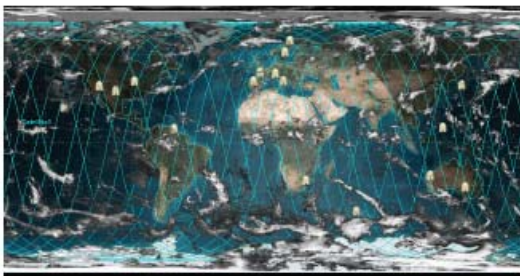
- Au moins aussi difficile que résoudre le problème de décision associé
 - Si le problème de décision est NP-Complet, le problème d'optimisation associé est dit NP-Difficile

11

Optimisation Combinatoire

- **Exemples dans le secteur spatial :**

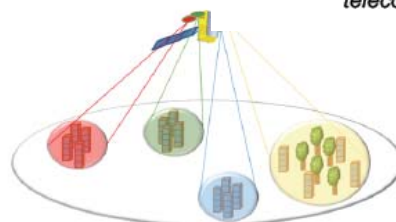
- Déterminer la meilleure solution parmi un ensemble discret (combinatoire) de solutions possibles



Placement de stations-sol optiques pour optimiser la transmission en présence de nébulosité



Séquencement de tests de satellites de télécommunication



Optimisation intégrée de l'allocation de fréquences et du placement de faisceau

12

Section 2. Optimisation Combinatoire

- Définitions générales
- Démarche de résolution
- Caractéristiques de différents types de méthodes
 - Méthodes exactes
 - Méthodes avec garantie
 - Méthodes approchées

Formalisation (1)

- **Problème** (X, D, C, f) :
 - X : ensemble des variables du problème
 - D : domaine des variables : $x_i \rightarrow D(x_i)$: les valeurs pouvant être prises par x_i
 - **Espaces des solutions (alternatives)** A : produit cartésien des domaines
 - Taille de l'ensemble des solutions : $D_1 \times D_2 \times \dots \times D_n$
 - Une solution : instantiation des variables $s : \{(x_i \leftarrow v_i)\}, v_i \in D_i, \forall x_i \in X$
 - C : contraintes du problème
 - Relations entre les variables X : restreindre les valeurs possibles
 - **Espace des solutions admissibles / réalisables** Ω :
 - Une solution s telle que les contraintes C sont satisfaites
 - $f: X \rightarrow \mathbb{R}$: fonction Objectif
 - Minimiser (cout/ perte); Maximiser (gain, profit)

Formalisation (2)

- Résoudre un problème d'optimisation (minimisation) :
 - solution $s^* \in \Omega$, telle que $f(s^*) = \min_{s \in \Omega} f(s)$: ie $s^* = \operatorname{argmin}(f)$
 - Trouver des valeur pour toutes les variables de X
 - Toutes les contraintes de C doivent être satisfaites
- Optimisation Combinatoire :
 - minimisation sur un ensemble Ω discret et fini
 - potentiellement de très grande taille
- En pratique : conserver les 2 ensembles A et Ω
 - Un algorithme peut générer des solutions intermédiaires non réalisables avant d'arriver à une solution (optimale) réalisable

Vocabulaire

- **Problème** : par exemple le TSP
- **Instance** : par exemple un ensemble donné de villes, de routes et de distances
- **Solution** : par exemple une tournée pour le TSP
 - **Solution candidate (alternative)**
 - Instanciation des variables; ne respecte pas nécessairement les contraintes
 - **Solution réalisable**
 - Respecte les contraintes
 - **Solution optimale**
 - La meilleure solution / fonction objectif
 - Plusieurs solutions peuvent avoir la même valeur de fonction objectif
 - La valeur de l'optimum est unique

Démarche (1)

- **Problème**

- Comprendre le problème (données, paramètres, contraintes, objectifs)
- Analyser le problème :
 - est-il facile ou difficile (en termes de complexité) ?
 - Ressemble-t-il à un problème déjà connu (état de l'art) ?

- **Modèle**

- A un même problème peut correspondre plusieurs modèles
- Différents formalismes de modélisation
 - Logique, Graphe, Programmation Linéaire,
 - Programmation Linéaire en Nombres entiers ou Mixtes
 - Programmation par Contraintes
 - Etc

17

Démarche (2)

- **Méthode**

- Quelle méthode utiliser pour résoudre le problème ?
- Adapter une méthode existante ou développer une méthode ad-hoc ?
- S'assurer que la méthode fournit des résultats corrects (tests de correction)
 - La solution obtenue est-elle bien réalisable ?
 - Si pas de solution est-ce parce qu'il n'en existe pas ?

- **Evaluation expérimentale**

- Mettre en place une campagne de tests pour évaluer les performances de la méthode
 - A quels autres résultats se comparer ? Existe-t-il des benchmarks de ce problème ?
 - Jeux d'instances de tests de performances (académiques, réels,) ?
 - Autres méthodes ?
 - Les tests de performance sont-ils représentatifs des instances du problème ?
 - Les tests effectués sont-ils reproductibles ?
 - Les résultats obtenus sont-ils conformes à l'analyse du problème ?

18

Types de problèmes (1)

- **Types de problèmes :**
 - Planifier, Ordonnancer (Planning, Scheduling, Allocation)
 - Transporter (Routing, Transportation)
 - Découpe et Empilement (Cutting, Packing)
- **Problèmes académiques** (TSP, Bin-Packing,) / **Problèmes réels**
- **Secteurs économiques :**
 - Secteur industriel (gestion production, stratégie)
 - Services, Hôpitaux, Administrations
 - Logistique et Transport
 - Aéronautique, Espace
 - Réseaux et Télécom, Systèmes Embarqués
 - Développement durable (gestion de l'énergie, de ressources, ...)
 - etc.

19

Types de problèmes (2)

- **Objectifs usuels**
 - Temps
 - Argent
 - Quantité
 - Equilibrage (ex : utilisation de ressources)
- Qualité de service
- Robustesse
- Mais aussi :
 - Sécurité
 - Environnement
 - Vie privée

20

Résolution de problèmes combinatoires

- **Problèmes d'optimisation combinatoire NP-Difficile**

- Espace de recherche : espace exploré pour trouver la solution (optimale)
 - Solutions candidates ou réalisables
- Comment faire face à l'explosion combinatoire ie. la taille de l'espace de recherche ?

- **Méthodes génériques**

- Contenir l'explosion combinatoire en structurant et en élaguant l'exploration de l'espace de recherche → **méthodes exactes**
- Contourner l'explosion combinatoire en faisant des impasses (sans explorer tout l'espace de recherche) → **méthodes approchées**

21

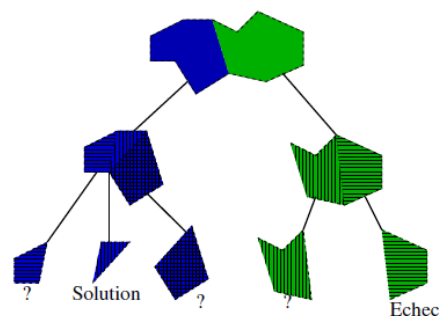
Méthodes exactes

- **Principe :**

- Structurer l'exploration de l'espace de recherche de solutions en arbre
 - **Recherche arborescente**

- Construction d'un arbre de recherche

- Séparation :
 - Partitionner en sous espace
- Evaluation :
 - L'espace obtenu est-il une solution ?
 - Correspond-il à un problème facile ?
 - Est-il réalisable (échec) ?
- Sélection :
 - Choisir la prochaine partie à explorer



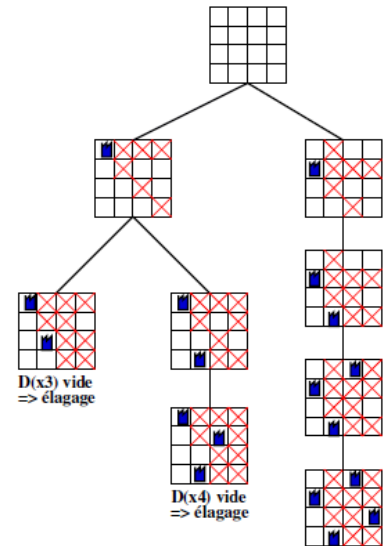
22

Méthodes exactes

- **Autres principes :**

- Filtrer et/ou élaguer → supprimer des parties de l'arbre
 - Parties ne pouvant conduire à une solution
 - Parties ne pouvant conduire à une bonne solution
- Sélection (Heuristiques d'ordre) → explorer en premier les branches les plus prometteuses

Ex : pb des 4 reines



23

Exemple (1)

- **Problème de planification**

- Un ensemble d'états E
- Un état initial E_0 , un ensemble d'états finaux F
- Un ensemble d'actions/opérateurs permettant d'effectuer des transitions entre états : $E_i \rightarrow^{o_k} E_j$

- Un plan (une solution) :

- Séquence d'actions permettant de passer de l'état initial à un état final
 - $E_0 \rightarrow^{o_1} E_1 \rightarrow^{o_2} E_2 \rightarrow \dots \rightarrow^{o_k} E_n$, avec $E_n \in F$

- Problème d'optimisation :

- Trouver le plan de coût minimal
 - Définir le coût de chaque action
- Espace de recherche : ensemble des séquences d'actions débutant en E_0

24

Exemple (2)

- **Résolution du problème de planification**

- Recherche arborescente : Algorithme A*

- Arbre

- Racine : suite vide d'actions (état initial E_0)
- Nœud : suite d'actions partant de E_0

- **Séparation** : 1 fils pour chaque action (transition) possible

- **Evaluation** : 2 fonctions de coût

- $g(n)$: coût des actions pour aller de la racine au nœud n
- $h(n)$: estimation du coût des actions pour atteindre un état final à partir de nœud n
- Evaluation : $f(n) = g(n) + h(n)$

- **Sélection** :

- nœud ayant la plus petite valeur de $f(n)$

25

Exemple (3)

- **Application vue en TP : problème du taquin**

- Etat initial =

	B	C
A	D	F
G	E	H

- Etat final =

A	B	C
D	E	F
G	H	

- Opération = échanger la case vide avec une case adj.
- But = trouver le plus petit plan (en nombre d'opérations)

- Coût du début du plan = nombre d'opérations déjà effectuées

- Estimation du coût pour atteindre l'état final = **dist de Manhattan**

B	D	C
	A	F
G	E	H

$$\delta(A) = 2 \quad \delta(D) = 2 \quad \delta(G) = 0$$

$$\delta(B) = 1 \quad \delta(E) = 1 \quad \delta(H) = 1$$

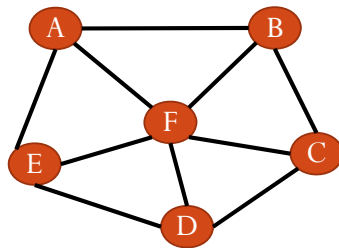
$$\delta(C) = 0 \quad \delta(F) = 0$$

⇒ au moins 7 opérations pour aller jusqu'à l'état final

26

Exercice

• Recherche arborescente pour le TSP (exemple A*)



	A	B	C	D	E	F
A	--	9			3	5
B	9	--	5			4
C		5	--	2		8
D			2	--	1	7
E	3			1	--	5
F	5	4	8	7	5	--

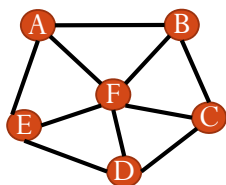
- Etat initial : une ville X
- Etats Finaux : toute séquence X*X
- Changement d'état : ajouter une ville dans la séquence
- Fonctions d'évaluation
 - $g(X*Y)$ = cout depuis X jusqu'à Y
 - $h(Y*X)$ = estimation du trajet restant depuis X

Nb arcs restant x cout min (possible)
Somme des p arcs de cout min
.....

→ **Borne Inférieure de l'optimum**

27

Exercice



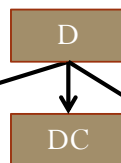
1+ 5 arcs restants x 1=6

	A	B	C	D	E	F
A	--	9			3	5
B	9	--	5			4
C		5	--	2		8
D			2	--	1	7
E	3			1	--	5
F	5	4	8	7	5	--

2+ 5 arcs restants x 1=7

	A	B	C	D	E	F
A	--	9			3	5
B	9	--	5			4
C		5	--	2		8
D			2	--	1	7
E	3			1	--	5
F	5	4	8	7	5	--

6 arcs restants x 1=6



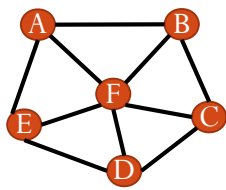
DF

7+ 5 arcs restants x 1=12

	A	B	C	D	E	F
A	--	9			3	5
B	9	--	5			4
C		5	--	2		8
D			2	--	1	7
E	3			1	--	5
F	5	4	8	7	5	--

28

Exercise



DE

1 + 5 arcs restants x 1 = 6

DEA

DEF

4 + 4 arcs restants x 2 = 12

6 + 4 arcs restants x 2 = 14

	A	B	C	D	E	F
A	--	9			3	5
B	9	--	5			4
C		5	--	2		8
D			2	--	1	7
E	3			1	--	5
F	5	4	8	7	5	--

	A	B	C	D	E	F
A	--	9			3	5
B	9	--	5			4
C		5	--	2		8
D			2	--	1	7
E	3			1	--	5
F	5	4	8	7	5	--

D

6 arcs restants x 1 = 6

1 + 5 arcs restants x 1 = 6

DE

DC

DF

2 + 5 arcs restants x 1 = 7

7 + 5 arcs restants x 1 = 12

DEA

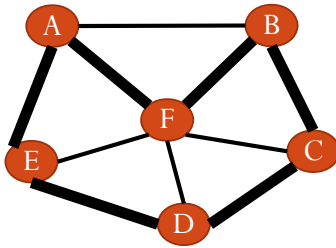
DEF

4 + 4 arcs restants x 2 = 12

6 + 4 arcs restants x 2 = 14

Exercice

- Solution optimale :



Cout = 20

	A	B	C	D	E	F
A	--	9			3	5
B	9	--	5			4
C		5	--	2		8
D			2	--	1	7
E	3			1	--	5
F	5	4	8	7	5	--

- Exploration
 - Les plus prometteurs d'abord
 - Quand une solution est trouvée
 - Elaguer les nœuds ayant évaluation > solution courante
 - Intérêt d'avoir une évaluation (borne inférieure) de bonne qualité
 - **Branch and Bound**

31

Bilan sur les méthodes exactes

- **Approches correctes et complètes**
 - Fournissent la solution optimale (s'il existe des solutions)
- **Complexité exponentielle**
 - Sont limitées à des problèmes de taille raisonnable
- **Différentes variantes des méthodes arborescentes**
 - En liaison avec les formalismes de modélisation
 - Rappel : Algorithme DPLL pour SAT (Logique et Programmation Logique)
 - Instanciation progressive de variables (ordre de parcours)
 - Elagage de l'arborescence (simplifier les clauses)
 - Retour arrière chronologique en cas d'échec
 - Rappel : Algorithmes de Programmation Dynamique (Algorithmique avancée)

32

Bilan sur les méthodes exactes

- **La suite en 5^e année SDBD – Mineure Analyse Prescriptive**
 - Etudes de différents formalismes, de méthodes exactes associées et de solvers
 - Programmation Linéaire en Nombres Entiers,
 - Programmation par Contraintes,
 - Satisfiabilité

Méthodes approchées

- **Principe :**
 - Ne pas explorer tout l'espace de recherche
- **Méthodes avec garantie de qualité / solution obtenue**
 - Algorithmes d'approximation
 - Trouver des solution avec garantie / optimum
- **Méthodes guidées par des heuristiques**
 - Trouver des « bonnes » solutions mais sans garantie / optimum
 - Complexité raisonnable

Algorithmes d'approximation : exemple (1)

- **Algorithme de ρ -approximation :**

- Algorithme polynomial qui renvoie une solution approchée garantie au pire cas à un facteur ρ de l'optimum :
 - $Opt \leq Alg \leq \rho Opt$ (avec $\rho > 1$)
 - avec Opt la solution optimale et Alg la solution obtenue

