Métaheuristiques

Marie-Ange MANIER

Extrait des cours de MANIER Marie-Ange (Formation d'ingénieurs, UTBM) :
« Métaheuristiques – IA52-53 »
et « Logistique – OL54 »

Métaheuristiques (plan)

- Introduction
- Heuristiques constructives
- Heuristiques de recherche locale
- Approches évolutionnaires

Méthodes d'optimisation

- méthodes **exactes** / méthodes **approchées**

trouvent la **solution optimale**
- programmation linéaire
- branch and bound
- programmation dynamique
-…

trouvent une **bonne solution**
- heuristiques spécifiques: FIFO, EDD, …
- métaheuristiques: recherche locale, recuit simulé, recherche tabou, algos génétiques, colonies de fourmis, essaims particulaires,…

- méthodes évaluant des solutions **complètes** / **partielles**

(incomplètes ou problèmes réduits)

Introduction aux méthodes heuristiques

Objectif : trouver

- des solutions raisonnablement bonnes
- pour des problèmes pratiques « intraitables » par méthodes exactes

  (combinatoires et/ou à contraintes spécifiques)
- en un temps raisonnable

D'où :

- algorithmes généralement spécifiques à chaque problème
- évaluation de performance par comparaison avec d'autres approches

  sur des problèmes de référence (bibliothèques) ou générés aléatoirement

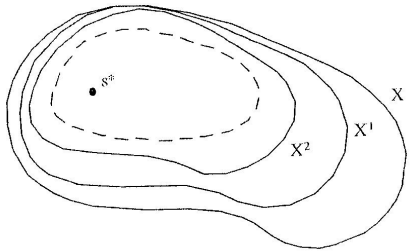
3 classes principales

heuristiques

constructives

décisions successives

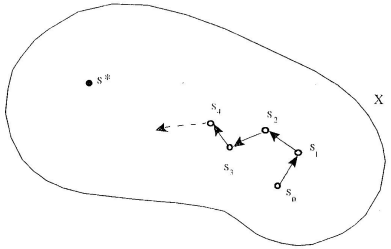
locales et définitives



S, 1, 2, 3, ?

de recherche locale

« voisinage »
mouvements

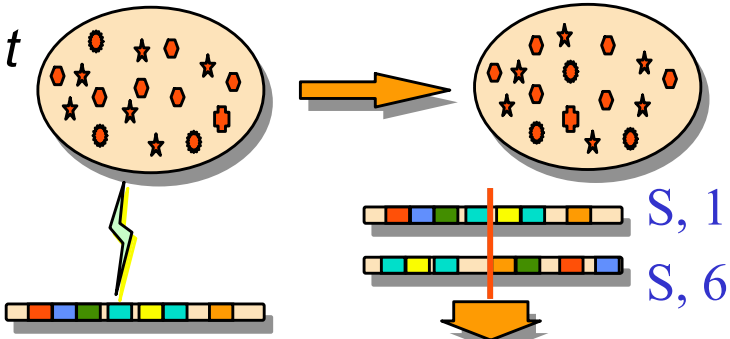


S, 1, 2, 3, 4, 5, D

S, 1, 2, 3, 4, 9, 10, D

évolutives

ensemble de solutions admissibles
Analogie avec des phénomènes naturels



S, 1, 2, 3, 4, 5, D

S, 1, 2, 3, 4, 5, D

S, 6,7, 8, 9,10, D

S, 1, 2, 3, 4, 9, 10, D

S, 1, 2, 3, 4, 9, 10, D

Les métaheuristiques

- famille d'algorithmes d'optimisation visant à résoudre des problèmes d'optimisation difficile pour lesquels on ne connaît pas de méthode classique plus efficace.

- grand nombre de métaheuristiques différentes

  simple recherche locale => algorithmes complexes de recherche globale.