- **Problème de Bin Packing**

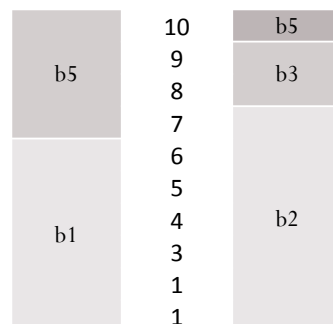
- un ensemble B de n objets (items) de « taille » w_i , $\forall b_i \in B$
- un entier C (capacité des sacs/bin)
- Un entier k
- **Problème de décision** : existe-il un rangement des objets de B dans k sacs de capacité C ? \rightarrow problème NP-Complet
- **Problème d'optimisation** : quel est le nombre minimum de sacs de capacité C pour ranger les objets de B ? \rightarrow problème NP-difficile

35

Algorithmes d'approximation : exemple (2)

- **Exemple**

- $B = \{b_1, b_2, b_3, b_4, b_5\}$; $w = \{6, 7, 2, 4, 1\}$; $c = 10$
- Existe-t-il un rangement de ces objets dans 2 sacs de capacité 10 ?
- Réponse :



36

Algorithmes d'approximation : exemple (3)

• Algorithme Next-Fit

- Entrées : ensemble $B = \{b_1, \dots, b_n\}$ d'objets, fonction $w : B \rightarrow \mathbb{N}$, entier c
- Sortie : un entier k


```
1.  $j \leftarrow 1$  // indice des sacs
2. Pour  $i$  de 1 à  $n$  faire
    • si  $b_i$  peut être rangé dans sac  $S_j$  alors  $S_j \leftarrow S_j \cup \{b_i\}$ 
    • sinon
        •  $j \leftarrow j + 1$ 
        •  $S_j \leftarrow \{b_i\}$ 
3. Retourner  $j$ 
```

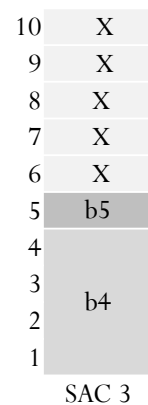
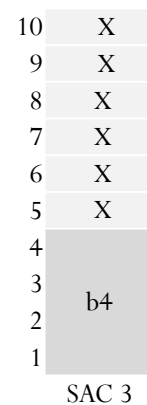
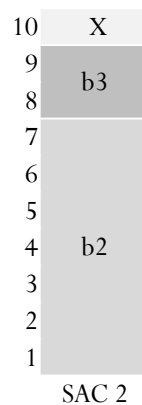
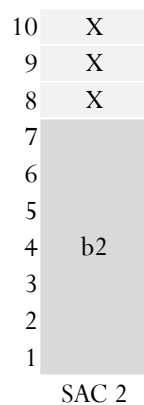
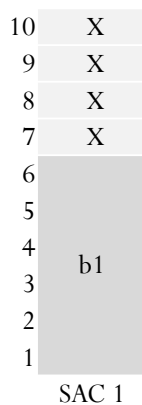
- Complexité : linéaire dans le nombre d'objets

37

Application Algorithme Next-Fit

• Exemple

- $B = \{b_1, b_2, b_3, b_4, b_5\}; w = \{6, 7, 2, 4, 1\}; c = 10$
- Minimiser le nombre de sacs ? 



38

Facteur d'approximation pour Next-Fit (1)

- Soit $w(b_i)$ les tailles des objets, c la capacité des sacs et k la valeur retournée par l'algorithme Next-Fit
- Quel est le nombre minimal théorique de sacs ? $\frac{\sum_{i=1}^n w(b_i)}{c}$
- Donc : $Opt \geq \frac{\sum_{i=1}^n w(b_i)}{c}$ Borne inférieure
- Si on considère le fonctionnement de l'algorithme et regardant les sacs 2 par 2 :
 - $w = \{50, 20, 60, 10, 10, 10, 20, 40, 10, 10, 30, 80, 10, 20, 60, 30\}; c = 100$

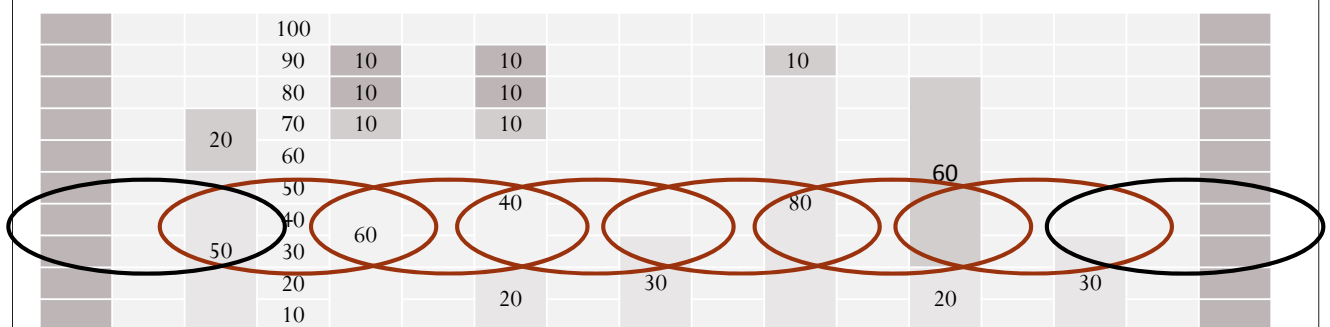


39

- Somme des taille des objets dans 2 sacs consécutifs > capacité d'un sac

Facteur d'approximation pour Next-Fit (2)

- Somme des tailles des objets sur toutes les paires de sacs



- Chaque paire a une taille > capacité (100) et il y a $k - 1$ paires
 - Chaque élément est compté 2 fois (sauf 1^{ère} et dernier sacs)
 - Ajouter des sacs fictifs en début et à la fin (supposés pleins) : 2 paires de plus ($k + 1$) et la taille de chaque objet est compté 2 fois
 - $2 \times \sum_{i=1}^n w(b_i) + 200 > 100 \times (k + 1)$
 - $2 \times \sum_{i=1}^n w(b_i) > 100 \times (k - 1)$
 - $\sum_{i=1}^n w(b_i) > \frac{100 \times (k - 1)}{2}$

40

Facteur d'approximation pour Next-Fit (3)

- On regroupe les inégalités :
 - $Opt \geq \frac{\sum_{i=1}^n w(b_i)}{100}$ et $\sum_{i=1}^n w(b_i) \geq \frac{100 \times (k-1)}{2}$
- C'est à dire :
 - $Opt \geq \frac{\sum_{i=1}^n w(b_i)}{100} > \frac{k-1}{2}$
- D'où
 - $k < 2 \times Opt + 1$ et $k \leq 2 \times Opt$
 - Facteur d'approximation : 2

Pour aller plus loin sur le Bin Packing

- Autres algorithmes d'approximation pour le Bin Packing
 - First Fit Decreasing / Best-Fit Decreasing
 - Tri des objets dans l'ordre décroissant
 - Placer chacun dans le premier/meilleur sac qui peut le contenir
- Méthodes exactes
- **Objets avec plusieurs dimensions**
- **En pratique**
 - Problèmes de rangement/remplissage (de caisses dans des camions, de fichiers sur des supports, ...)
 - Problèmes de découpe

Méthodes approchées

- **Principe :**

- Ne pas explorer tout l'espace de recherche
- **Méthodes avec garantie de qualité / solution obtenue**
 - Algorithmes d'approximation
 - Trouver des solution avec garantie / optimum
- **Méthodes guidées par des heuristiques**
 - Trouver des « bonnes » solutions mais sans garantie / optimum
 - Complexité raisonnable

43

Méthodes basées sur des heuristiques (1)

- **But : trouver une « bonne » solution mais sans garantie sur l'optimalité**
 - Avec une méthode de complexité raisonnable
 - Qui est robuste : fournit souvent une bonne solution
 - Qui est simple en œuvre
- **Grands principes :**
 - **Intensifier** l'exploration de zones prometteuses de l'espace de recherche
 - **Diversifier** pour découvrir de nouvelles zones

44

Heuristique versus Métaheuristique (1)

- Pas de consensus sur des définitions précises. Le plus souvent :
- **Méthode heuristique**
 - Détermine de bonnes solutions (c'est-à-dire presque optimales)
 - avec un coût de calcul raisonnable sans pouvoir garantir ni la faisabilité ni l'optimalité,
 - Reflète une stratégie par rapport à une connaissance du problème
 - Une heuristique est une méthode de résolution spécialisée pour un problème
- **Métaheuristique**
 - processus de génération qui guide une heuristique
 - en combinant des concepts différents pour explorer l'espace de recherche afin de trouver efficacement des solutions quasi optimales
 - Une métaheuristique est un principe générique à adapter pour chaque problème

45

Heuristique versus Métaheuristique (2)

	Heuristique	Métaheuristique
Domaine	Problème d'optimisation	Optimisation combinatoire
Entrée	Données	Problème d'optimisation
Coeur	Algorithme	Ensemble de principes
Sortie	Solution	Algorithme

Classer les ratios poids/coût par ordre décroissant et sélectionner les variables correspondantes tant que la contrainte n'est pas saturée; ...

Mémoire des solutions visitées;
Choisir parmi les a bonnes solutions;
Ajuster la liste t des mouvements interdits dynamiquement;
etc...

46

Exploration de l'espace de recherche

• Principe des méthodes approchées

But : contourner l'explosion combinatoire

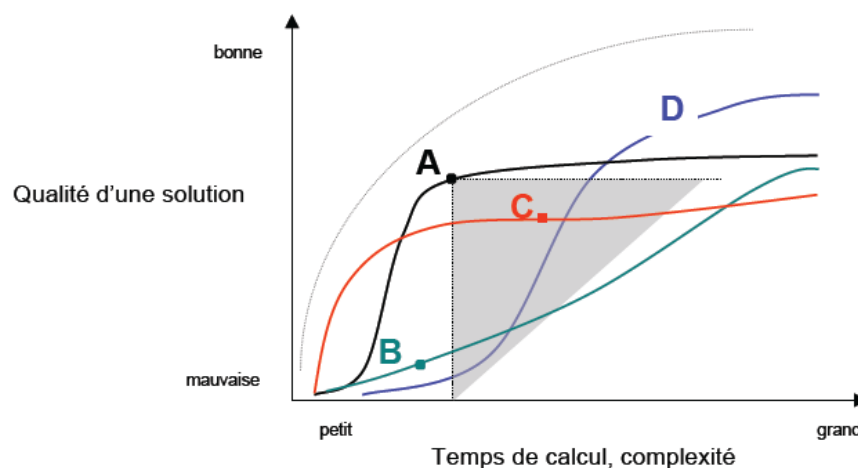
- Intensification :
 - Accentuer l'exploration dans des zones prometteuses
- Diversification :
 - Découvrir de nouvelles zones
- Compromis intensification / diversification
 - Aléatoire
 - Guidé par la fonction objectif
 - Guidé par d'autres évaluations (solutions explorées, contraintes, paysage, ...)

47

Evaluation d'une méthode approchée (1)

• Qualité d'une heuristique / métaheuristique

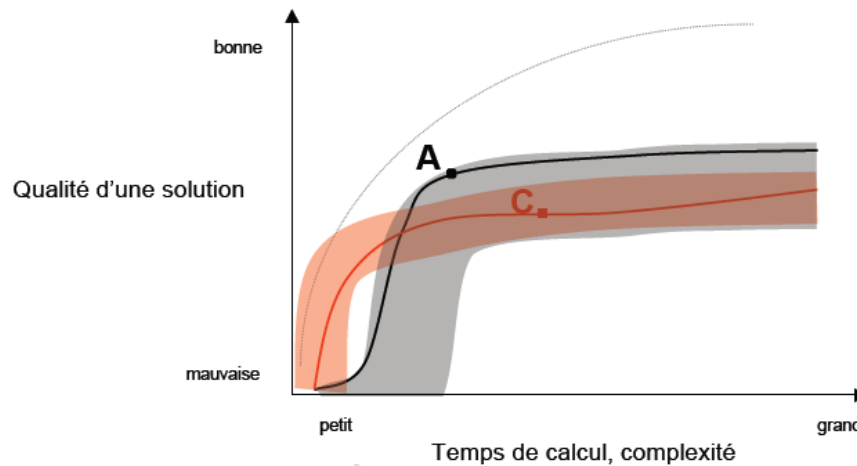
- Comparaison : qualité de solution (pour un temps de calcul donné)



48

Evaluation d'une méthode approchée (2)

- **Qualité / temps de calcul**



49

Evaluation d'une méthode approchée (3)

- **Comparaison solution optimale / borne**

- Hypothèse : problème de minimisation

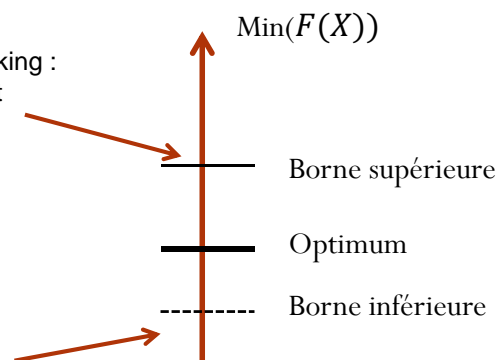
- **Solution obtenue (heuristique / métaheuristique)**

- Borne supérieure de l'optimum

- **Comparaison**

- Optimum si connu
- Borne inférieure de l'optimum
 - Calcul analytique

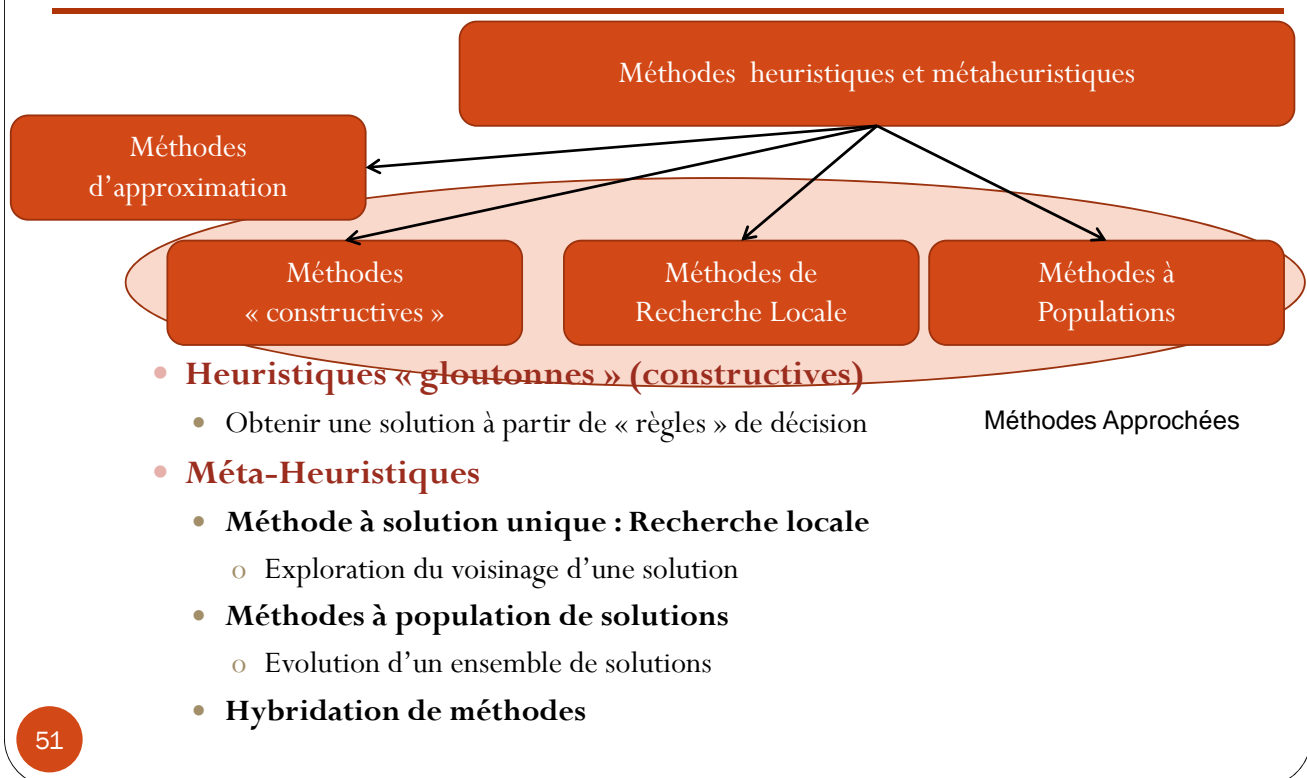
Ex pour le Bin-Packing :
Algorithme Next-Fit



Ex pour le Bin-Packing :
Borne inférieure = $\frac{\sum_{i=1}^n w(b_i)}{c}$

50

Différentes familles de méthodes approchées



Intérêts des méthodes approchées

- **Quand utiliser une méthode approchée ?**
 - Impossible de passer à l'échelle avec une méthode exacte
 - Contraintes de temps de calcul, de mémoire, de modélisation,
- **Une bonne méthode approchée ?**
 - Complexité raisonnable
 - Solution de bonne qualité, rarement de mauvaise solution
 - Simple à mettre en œuvre
- **Evaluation d'une méthode approchée**
 - Comparaison méthodes exactes sur des jeux de données de taille raisonnable
 - Comparaison / bornes / temps de calcul / solutions ...

Autres qualités pour une métaheuristique

- Robustesse
 - Variabilité dans les données (et les contraintes,)
- Simplicité de mise en œuvre
 - Paramétrage de la méthode
- Applicable à une grande variété de problèmes
 - Optimisation combinatoire
 - Optimisation continue
- Attractivité de la métaphore
 - Recuit simulé
 - Recherche Tabou
 - Algorithmes génétiques
 - Colonies de Fourmis
 - Systèmes immunitaires artificiels, essais particuliers, scatter search,

53

(Bilan) sur les caractéristiques des méthodes

- **Résolution de problèmes NP-Complets / NP-Difficiles**
- **Méthodes exactes**
 - Fournissent solution optimale (si existe)
 - Complexité exponentielle : sont limitées à des instances de taille raisonnable
- **Méthodes approchées**
 - Fournissent une solution réalisable mais pas nécessairement optimale
 - Complexité « correcte »
- **En pratique**
 - Certaines instances peuvent être faciles à résoudre
 - Comprendre pourquoi : données ? contraintes ? cas particulier polynomial ?
 - Certains problèmes NP-difficiles admettent des approximations polynomiales
 - Algorithmes polynomiaux permettant le calcul d'une solution avec erreur bornée par rapport à la solution optimale

54

En savoir plus

- **Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF)**
 - Chercheurs/Enseignants Chercheurs/Industriels <http://www.roadef.org>
 - Journées annuelles / Forum stages et emplois (ingénieurs, thèses, post-doc, ...)
 - Liens forts avec Intelligence Artificielle sur l'optimisation combinatoire
- **A Toulouse**
 - Equipe de recherche au LAAS <https://www.laas.fr/public/fr/roc>
 - mais aussi autres établissements (ENAC, ONERA, IRIT, IMT, INRA)
 - Industries et Services
- **A l'international**
 - IFORS (International Federation of Operations Research Societies) <http://ifors.org/>
 - EURO
 - Decision Science / Decision Management / Operations Research

Vocabulaire ...

- **Méthode exacte** : explore l'espace de recherche dans sa totalité (complète)
 - Optimum global
- **Méthode approchée** : explore une partie de l'espace de recherche
 - Optimum local
- **Méthode complète** : explore l'ensemble des solutions de façon exhaustive et systématique en structurant l'espace de recherche
- **Méthode incomplète** : explore seulement une sous-partie de l'ensemble des solutions en utilisant des heuristiques pour se guider vers les zones qui semblent plus prometteuses.
- **Méthode déterministe** : réalise toujours la même suite d'actions
- **Méthode stochastique** : effectue des choix aléatoires (probabilistes)

Vocabulaire ...

- **Hyper-heuristique**

- Heuristique pour déterminer quelle heuristique appliquer

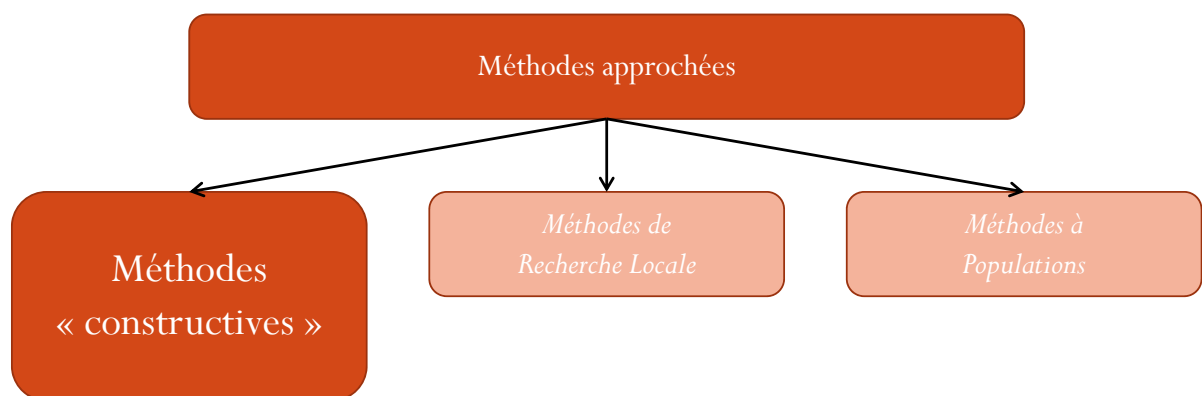
- **Matheuristique**

- Combinaison de méthodes heuristiques et de programmation mathématique
 - Ex : explorer un espace de recherche avec un algo de prog dynamique

57

Section 3. Méthodes approchées

- **Heuristiques gloutonnes**



Principes d'une Recherche Gloutonne

• Méthode naïve

Greedy Search

- Partir d'une affectation vide des variables
- Tant qu'il y a des variables non affectées
 - Choisir une variable
 - Lui affecter une valeur
- Les choix successifs ne sont pas remis en cause
- Une branche d'une recherche arborescente (sans backtrack)
- Les choix effectués doivent garantir l'admissibilité de la solution obtenue (ie. respect des contraintes)
- **Peut-on toujours le garantir ?**

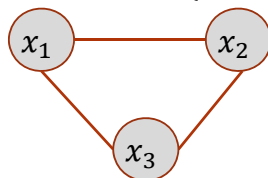
Méthode « Constructive »

59

Exemple heuristique gloutonne (1)

• Coloration de graphe

{bleu, rouge} {bleu, rouge, vert}



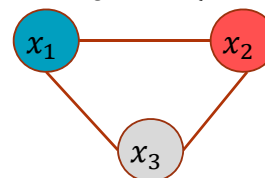
{bleu, rouge}

Minimiser nombre de couleurs

Solution pas toujours admissibles

Ordre Instanciation : x_1, x_2, x_3

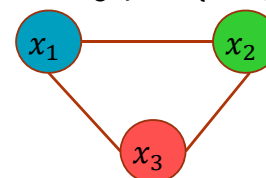
{bleu, ~~rouge~~} {bleu, rouge, ~~vert~~}



{bleu, rouge}

Ordre Instanciation : x_1, x_3, x_2

{bleu, ~~rouge~~} {bleu, rouge, vert}



{bleu, rouge}

60

Exemple heuristique gloutonne (2)

Planification

activités	durée	ressource
A1	1	1
A2	1	2
A3	2	1
A4	1	2
A5	2	1

$$A_1 < A_2 < A_5$$

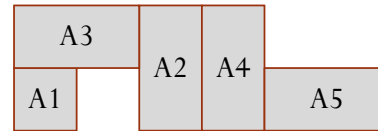
$$A_3 < A_4$$

Quantité de ressource disponible : 2

Minimiser durée totale

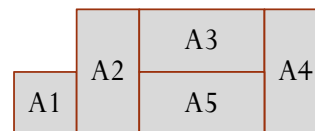
Solutions admissibles
Valeurs différentes / objectif

Instanciation : $A_1; A_3; A_2; A_4; A_5$



Durée Totale = 6

Instanciation : $A_1; A_2; A_3; A_5; A_4$



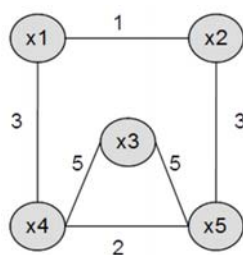
Durée Totale = 5

61

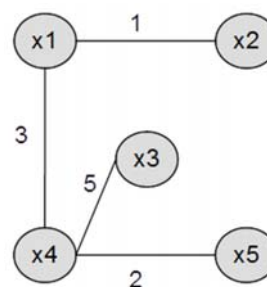
Exemple heuristique gloutonne (3)

Arbres couvrants de cout minimal

- Soit un graphe non orienté et pondéré
- Trouver un arbre couvrant de poids minimal :
 - Sélectionner les arêtes telles que la somme des poids des arêtes soit minimale



Graphe initial



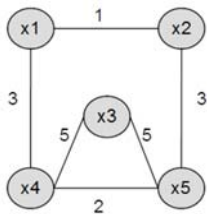
Arbre couvrant de cout minimal (11)

62

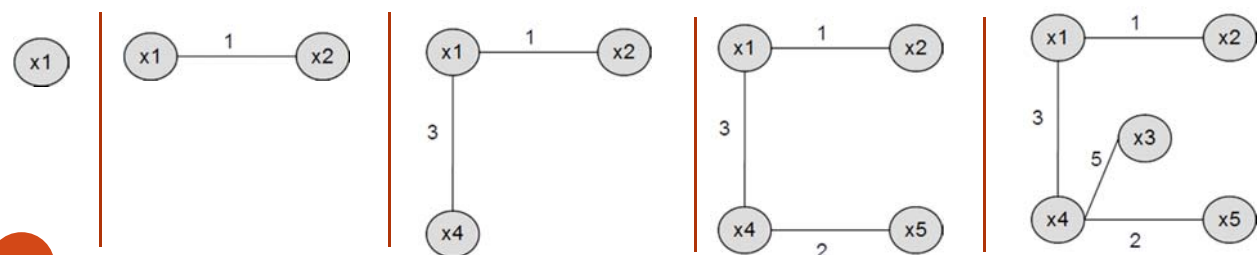
Exemple heuristique gloutonne (4)

- **Arbres couvrants de cout minimal**

- Algorithme de Prim : ajout progressif d'arêtes
 - arbre = graphe connexe avec un nombre minimal d'arêtes
 - Maintenir un graphe connexe à chaque itération en ajoutant une arête pour connecter la partie connexe aux sommets non encore couverts



Algorithme glouton optimal (voir cours Algorithmique avancée 4IR)



63

Caractéristiques d'une Recherche Gloutonne

- **Qualité de la solution / fonction objectif**

- Dépend de l'ordre sur les variables et des choix de valeurs
- **Pas de garantie d'optimalité**
- Obtention d'une **borne supérieure** de la valeur optimale (minimisation)

- **Ordre d'instanciation**

- Quelle variable choisir ?
- Quelle valeur lui affecter ?
- **Principe général**
 - Choix de variable : la plus importante d'abord (fail-first)
 - Choix de valeur : celle ayant le plus de chance de conduire à une (bonne) solution (succeed-first)

64

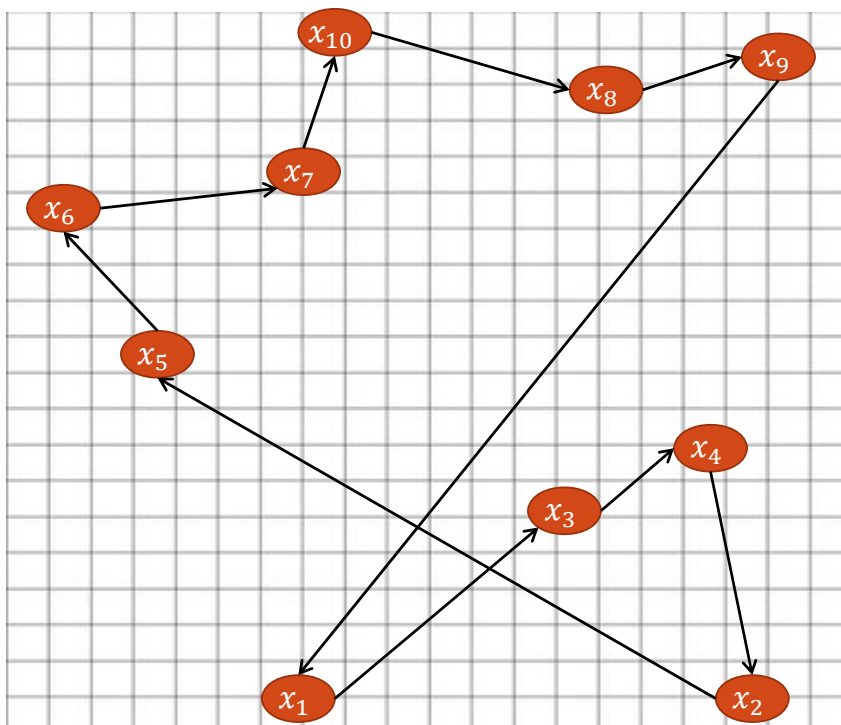
Application 1 : TSP

- **Heuristique : Plus proche Voisin**

- Choisir un sommet de départ
- Tant qu'il reste des sommets non traités
 - Connecter le dernier sommet atteint au sommet libre le plus proche
- Relier le dernier sommet au sommet initial
- Données :
 - Graphe (n sommets) avec matrice de distance (2×2)
- Résultat : cycle (permutation des sommets)
- Nombre de solutions : $(n - 1)!$
- Complexité heuristique : $(O(n^2))$
 - n itérations
 - À chaque itération : trouver le plus proche ($O(n)$)

65

Application 1 : TSP



Borne supérieure de la solution optimale

Garantie de qualité (si symétrique et inégalités triangulaires)

66

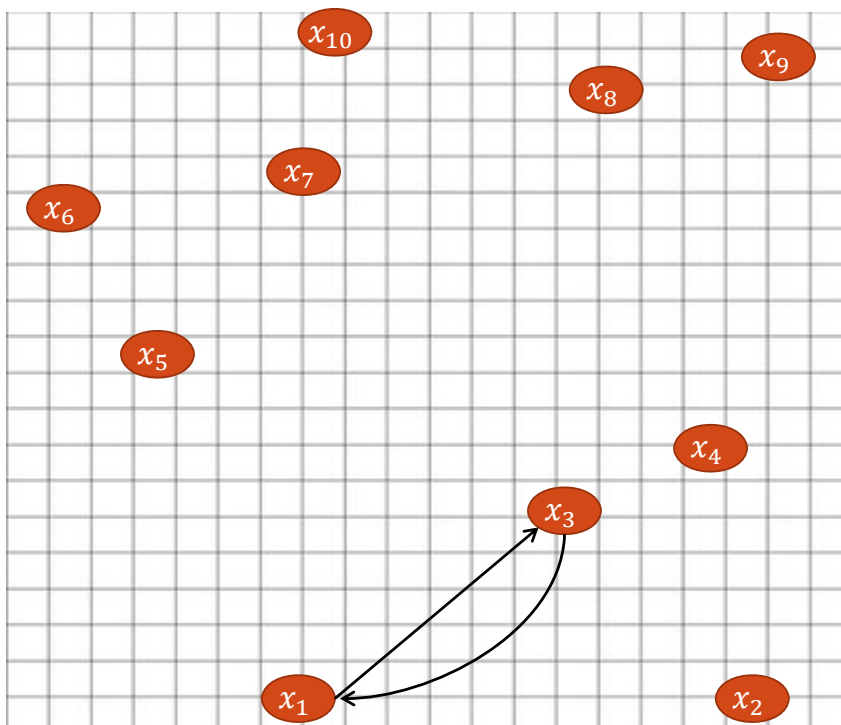
Application 1 : TSP

- **Autre Heuristique : Insertion dans un cycle**

- Choisir un sommet de départ
- A chaque étape : il existe un cycle
 - Insérer dans ce cycle le sommet minimisant un critère
 - Insertion du plus proche voisin : insérer dans le cycle le sommet le plus proche de ceux déjà présents (nearest insertion) – après le plus proche
 - Insertion du voisin à moindre coût : insérer le sommet engendrant la plus petite augmentation de la longueur du cycle (cheapest insertion)
- Complexité heuristique d'insertion :
 - n itérations
 - À chaque itération :
 - Nearest insertion : $O(n)$
 - Cheapest insertion : $O(n^2)$