Résolution par métaheuristiques

Modélisation

- du problème (réseau = graphe)
- d'une solution
  - codage
  - évaluation

Résolution

- opérateurs
- critères d'arrêt
- paramètres de réglages

Tests / Simulation

Modélisation

Représentation d'une solution et évaluation

Codage

- binaire,
- réel,
- séquences,
- arbres,…

Evaluation

- directe ou indirecte
- faisabilité et pénalisation
- forme de la fonction objectif en multi-objectif

Détermination d'une représentation

Le but est de déterminer le codage

lien (représentation solutions, réalité)

choix qui prennent en compte

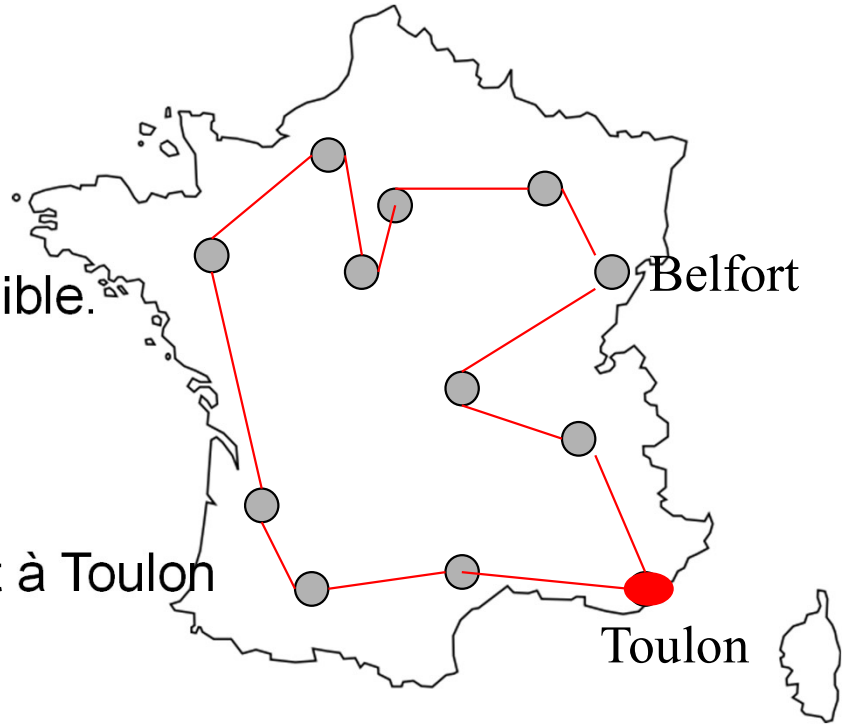
- les aspects relatifs au problème
- l'évaluation et la nature future des opérateurs.

Exemple : représentation ordonnée

- Les solutions sont représentées sous forme de permutations
- Utilisé pour les problèmes d'ordonnancement
- Besoin d'opérateurs spéciaux assurant que les solutions restent des permutations valides.

Exemple célèbre du TSP: Travelling Salesman Problem

- Un voyageur de commerce, basé à Toulon, doit visiter ses clients à travers la France.
- Il souhaite effectuer la tournée la plus courte possible.
- Instance : n villes avec une matrice de distances
- Solution : tournée visitant chaque ville et revenant à Toulon



Solution = tournée OU route = ensemble des clients visités

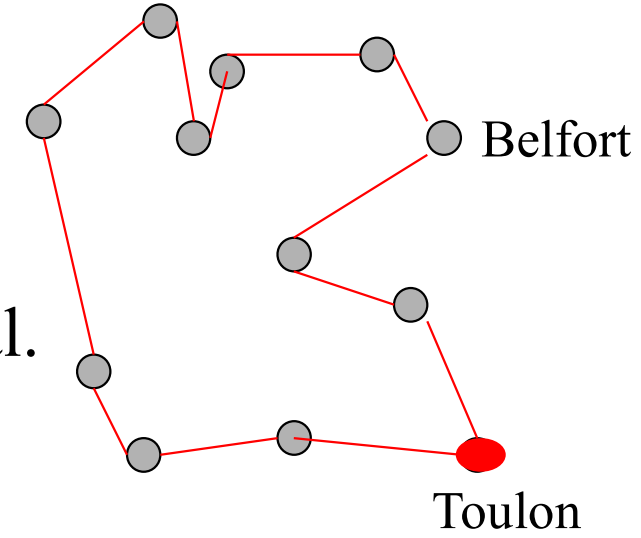
TSP = problème de tournées le plus simple

un commercial doit visiter ses clients 1 et 1 seule fois au cours d'une unique tournée

Soit G = (X;U;C) un graphe valué **complet**.