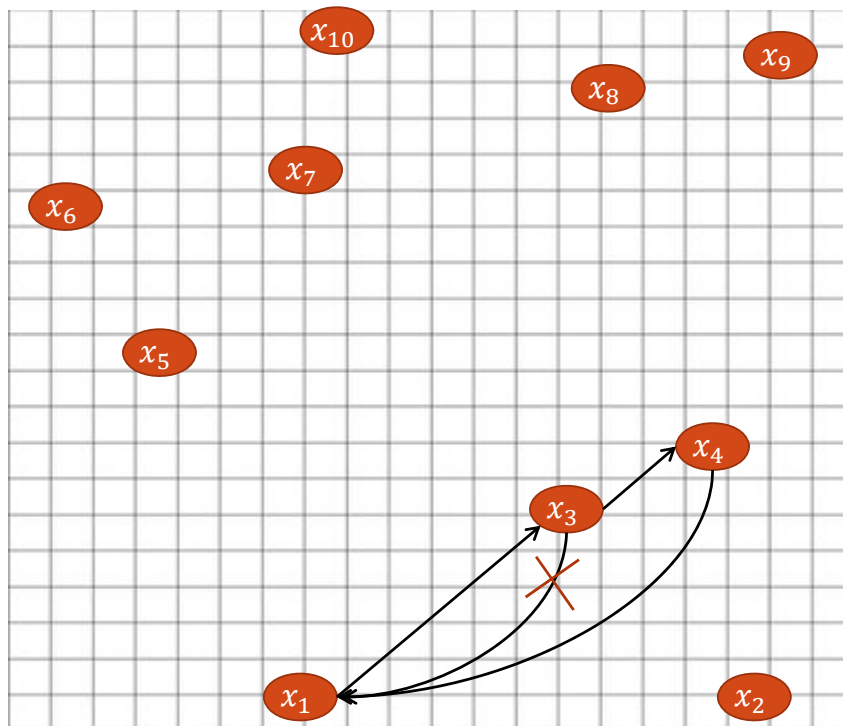
67

Application 1 : TSP



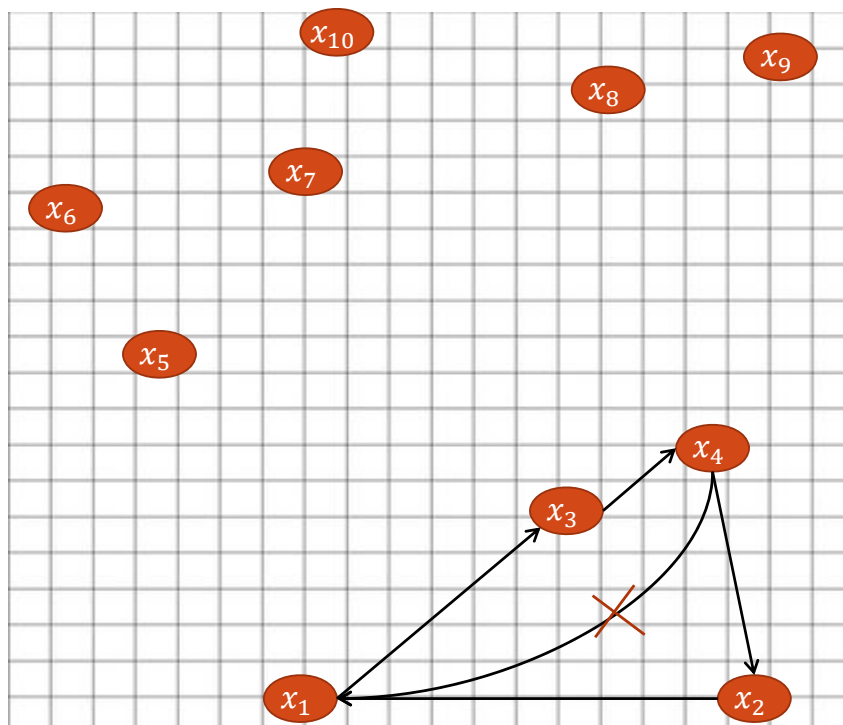
68

Application 1 : TSP



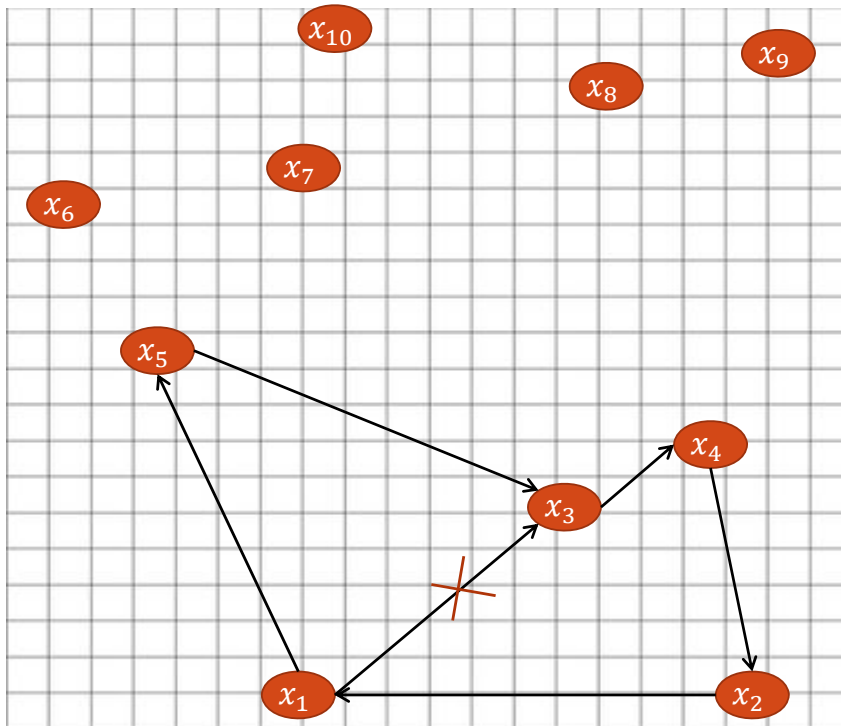
69

Application 1 : TSP



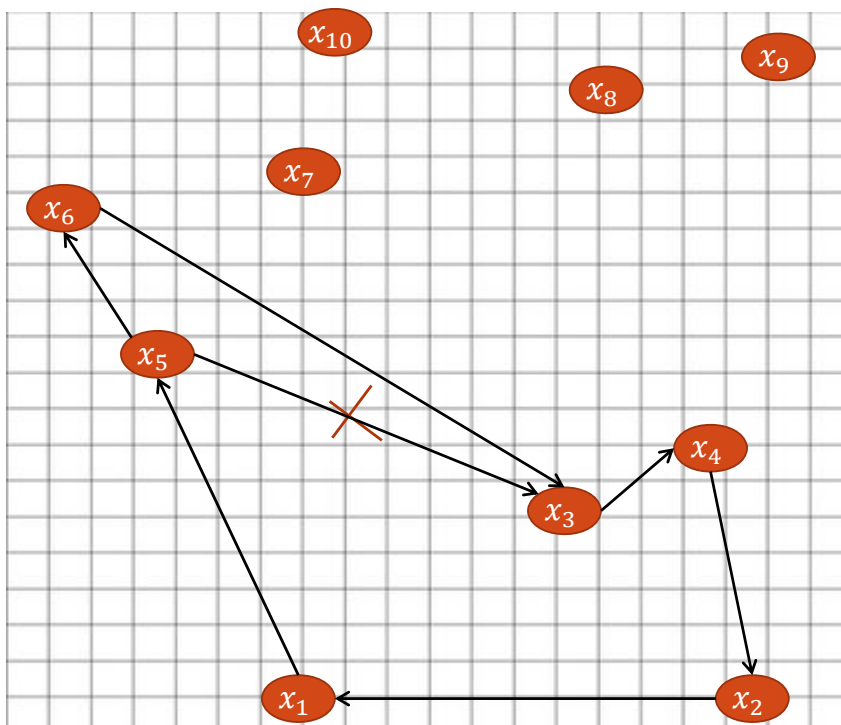
70

Application 1 : TSP



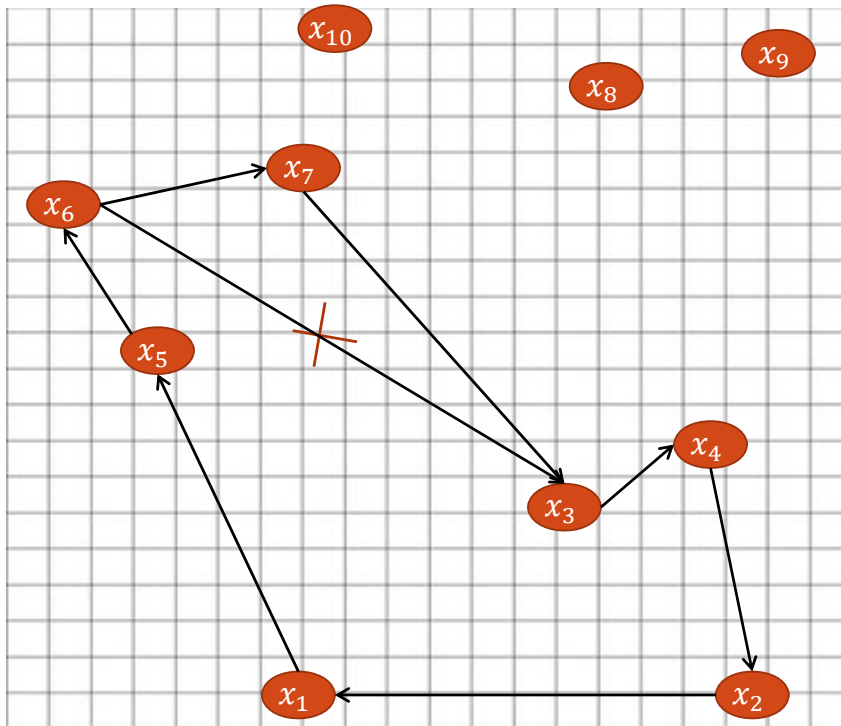
71

Application 1 : TSP



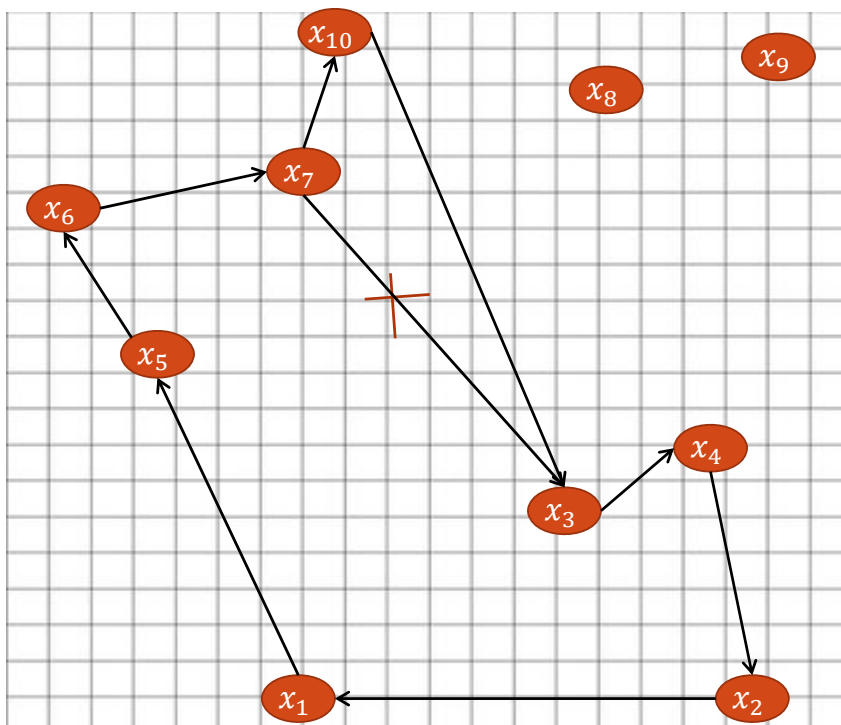
72

Application 1 : TSP



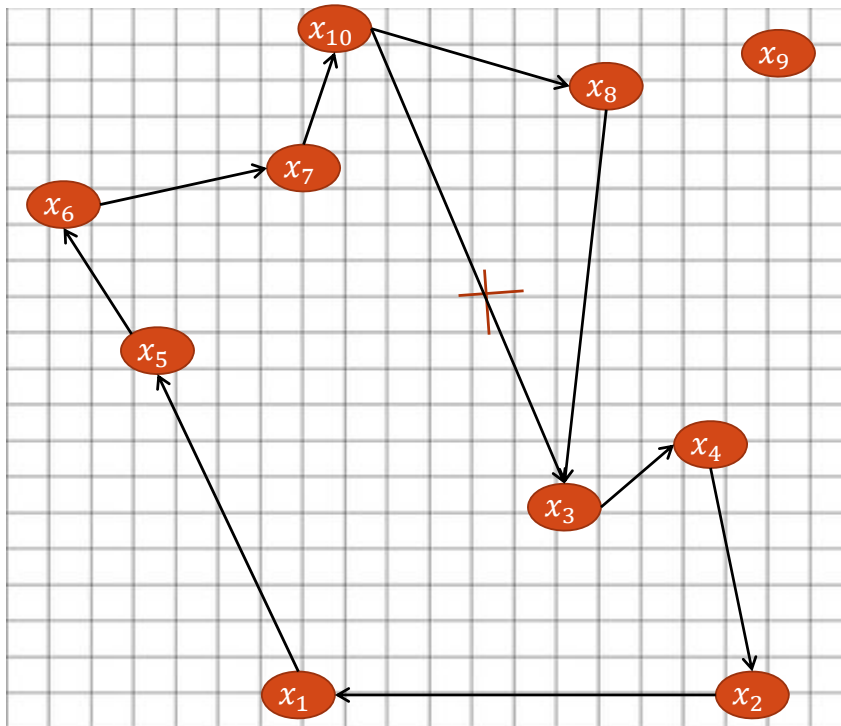
73

Application 1 : TSP



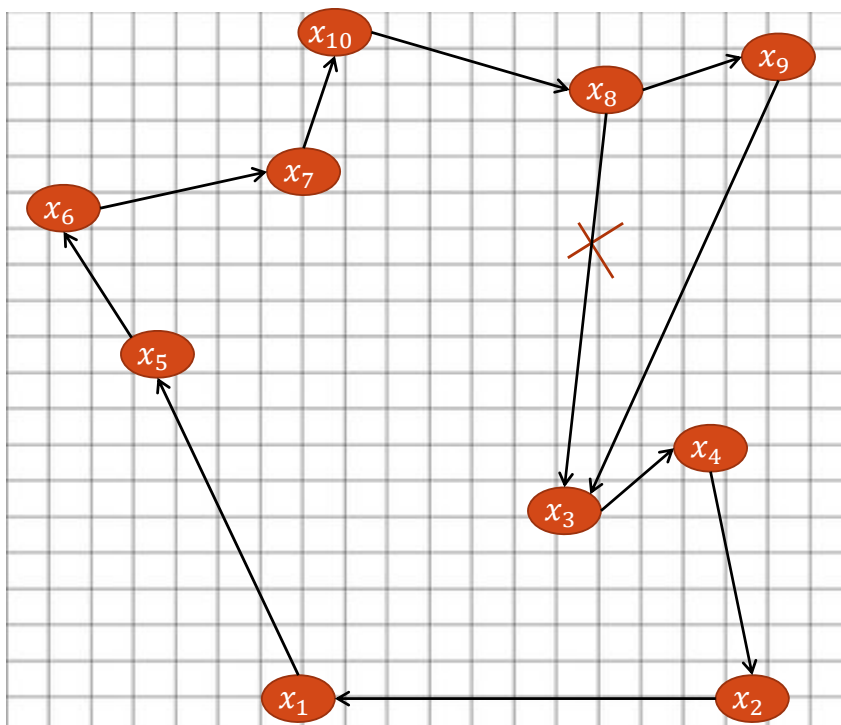
74

Application 1 : TSP



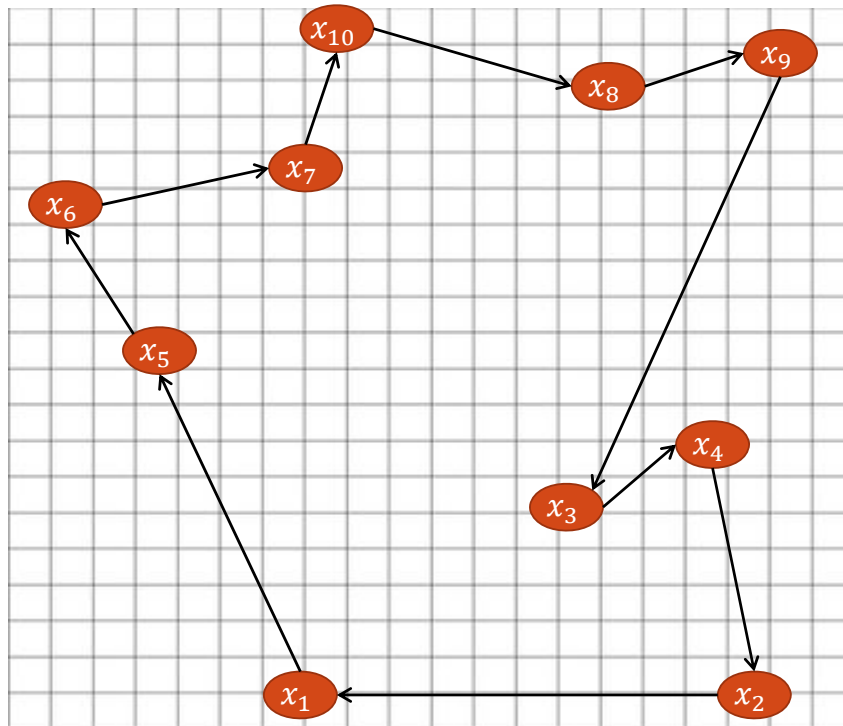
75

Application 1 : TSP



76

Application 1 : TSP



Borne supérieure de la solution optimale

Garantie de qualité (si symétrique et inégalités triangulaires)

77

Application 2 : Problème du Sac à dos (1)

• Problème :

Knapsack Problem

- Un ensemble de n objets $N = \{a_1, \dots, a_n\}$
- A chaque objet :
 - Un poids w_i et une utilité u_i
- Un sac à dos dont le poids total ne doit pas dépasser la capacité W
- Sélectionner les objets pour maximiser l'utilité
- Exemple sac de randonnée de capacité maximale 3 kg
 - Remplir le sac avec les objets les plus utiles

	Utilité	Poids (kg)
carte	10	0,2
gourde	8	1,5
2e gourde	3	1,5
pull	6	1,2
Kway	2	0,5
fromage	2	0,6
fuits secs	4	0,5

78

Exemple 2 : Problème du Sac à dos (2)

• Heuristique par intérêt décroissant

- Calculer pour chaque objet le ratio Utilité / Poids : $\frac{u_i}{w_i}$
- Trier les objets par ordre décroissant sur le ratio $\frac{u_i}{w_i}$
- Variable booléenne : prendre ou pas un objet
- Objets sélectionnés : $S = \emptyset$
 - Parcourir les objets dans l'ordre
 - Soit i l'objet courant
 - Si $w(S) + w_i \leq W$ alors prendre l'objet i : $S = S \cup \{i\}$
- Complexité : $O(n)$
 - Calcul du ratio pour tous les objets ($O(n)$) et tri ($O(\log(n))$)
 - Parcours de chaque objet et vérification du poids total ($O(n)$)

79

Application 2 : Problème du Sac à dos (3)

• Application

	Utilité	Poids	Ratio	Ordre
carte	10	0,2	50,00	1
gourde	8	1,5	5,33	3
2e gourde	3	1,5	2,00	7
pull	6	1,2	5,00	4
Kway	2	0,5	4,00	5
fromage	2	0,6	3,33	6
fruits secs	4	0,5	8,00	2

- $W(S) = 0$
- $i = \text{Carte} : W(S) + w(i) = 0,2 \leq 3 \rightarrow \text{OK}$
- $i = \text{Fruits secs} : W(S) + w(i) = 0,2 + 0,5 = 0,7 \leq 3 \rightarrow \text{OK}$
- $i = \text{Gourde} : W(S) + w(i) = 0,7 + 1,5 = 2,2 \leq 3 \rightarrow \text{OK}$
- $i = \text{Pull} : W(S) + w(i) = 2,2 + 1,5 = 3,7 > 3 \rightarrow \text{NON}$
- $i = \text{Kway} : W(S) + w(i) = 2,2 + 0,5 = 2,7 \leq 3 \rightarrow \text{OK}$
- $i = \text{Fromage} : W(S) + w(i) = 2,7 + 0,6 = 3,3 > 3 \rightarrow \text{NON}$
- $i = \text{2e gourde} : W(S) + w(i) = 2,7 + 1,5 = 4,2 > 3 \rightarrow \text{NON}$

- **Utilité** = $10 + 8 + 2 + 4 = 24$

Borne inférieure / utilité max

80

Exercice : Coloration de sommets

- Planifier sur la semaine les rattrapages de 6 étudiants et de 6 UF avec 2 créneaux par jour tout en minimisant le nombre de créneaux

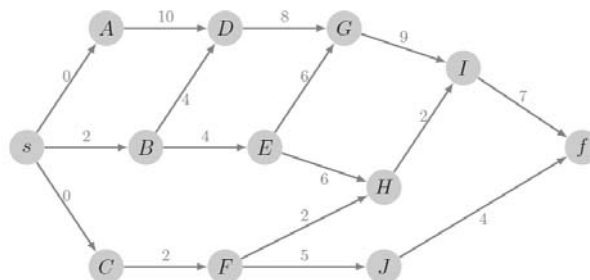
Etudiants	Epreuves
E1	UF1, UF2, UF5
E2	UF3, UF4
E3	UF2, UF6
E4	UF3, UF4, UF5
E5	UF3, UF6
E6	UF1, UF2, UF3

- Pistes :
 - Modéliser le problème par un graphe d'incompatibilité entre UF et se ramener à un problème de coloration de sommets
 - Calculer une borne inférieure du nombre de créneaux
 - Proposer une heuristique gloutonne
 - Voir cours Graphes (3MIC)

81

Exercice : Ordonnancement d'activités (1)

- On souhaite planifier la réalisation d'un ensemble d'activités (notées de A à J) utilisant deux ressources (R_1 et R_2) pour minimiser la durée totale de réalisation (appelée Makespan)
 - Le séquençement temporel entre activités est représenté par le graphe ci-dessous



- Les utilisations de ressources sont les suivantes

	A	B	C	D	E	F	G	H	I	J
R_1 (5)	3	3	1	1	1	2	3	2	1	2
R_2 (1)	0	0	0	1	1	1	0	1	0	0

82

Exercice : Ordonnancement d'activités (2)

- **Question 1.** Calculer la durée minimale de l'ordonnancement (sans prendre en compte les contraintes de ressource)
- **Question 2.** Donner les dates de début au plus tôt et au plus tard
- **Question 3.** Appliquer une heuristique gloutonne en classant les tâches par dates de début au plus tard croissantes
- **Question 4.** Donner le diagramme de Gantt associé et donner la valeur du Makespan de la solution obtenue.

[illegible]

- **Question 5.** A-t-on l'optimum ? Une borne supérieure ? Une borne inférieure ?

83

Récapitulatif – Heuristiques Gloutonnes

- **Caractéristiques d'une méthode gloutonne**
 - Exploiter des connaissances pour faire les meilleurs choix à chaque étape
 - Simple à mettre en œuvre
 - Temps de calcul limité :
 - Ex : complexité linéaire en fonction du nombre de variables
 - Aucune garantie d'optimalité :
 - Borne supérieure/ objectif en minimisation
- **Attention**
 - Peut ne pas aboutir à une solution réalisable
 - Sauf si problème peu contraint

84

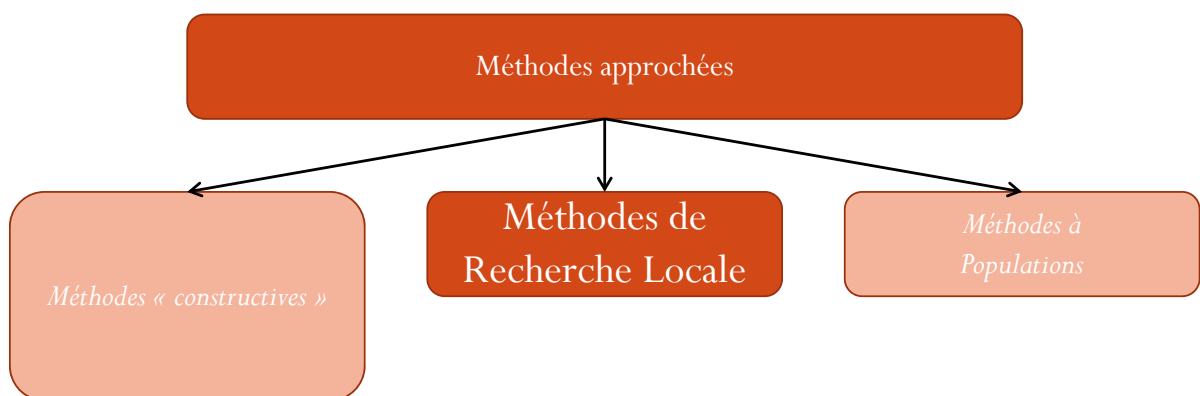
Variantes – Heuristiques Gloutonnes

- **Heuristiques statiques**
 - Ordre d'exploration est défini a priori
- **Heuristiques dynamiques**
 - Ordre d'exploration est recalculé lors de chaque étape
- **Heuristiques avec de l'aléatoire (« randomisées »)**
 - Introduire de l'aléatoire
 - Sur les choix de variables et de valeurs
 - Lancer plusieurs exécutions de la recherche gloutonne
 - Arrêt de la méthode
 - Toutes les variables sont instanciées
 - Récupérer la meilleure solution

85

Section 3. Méthodes approchées

- **Méthodes de Recherche Locale**



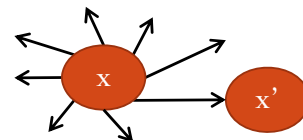
LA SUITE ...

- **A finaliser ...**
 - Voisinage / Recherche Locale
 - Recuit Simulé
 - Tabou
 - Algorithme génétique
 - Colonies de Fourmis
- Hybridation de méthodes
- Contexte multi-objectif ? Contexte multi-agent ?

87

Principes d'une recherche locale (1)

- **Idée :**
 - Les bonnes solutions ont des caractéristiques communes
- **Principe :**
 - Partir d'une solution initiale
 - Modification de cette solution
 - Mouvement
 - Ex : Echange de 2 sommets dans un cycle
 - Voisinage d'une solution x
 - $N(x)$: Ensemble des solutions atteignables par un mouvement donné
 - $x' \in N(x)$: voisin de la solution x

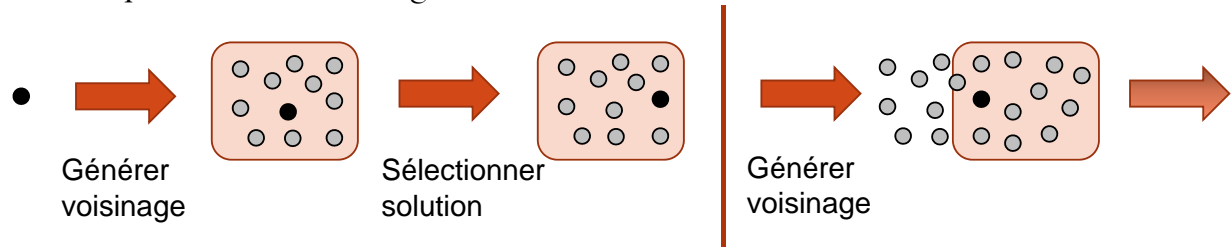


88

Principes d'une recherche locale (2)

- **Principe d'exploration :**

- Exploration de voisinages successifs



- But : améliorer la valeur de la fonction objectif (et assurer l'obtention de solution réalisable)
- Solution obtenue :
 - Issue de mouvements dans une structure de voisinage
- Conditions d'arrêt ?
- Cycles entre différentes solutions ?

89

Principes d'une recherche locale (3)

- **Composants**

- Espace de recherche des solutions : E
- Fonction objectif : f (cout à minimiser)
- Voisinage : mouvement qui pour tout $x \in E \rightarrow N(x) \in E$

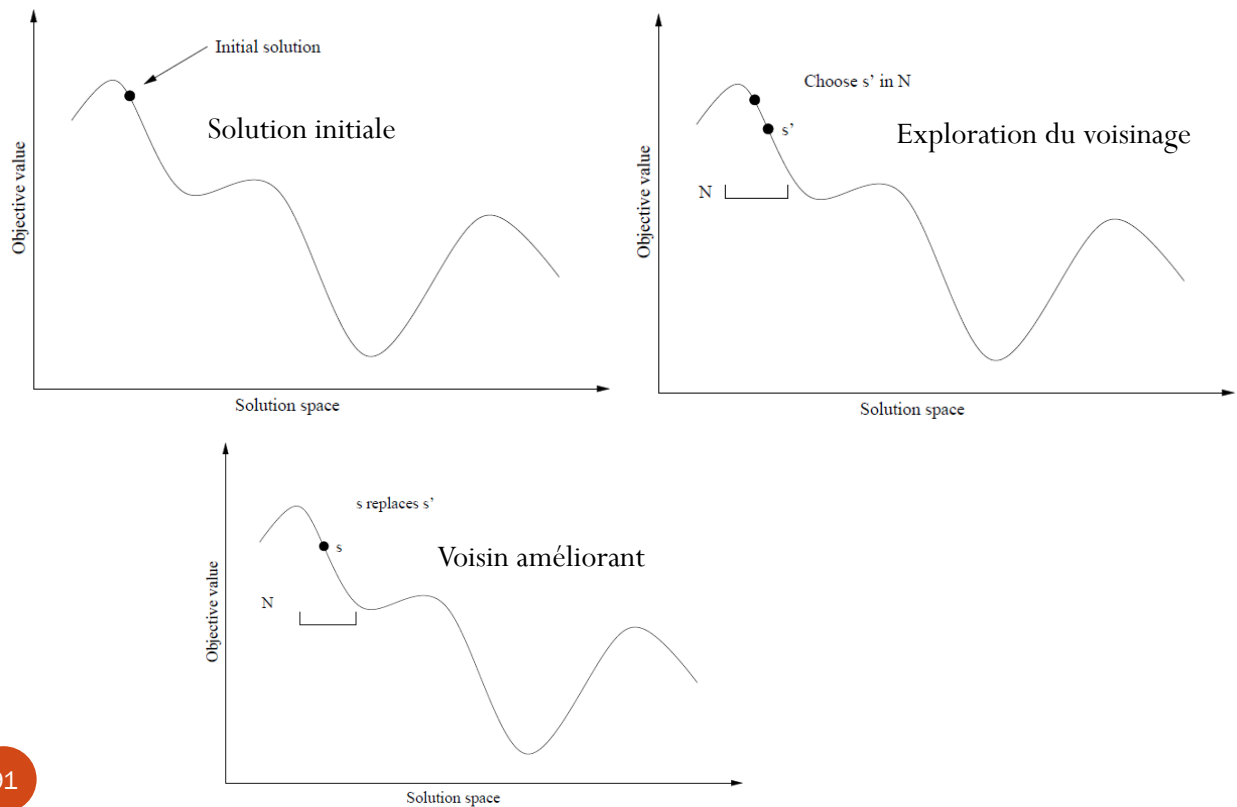
- **Algorithme de Recherche locale**

- Solution initiale : $s \in E$; $z \leftarrow f(s)$
- Meilleure solution : $s^* \leftarrow s$; $z^* \leftarrow z$
- Répéter
 - Choisir $s' \in N(s)$
 - $s \leftarrow s'$
 - Si $f(s) < f(s^*)$ alors $s^* \leftarrow s$, $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt »

Local Search

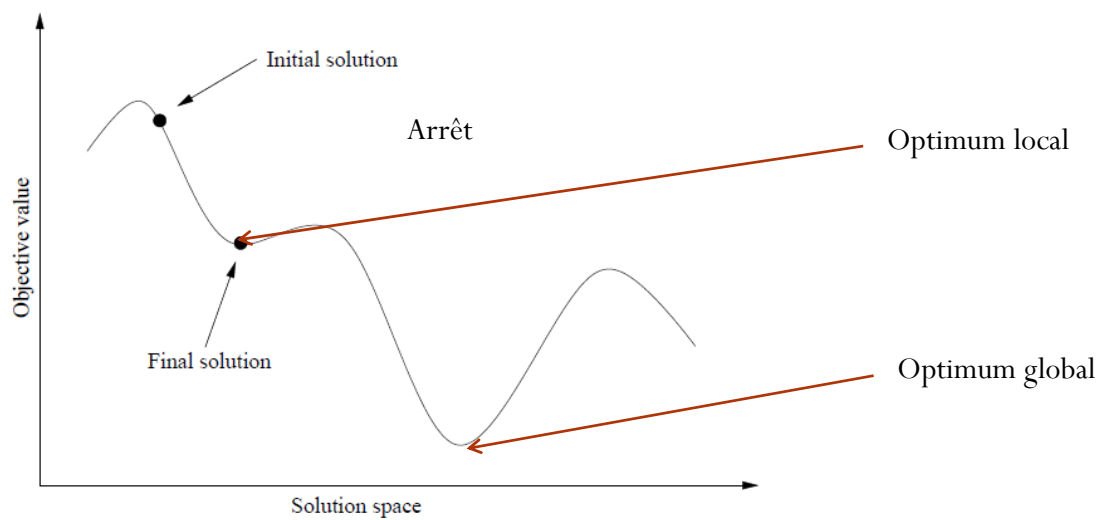
90

Exemple



91

Exemple



92

Algorithme de recherche locale (1)

- **Principe :**

- Solution initiale : $s \in E$; $z \leftarrow f(s)$ (1)
- Meilleure solution : $s^* \leftarrow s$; $z^* \leftarrow z$
- Répéter
 - Choisir $s' \in N(s)$ (2) (3)
 - $s \leftarrow s'$
 - Si $f(s) < f(s^*)$ alors $s^* \leftarrow s$, $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt » (4)

- **Points clés**

1. Comment obtenir une solution initiale ?
2. Comment générer un voisinage ?
3. Comment sélectionner la prochaine solution ?
4. Quand s'arrêter ?

Algorithme de recherche locale (2)

- **Algorithme de Recherche locale**

- Solution initiale : $s \in E$; $z \leftarrow f(s)$ (1)
- Meilleure solution : $s^* \leftarrow s$; $z^* \leftarrow z$
- Répéter
 - Choisir $s' \in N(s)$ (2) (3)
 - $s \leftarrow s'$
 - Si $f(s) < f(s^*)$ alors $s^* \leftarrow s$, $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt » (4)

- **Points clés**

1. **Comment obtenir une solution initiale ?**
 - Solution aléatoire
 - Solution connue
 - Heuristique gloutonne

Algorithme de recherche locale (3)

- **Algorithme de Recherche locale**

- Solution initiale : $s \in E$; $z \leftarrow f(s)$ (1)
- Meilleure solution : $s^* \leftarrow s$; $z^* \leftarrow z$
- Répéter
 - Choisir $s' \in N(s)$ (2) (3)
 - $s \leftarrow s'$
 - Si $f(s) < f(s^*)$ alors $s^* \leftarrow s$, $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt » (4)

- **Points clés**

- 4. **Quand s'arrêter ?**

- Obtention d'une solution de qualité voulue
 - Nombre d'itérations / Temps d'exécution maximum atteint
 - Nombre d'itérations / Temps d'exécution maximum sans amélioration atteint

Algorithme de recherche locale (4)

- **Algorithme de Recherche locale**

- Solution initiale : $s \in E$; $z \leftarrow f(s)$ (1)
- Meilleure solution : $s^* \leftarrow s$; $z^* \leftarrow z$
- Répéter
 - Choisir $s' \in N(s)$ (2) (3)
 - $s \leftarrow s'$
 - Si $f(s) < f(s^*)$ alors $s^* \leftarrow s$, $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt » (4)

- **Points clés**

- 2. **Comment générer un voisinage ?**

Voisinages (1)

- **Voisinage d'une solution**

- Opération de transformation : $x \in E \rightarrow N(x) \in E$
- Transformation locale sur une solution :
 - La structure globale de la solution n'est pas modifiée
- Doit pouvoir être généré et évalué rapidement
- Lié à la représentation d'une solution

- **Graphe de voisinage :**

- Sommets : espace de recherche E
- Si $y \in N(x) \rightarrow$ relation (x, y) dans le graphe

Voisinages (2)

- **Propriétés :**

- Accessibilité : tout élément de l'espace de recherche peut être atteint (toute solution intéressante) est accessible par transformations successives
 - Graphe de voisinage Connexe
- Réversibilité : on peut revenir à la solution initiale
 - Graphe symétrique

- **Choix d'un Voisinage :**

- Lié au codage de la solution
 - Ex : Voyageur de Commerce : ensemble des sommets ou des arêtes
- Lié à sa taille (cout de génération et d'évaluation)
 - Ex : $O(n)$, $O(n^2)$, ... / nombre de variables

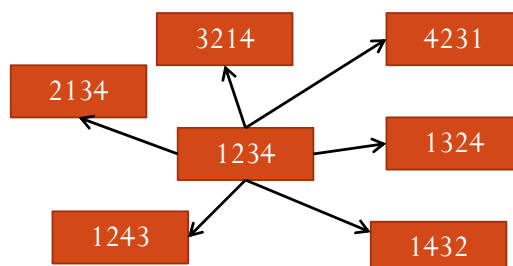
Exemple de voisinage (1)

- Ensemble de N variables binaires :
- Voisinage :
 - Distance :
 - Ensemble des chaînes binaires à une distance de Hamming de 1
 - Pour 5 variables binaires : $x = 01101$
 - $N(x) = \{11101; 00101; 01001; 01111; 01100\}$
 - $|N(x)| = n = 5$
 - Si distance de Hamming de 2, $|N(x)| = \frac{n(n-1)}{2}$
 - Complémentation
 - Solution = chaîne de variables binaires :
 - remplacer une ou plusieurs valeurs 0/1 par son complémentaire

99

Exemple de voisinage (2)

- Echange / Swap
 - Solution = chaîne de caractères
 - Choisir 2 positions i et j . Intervertir les caractères situés à ces deux positions



Taille du voisinage d'une solution de taille n : $\frac{n(n-1)}{2}$

100

Exemple de voisinage (3)

- Quelques transformations classiques :

- **Décalage**

- Solution = chaîne de caractères :
- Choisir une position i . Insérer élément de position i à la fin et décaler les caractères



- **Inversion**

- Solution = chaîne de caractères
- Choisir 2 positions i et j . Inverser l'ordre d'écriture entre i et j .



101

Voisinages pour le problème du Voyageur de Commerce (1)

- **Problème :**

- Un ensemble de villes (sommets d'un graphe) : $G(X, A)$ et distance entre villes
- Déterminer une tournée minimisant la somme des distances parcourues

- **Espace de recherche E** : ensemble des permutations de X

- Soit une solution contenant les arêtes (u, x) et (v, y)

- **Insertion**

- Supprimer u du cycle pour l'insérer après v
- Déplacer (u, x) après v

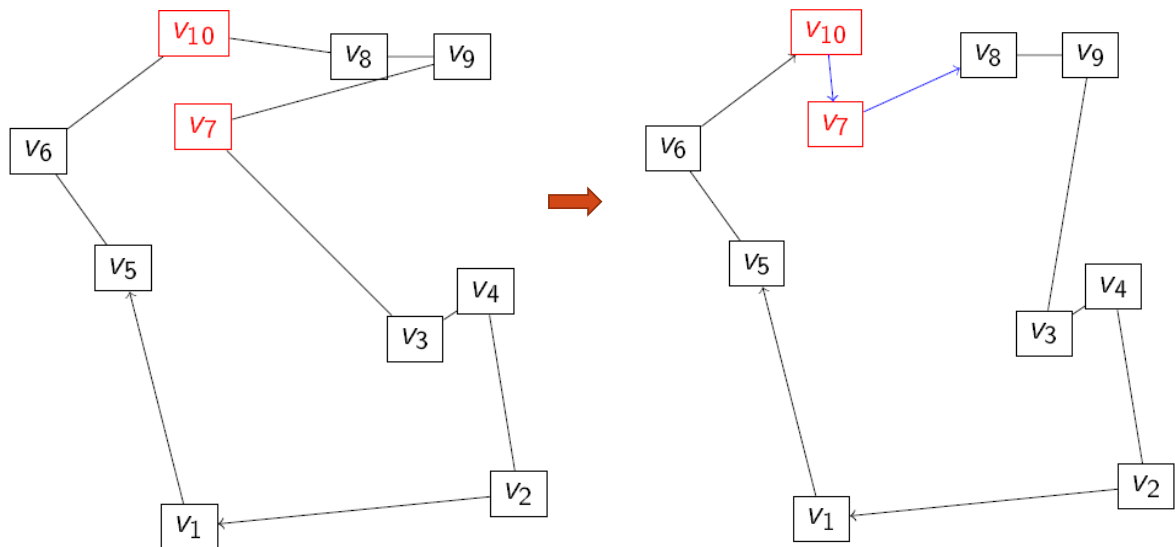
- **Swap**

- Echanger u et v

102

Voisinages pour le problème du Voyageur de Commerce (2) : Insertion

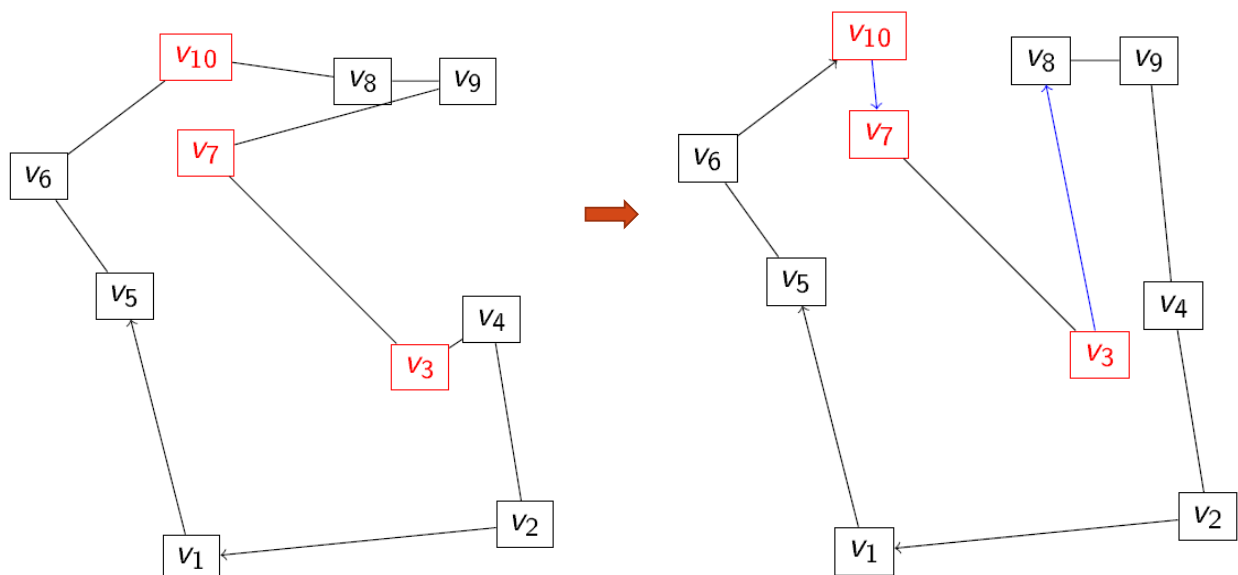
- Supprimer v_7 pour l'insérer après v_{10}



103

Voisinages pour le problème du Voyageur de Commerce (2) : Insertion

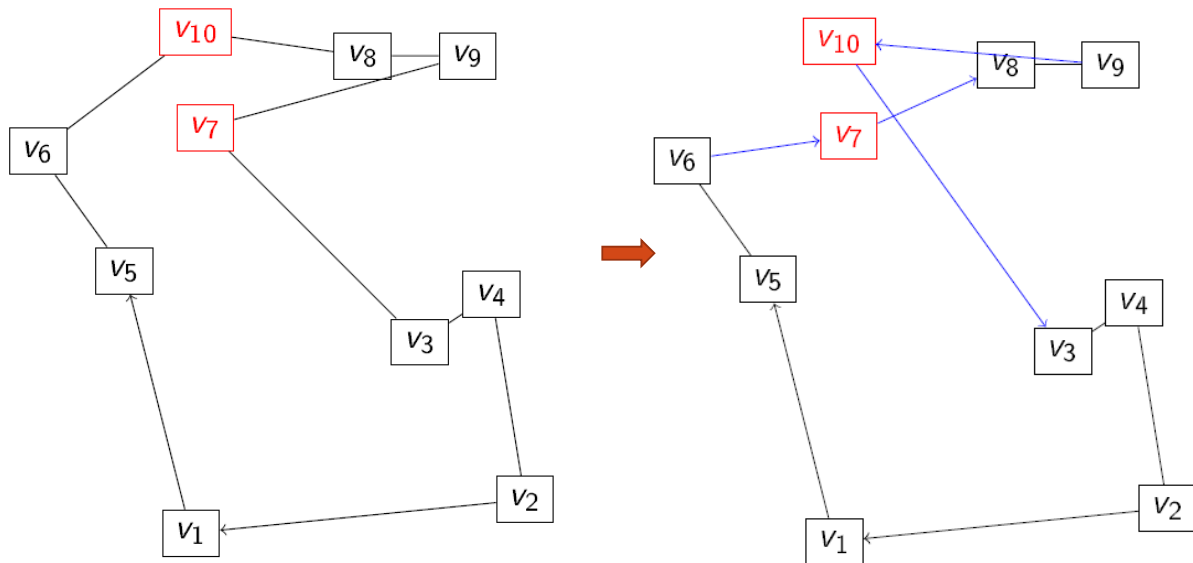
- Déplacer (v_7, v_3) pour le placer après v_{10}