Le problème du voyageur de commerce consiste à trouver un **circuit hamiltonien** de coût minimal.

- représentation graphique:



- **Codage**: à chaque ville correspond un entier entre 1 et n.

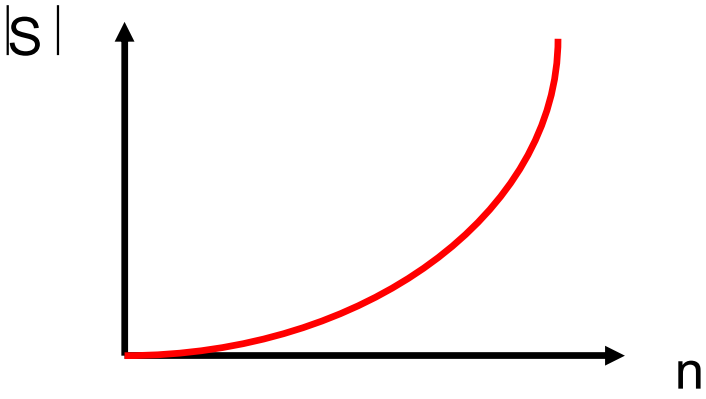
Une solution peut alors être (5, 4, 2, 1, 3) (ne pas oublier le retour en 5)

*un graphe est **complet** si pour tout couple de sommet Xi et Xj, il existe une arête entre Xi et Xj (il existe un arc (Xi, Xj) ou un arc (Xi, Xj)).*
***chemin hamiltonien** = **chemin** visitant une et une seule fois chaque sommet d'un graphe*

Optimisation combinatoire: **explosion combinatoire**

Taille de S par rapport à la taille du problème (n villes à visiter)

- Voyageur de commerce (n-1)! / 2
- Bi partitionnement 2^n
- Programme en variable bivalentes 2^n



- La taille de S est exponentielle

Cas du TSP

le nombre de chemins possibles passant par 71 villes est déjà un nombre d'une longueur de 100 chiffres, sachant qu'un nombre d'une longueur de 80 chiffres permettrait déjà de représenter le nombre d'atomes dans tout l'univers connu

Evaluation d'une solution

C'est généralement de loin l'étape la plus coûteuse pour les applications réelles.

Elle peut être directe ou indirecte

Elle peut être réalisée par une simple routine, un simulateur indépendant, ou par tout autre processus externe.

Cas du TSP: *évaluation directe d'une solution: calcul de la distance parcourue dans la tournée*

Evaluation d'une solution(2)

Gestion des contraintes : que faire des individus violant des contraintes relatives au problème ?

- pénaliser l'adaptation
- mise en place de méthodes spécifiques.
- problème de la dureté et de l'équilibre des contraintes

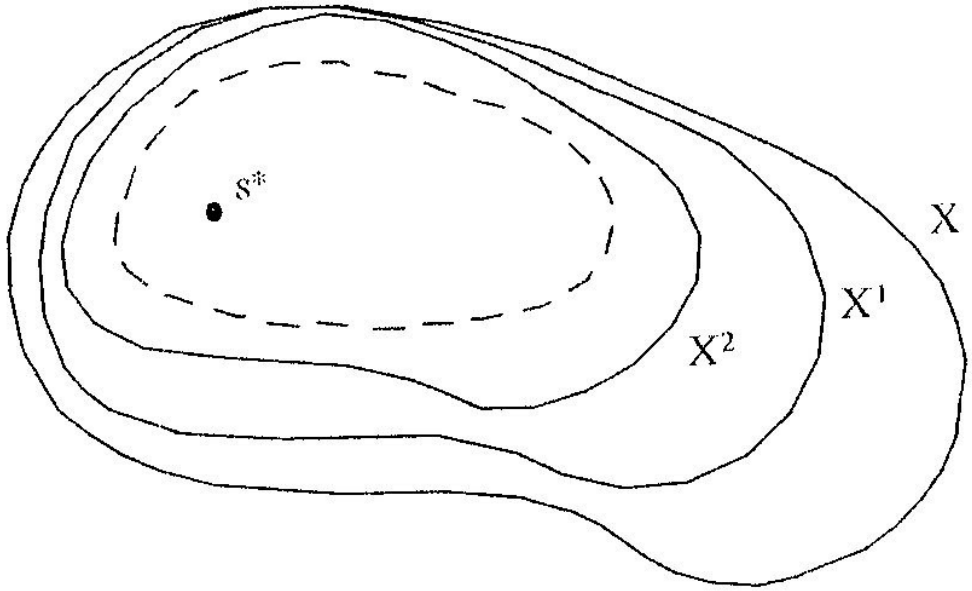
Résolution

Heuristiques constructives

construction progressive de $s = (x_1, x_2, \ldots, x_n)$

solution initiale vide s_0

décision prise à une étape donnée = indice de la composante à insérer
+ valeur de la composante



Heuristiques constructives (2)

algorithmes de type « **glouton** » :

meilleure décision « locale » (algorithmes dits myopes)

Inconvénients : peut fournir une très mauvaise solution

Avantages : rapidité et simplicité

Souvent utilisé comme point de départ d'autres méthodes

Heuristiques constructives (exemple)

Problème des reines: Maximiser le nombre de reines placées sur un échiquier n x n sans qu'elles soient « en prise »

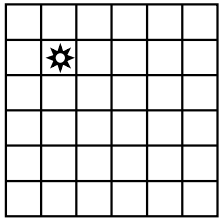
1 heuristique constructive possible : Tant qu'il reste des cases non occupées et non marquées

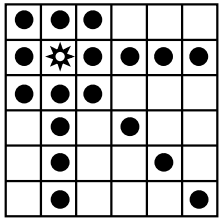
1.choisir une case au hasard et y placer une reine ✲

2. marquer les cases menacées par la reine ●

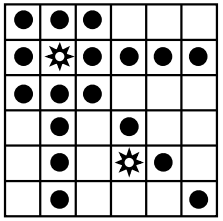
1ère itération

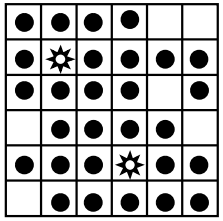
1.choisir 2. marquer



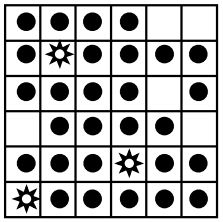


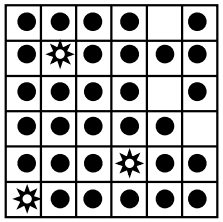
2nde itération



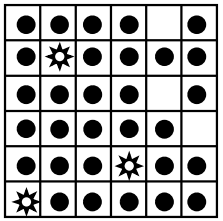


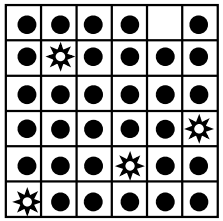
3ème itération



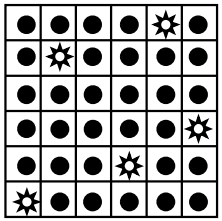


4ème itération





solution



<u>1 algorithme glouton possible</u> : choix de la case qui induira le nombre le moins important de cases marquées plutôt que choix aléatoire

Heuristiques constructives (exemple)