104

Voisinages pour le problème du Voyageur de Commerce (3) : Swap

- Echanger v_7 et v_{10}

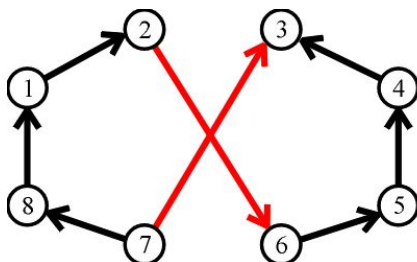


105

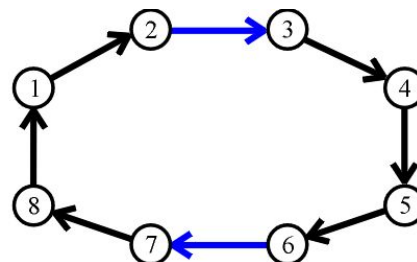
Voisinages pour le problème du Voyageur de Commerce (3) :

- Voisinage 2-opt**
 - Trouver 2 arêtes non consécutives
 - Les supprimer et reformer le cycle

- Supprimer : (x_2, x_6) et (x_7, x_3)



- Reformer : (x_2, x_3) et (x_6, x_7)



- Une seule façon de reformer le cycle

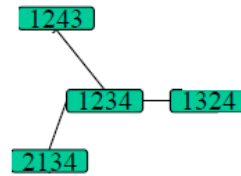
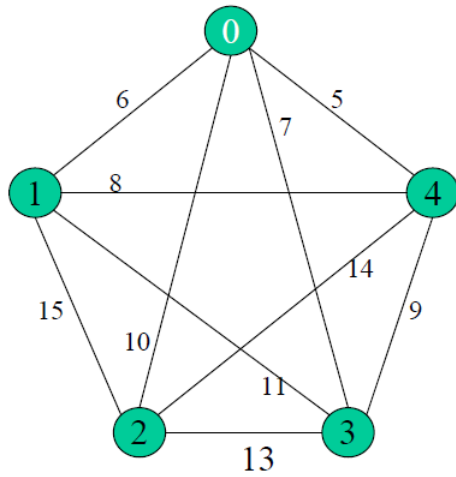
- Variante : 3-opt, k-opt,

106

Graphe de voisinage (1)

- Exemple sur le TSP

Voisinage = échanger 2 sommets consécutifs



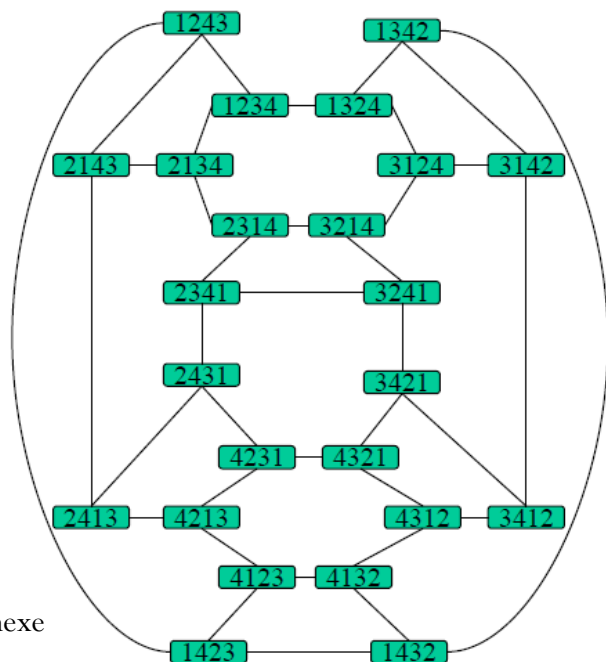
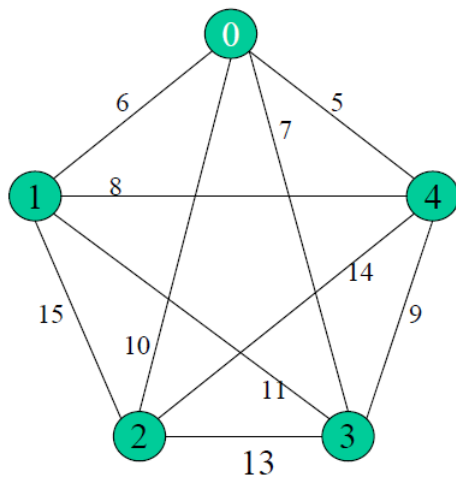
107

Graphe de voisinage (2)

- Exemple sur le TSP

Voisinage = échanger 2 sommets consécutifs

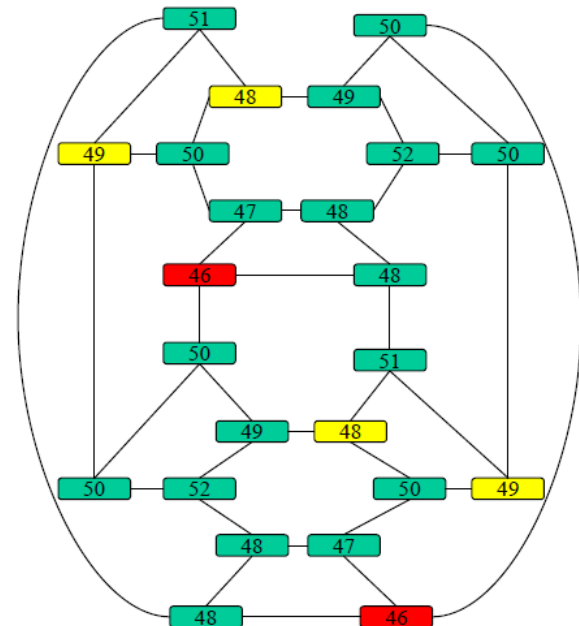
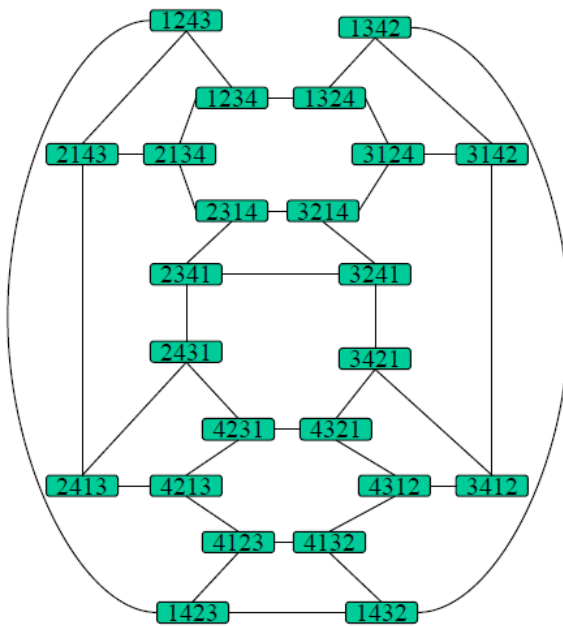
Départ et Retour en 0



108

Voisinage → graphe connexe

Graphe de voisinage (3)



Optimums locaux et globaux

109

Paysage de Recherche

- Une opération de voisinage → graphe de voisinage
 - Paysage de recherche
- **Optimum local** : tous les voisins sont moins bons / objectif
- **Plateau** : ensemble de points connexes dans le graphe de voisinage et ayant même valeur de fonction objectif
- **Bassin d'attraction** : voisins que l'on peut atteindre sans dégrader la fonction objectif

110

Choix d'un voisinage

- Dépend du problème à résoudre
- Dépend de la représentation des solutions
- Impact du voisinage sur le paysage des solutions
- Littérature sur le problème considéré
- **Attention à la taille du voisinage**
 - Si trop petit : risque de ne pas avoir de meilleure solution
 - Si trop grand : l'exploration est coûteuse
- **Impact entre voisinage et sélection d'une solution :**
 - 1 seul optimum et pas de plateau : Intensifier
 - Paysages rugueux : Diversifier



111

Algorithme de recherche locale (5)

- **Algorithme de Recherche locale**
 - Solution initiale : $s \in E$; $z \leftarrow f(s)$ (1)
 - Meilleure solution : $s^* \leftarrow s$; $z^* \leftarrow z$
 - Répéter
 - Choisir $s' \in N(s)$ (2) (3)
 - $s \leftarrow s'$
 - Si $f(s) < f(s^*)$ alors $s^* \leftarrow s$, $z^* \leftarrow z$
 - Jusqu'à « Critères d'arrêt » (4)
- **Points clés**
 2. **Comment sélectionner la prochaine solution ?**

112

Exploration d'un voisinage (1)

- **Algorithme de Recherche locale**

- Solution initiale : $s \in E$; $z \leftarrow f(s)$ (1)
- Meilleure solution : $s^* \leftarrow s$; $z^* \leftarrow z$
- Répéter
 - Choisir $s' \in N(s)$ (2) (3)
 - $s \leftarrow s'$
 - Si $f(s) < f(s^*)$ alors $s^* \leftarrow s$, $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt » (4)

- **Points clés**

- 3. **Comment sélectionner la prochaine solution ?**

- Premier voisin améliorant
 - Meilleur voisin améliorant
 - Aléatoire
 -

113

Exploration d'un voisinage (2)

- **Comment sélectionner la prochaine solution ?**

- Exploration de l'espace de recherche pour déterminer la nouvelle solution
- Sélectionner une solution de meilleure qualité :
 - **Premier voisin améliorant (First Improvement)**
 - Savoir générer des voisins prometteurs
 - **Meilleur voisin (Best improvement)**
 - Exploration de tout le voisinage
- Méthode **Hill-Climbing** ou **Plus grande pente** ou **Descente**
 - Méthode du gradient en optimisation continue
- Permet une **intensification** de l'exploration autour d'une solution initiale
- Aboutit à un optimum local (ou reste bloqué sur un plateau)

114

Algorithmes Hill-Climbing (1)

Algorithme First Improvement

Solution initiale : $s \in E$; $z \leftarrow f(s)$ (1)
Meilleure solution : $s^* \leftarrow s$; $z^* \leftarrow z$
Répéter
 Choisir $s' \in N(s)$ tq $f(s') < f(s)$ (2) (3)
 $s \leftarrow s'$
 Si $f(s) < f(s^*)$ alors $s^* \leftarrow s$, $z^* \leftarrow z$
Jusqu'à « Critères d'arrêt » (4)

Algorithme Best Improvement

Solution initiale : $s \in E$; $z \leftarrow f(s)$ (1)
Meilleure solution : $s^* \leftarrow s$; $z^* \leftarrow z$
Répéter
 Choisir $s' \in N(s)$ tq $\forall s'' \in N(s) \setminus \{s'\} f(s') < f(s'')$ (2) (3)
 $s \leftarrow s'$
 Si $f(s) < f(s^*)$ alors $s^* \leftarrow s$, $z^* \leftarrow z$
Jusqu'à « Critères d'arrêt » (4)

115

Algorithmes Hill-Climbing (2)

• Terminaison de la méthode Hill-Climbing

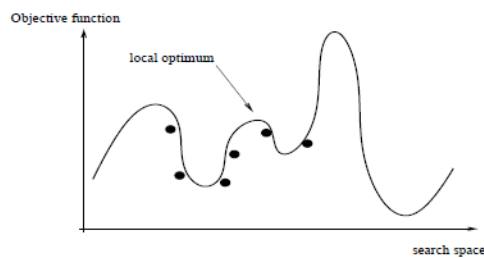
- La taille du voisinage ne permet pas de trouver de meilleure solution
- Solution localement optimale
- Pas de bouclage de la méthode
 - Pas de retour sur une solution déjà trouvée
 - Quand on sélectionne s' comme solution améliorante, s' n'a jamais été sélectionnée auparavant comme solution améliorante

116

Exploration aléatoire d'un voisinage

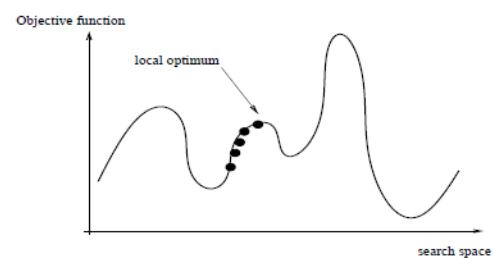
- Sélectionner une solution aléatoire: **Random Walk**
 - Accepte des solutions même non améliorante
 - Permet une **diversification** de l'exploration (sortir d'un optimum local)
- Exemple en maximisation

Random walk



Diversification

Hill-climbing



Intensification

117

Intensification / Diversification

- **Combinaison intensification / diversification**
 - Fixer une probabilité de choisir un mouvement « random » : ρ
 - Choisir aléatoirement une valeur m
 - Si $m < \rho$ alors
 - Choisir une solution s' aléatoirement dans $N(s)$ -- **diversification**
 - Sinon
 - Choisir la meilleure solution $s' \in N(s)$ -- **intensification**
 - Selon la probabilité ρ : compromis diversification / intensification
 - $\rho = 1$: random walk
 - $\rho = 0$: hill-climbing
 - En pratique : ρ faible

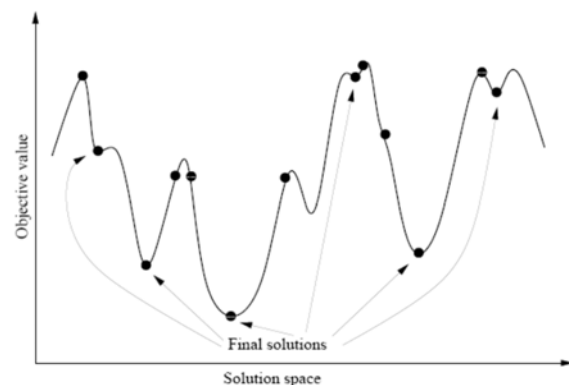
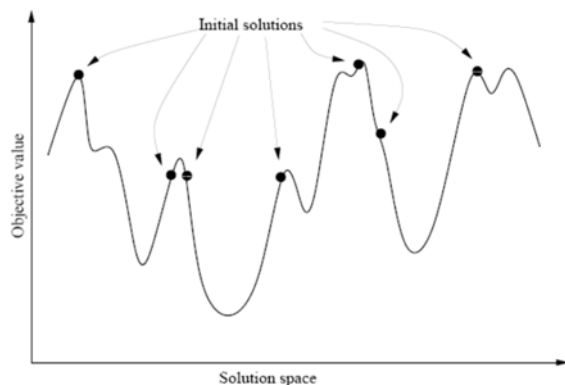
118

Plusieurs solutions initiales

- **Autre amélioration de la méthode Hill-Climbing**

- Relancer la méthode de descente à partir de plusieurs solutions initiales
- Conserver la meilleure solution trouvée

- Méthode de **Descente Multi-Start**



119

Algorithme de recherche locale (6)

- **Algorithme de Recherche locale**

- Solution initiale : $s \in E$; $z \leftarrow f(s)$ (1)
- Meilleure solution : $s^* \leftarrow s$; $z^* \leftarrow z$
- Répéter
 - Choisir $s' \in N(s)$ (2) (3)
 - $s \leftarrow s'$
 - Si $f(s) < f(s^*)$ alors $s^* \leftarrow s$, $z^* \leftarrow z$
- Jusqu'à « Critères d'arrêt » (4)

- **Points clés**

1. Comment obtenir une solution initiale ?
2. Comment générer un voisinage ? ➔ **et évaluer**
3. Comment sélectionner la prochaine solution ?
4. Quand s'arrêter ?

120

Evaluation d'une solution voisine

- **Evaluation d'une solution / d'un voisin**

- Estimation **a priori** de la qualité d'un voisin
- Permet de se guider dans l'espace de recherche pour trouver un chemin vers de bonnes solutions

- Comment évaluer ?

- Fonction Objectif
- Mais pas toujours !
 - autre fonction d'évaluation (ex SAT : nombre de clauses non vérifiées)

- **L'évaluation : opération de calcul très fréquente**

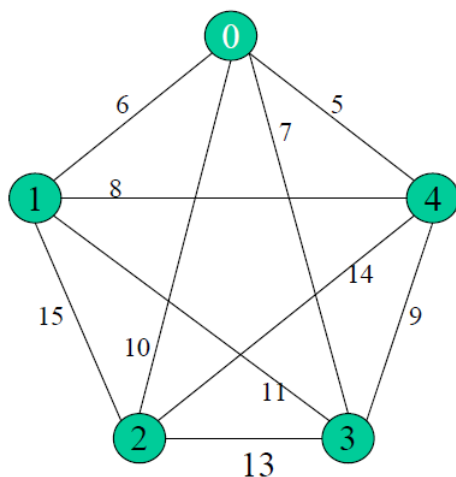
- Doit être la moins coûteuse en temps de calcul
- Si possible incrémentale :
 - Calculer l'apport / la perte générée par un voisin
 - Variation de $f(s')$ par rapport à $f(s)$

121

Exemple évaluation

- **Exemple**

Départ et Retour en 0



Solution $s = \{0,1,2,3,4,0\}$; $f(s) = 48$

Voisin s'

- Permutation 1 et 2 : $\{0,2,1,3,4\}$

Evaluation de s' :

- Retirer (0,1) et (2,3) : $-6-13 = -19$
- Ajouter (0,2) et (1,3) : $+10+11=+21$
- $f(s') = f(s) - 19 + 21 = 50$

122

S'échapper des optima locaux (1)

- **Dans une recherche locale :**
 - Impact du choix de la structure de voisinage
 - Impact du choix de la fonction d'évaluation (ie. du paysage parcouru)
- **Pour diversifier**
 - Introduire de l'aléatoire
 - Autoriser des solutions non améliorantes
 - Mais comment éviter de cycler sur des solutions ?
 - Faire varier les voisinages
 - Re-démarrage
 - Mémorisation
- **Nombreuses variantes en Recherche Locale**

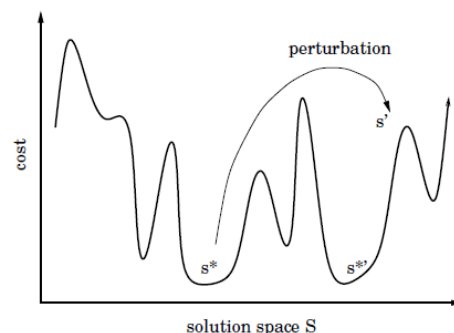
123

Variantes (1) : Iterated Local Search

Perturbation aléatoire de la solution

Iterated Local Search

1. $s_0 \leftarrow$ Solution initiale; Best $\leftarrow s_0$
2. $s \leftarrow$ Descent(s_0) // *Exploration*
3. repeat
4. $s' \leftarrow$ Perturbation (s)
5. $s'' \leftarrow$ Descent(s')
6. Acceptation :
 - if $f(s'') < f(s)$ then
 - $s \leftarrow s''$; Mise-à-jour(Best, s);
7. until : conditions à définir



124

Variantes (2) : Méthode à seuil

Threshold Accepting

- **Idée :**

- Introduire un seuil d'acceptation τ des solutions non améliorantes
- Choisir le premier voisin s' tel que $f(s') - f(s) < \tau$

- **Réglage du seuil**

- Détermine le compromis diversification / intensification
 - Si $\tau = \infty$: aléatoire
 - Si $\tau \leq 0$: les mouvements dégradant la solution sont interdits
- Le seuil peut varier au cours des itérations

125

Variantes (3) : Voisinages variables

- **Variable Neighborhood Descent/ Search**

- **Définir plusieurs voisinages (diversification)**

- But : pouvoir sortir des optima locaux et améliorer la qualité de la solution

- **Principe**

- Effectuer une succession de méthodes de descente
- Quand un optimum local est atteint par une méthode de descente:
 - changer de voisinage
- Ordonner a priori l'ensemble des voisinages
 - N_1, N_2, \dots, N_k
 - Complexité croissante

126

Variantes (3) : Voisinages variables

- Principe

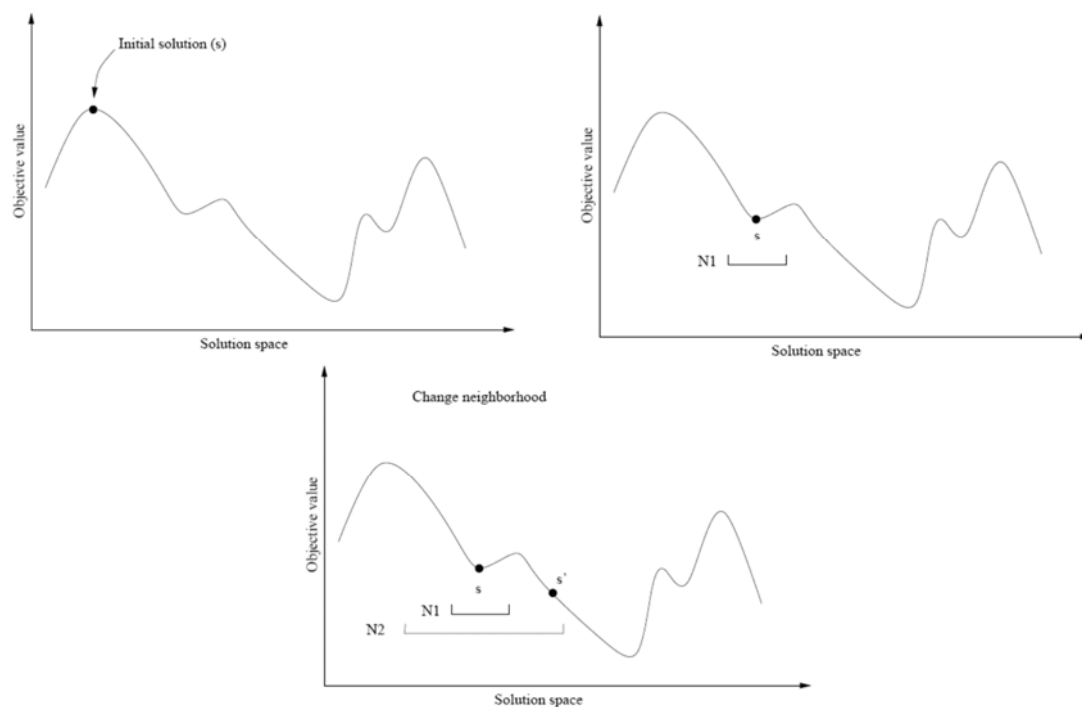
```

1. s ← Solution initiale
2. i ← 1 : numéro du voisinage
3. repeat
4.     s' ← Meilleur Voisin dans Ni(s) //Variante : Random(Ni)
                                         // et Descent(s')
5.     if f(s') < f(s) then                //Variante : Rester sur Ni
        i ← 1;      s ← s'      // et appliquer Descent(s')
6.     else
        i ← i+1
    end if
7. until i > k (nombre de voisinages)

```

127

Variantes (3) : Voisinages variables



128

Variantes (4) : Recuit Simulé

Simulated Annealing

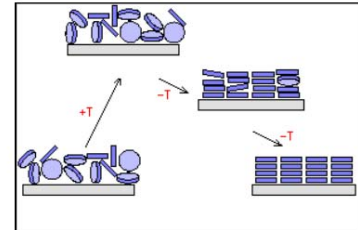
- **Idée**

- Analogie métallurgie : en chauffant un métal puis en le refroidissant doucement on peut obtenir des structures cristallines résistantes

Principe

Repose sur la capacité d'un système physique d'évoluer vers un état énergétique minimal.

- D'abord, une agitation thermique permet de sortir des minima locaux de l'énergie.
- Ensuite, quand on refroidit le système physique assez lentement, il tend à évoluer vers une structure d'énergie minimale.



- Métaheuristique pour l'optimisation
 - S. Kirkpatrick 1983 / V. Cerny 1985

129

Variantes (4) : Recuit Simulé

Simulated Annealing

- **Pour l'optimisation :**

- Diversifier la recherche en acceptant des voisins qui dégradent la fonction objectif en fonction d'une probabilité d'acceptation qui décroît dans le temps

- **Stratégie d'exploration :**

- **Paramètre T (température)** qui décroît au cours des itérations
- Choix d'un voisin s' tel que :
 - Si $\Delta = f(s') - f(s) < 0$ alors $s \leftarrow s'$ -- **intensification**
 - Si $x < e^{\frac{-\Delta}{T}}$ (x choisi aléatoirement dans $[0,1]$) alors $s \leftarrow s'$
 - Sinon Rester sur la solution s -- **diversification**

130

Variantes (4) : Recuit Simulé

- **Température** : probabilité d'accepter une solution non améliorante
- **Condition de Métropolis** :

- accepter la nouvelle solution avec une probabilité : $e^{\frac{-\Delta}{T}}$

- Si $\Delta \nearrow$ alors $e^{\frac{-\Delta}{T}} \searrow$; si $T \searrow$ alors $e^{\frac{-\Delta}{T}} \searrow$

- **Algorithme**

```
1. s0 ← solution initiale
2. T0 ← Température initiale
3. s ← s0; T ← T0
4. while (Conditions) loop
5.     s' ← Random(N(s))
6.     Deltaf ← f(s') - f(s)
7.     if Deltaf < 0 ou random < exp(-Delta/T) then
8.         s ← s'
9.     end if
10.    T ← k.T
11. end while
12. Return s
```

131

Variantes (4) : Recuit Simulé

- **Au début des itérations** :
 - T élevé : Acceptation fréquente de solutions non améliorantes
- **En fin des itérations** :
 - T faible : acceptation rare de solution non améliorante
- **Réglage des paramètres**
 - Température initiale
 - Variation de température : à chaque étape / par palier / adaptative
 - Conditions d'arrêt (température, fonction objectif, ...)
 - Trouver le bon paramétrage

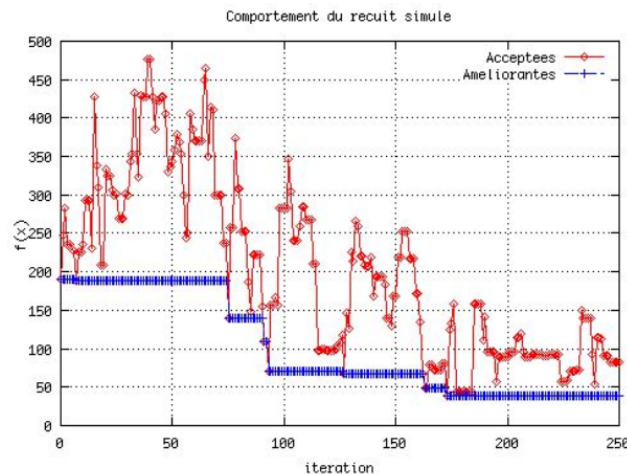
132

Variantes (4) : Recuit Simulé

- **Etat initial**

- Solution initiale
- Température : doit permettre d'accepter « suffisamment » de solutions voisines

- **Graphique des solutions visitées / acceptées**

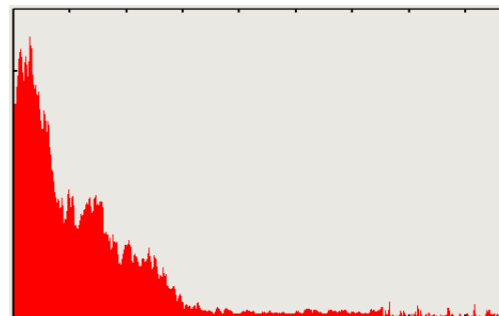
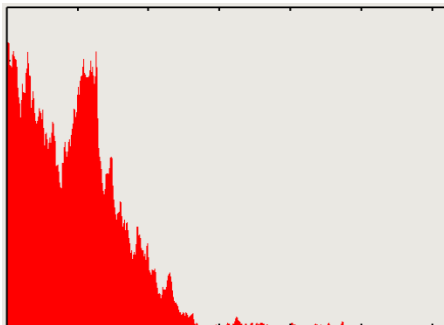


133

Variantes (4) : Recuit Simulé

- **Schéma de refroidissement**

- Si trop rapide : convergence prématurée : on reste dans un optimum local
- Si trop lente : exploration trop importante



134

Variantes (5) : Recherche Tabou

Tabu Search

- **Constat :**

- Quand on est sur un optimum local : les solutions voisines sont toutes de moins bonne qualité → bassin d'attraction
- Glover 1986 / Hansen 1986

- **Idée :**

- Sortir du bassin d'attraction en acceptant des solutions de moins bonne qualité
 - Choisir le meilleur voisin même si non améliorant
- Mais interdire de revisiter des solutions déjà explorées
- Structure pour mémoriser des informations sur les solutions visitées, appelée **Liste Tabou** pendant un certain nombre d'itérations

135

Variantes (5) : Recherche Tabou

- **Stratégie d'exploration :**

- Introduire une liste L (initialement vide)
- A chaque itération : ajouter le dernier mouvement effectué dans L
- Choisir une solution voisine s' telle que :
 - Le mouvement $s \rightarrow s' \notin L$ -- **diversification**
 - Le cout $f(s')$ soit minimal -- **intensification**

- **Algorithme**

```
1.  $s \leftarrow$  solution initiale
2.  $best \leftarrow s$ 
3.  $L \leftarrow \emptyset$  // Tabu
4. while (Conditions) loop
5.    $s' \leftarrow$  Meilleur-Voisin( $N(s)$ ,  $L$ ) // Voisin non tabou
6.   if  $f(s') < f(best)$  then
7.      $best \leftarrow s'$ 
8.   end if
9.   Actu_Tabu( $s'$ ,  $L$ )
10. end while
11. Return best
```

136

Variantes (5) : Recherche Tabou

- **Liste Tabou**

- **Conserver les mouvements effectués** et non pas les solutions visités
 - Exemple : variables échangées (swap)
 - Plus rapide à vérifier et moins coûteux à mémoriser ...
- **Est parcourue fréquemment** dans la recherche de solutions
 - Accès efficace pour vérifier si une solution est tabou
 - Table de hachage (sur les mouvements, sur la fonction objectif)
 - Si collision : Taille de la liste trop petite
- **Ne pas déconnecter** la solution optimale de la solution courante
 - Les informations restent dans la liste pendant une durée limitée (ie un nombre d'itérations)

137

Variantes (5) : Recherche Tabou

- **Exemple d'une liste Tabou**

- **Mouvement effectué sur les solutions :**
 - Interdire le mouvement inverse pendant k itérations
 - Itération p : solution obtenue après $\text{swap}(i, j)$
 - Interdire $\text{swap}(j, i)$ jusqu'à itération $p + k$
 - Matrice pour mémoriser toutes les paires de swap possibles
 - Mouvement inverse peut être complexe

- **Le contenu de la liste Tabou**

- interdit plus de solutions que celles réellement explorées
- Ne prévient pas totalement des risques de cyclage

138

Variantes (5) : Recherche Tabou

- **Durée des interdiction**

- **Ne conserver que les k derniers mouvements effectués**

- Valeur de k : longueur de la liste → compromis diversification / intensification
 - k faible :
 - peu de voisins interdits risque de rester bloqué sur un optimum local
 - k élevé :
 - beaucoup de voisins interdits / parcours potentiellement plus long
 - diversification importante mais on risque de louper l'optimum global
- Réglage adaptatif en fonction du problème / d'une instance

- **Annuler une interdiction**

- Autoriser mouvement tabou si amélioration de la fonction objectif
- Critère d'aspiration

139

Variantes (5) : Recherche Tabou

- **Attention à l'exploration du voisinage**

- Taille : se limiter si besoin à une liste de voisins candidats
 - Aléatoire
 - Les plus pertinents a priori
- Evaluation :
 - doit être efficace (incrémentale, approchée)

- **Variante**

- Mémoire dite à long terme pour guider la recherche
 - Mémoriser les mouvements effectués et leur qualités respectives
 - Diversification : Guider vers des parties non explorées
 - Intensification : Repartir de caractéristiques de bonnes solutions

140

Recherche Tabou - A compléter

141

Section 3. Méthodes approchées

- **Méthodes à population**