Problème du TSP: minimiser la distance totale parcourue sur la tournée

1- Heuristique du plus proche voisin: cas du TSP

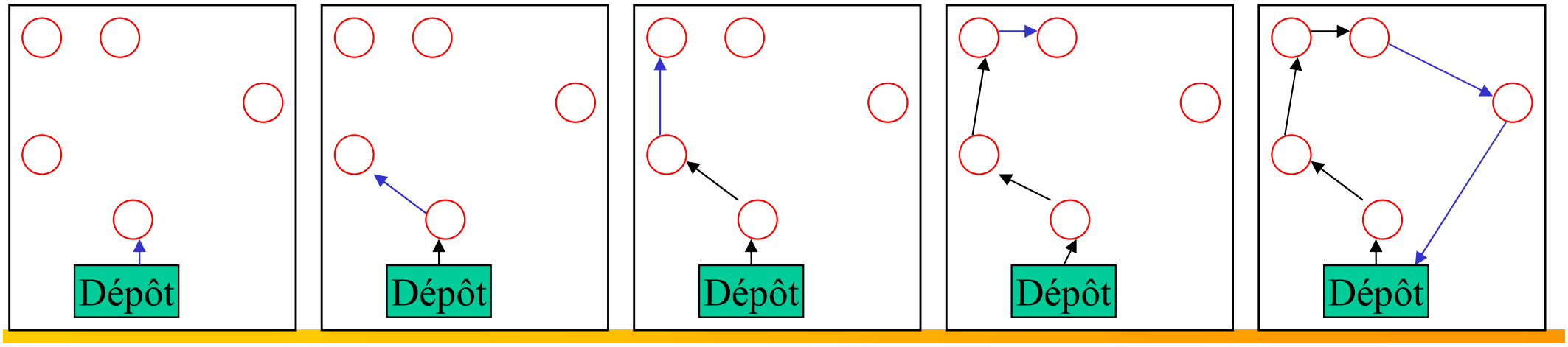
Dans cette heuristique, le déplacement doit toujours être fait vers le nœud le plus près qui n'a pas encore été visité. Plus précisément, elle doit être appliquée de la façon suivante :

•1. À partir du dépôt, **trouver le nœud *a* le plus près (au sens des coûts).** Créer l'arc dépôt -> a.
Le nœud actif est maintenant le nœud a.

2. À partir du nœud actif, trouver le nœud non visité le plus près. Créer l'arc liant ces deux nœuds. Le nœud qui était non visité devient le nœud actif.

3. Répéter l'étape 2 jusqu'à ce que tous les nœuds soient visités.

4. Créer un arc partant du nœud actif et se rendant au dépôt.



Heuristiques constructives (exemple)

Problème du TSP: minimiser la distance totale parcourue sur la tournée

2- Meilleure Insertion (insertion la moins coûteuse): cas du TSP

Cette heuristique construit une tournée en insérant toujours le noeud qui implique les coûts supplémentaires les moins élevés. Plus précisément, la tournée se construit comme suit :

1. À partir du dépôt, trouver le noeud *a* le plus loin. Créer l'arc dépôt -> a.

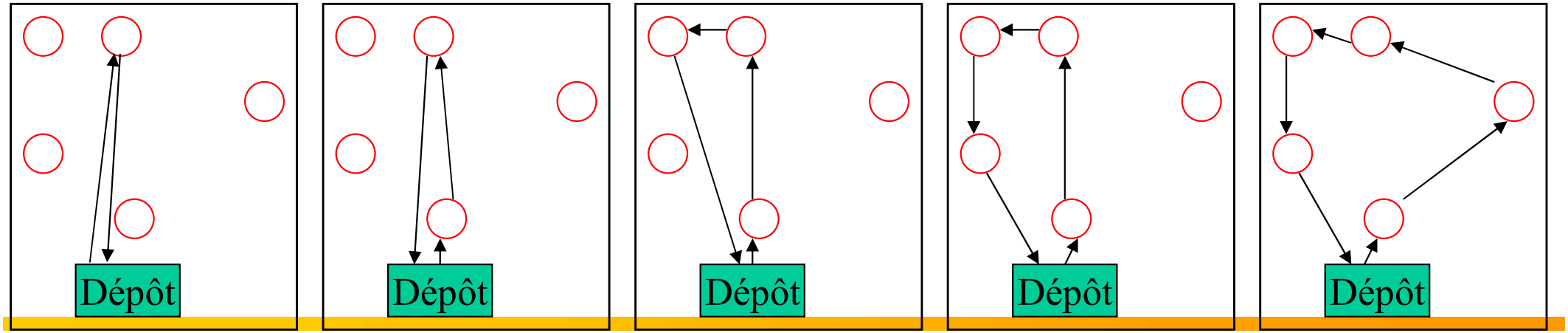
2. Calculer les coûts supplémentaires d'ajouts de chacun des noeuds non visités de façon individuelle. Ces valeurs se calculent ainsi :

Coût d'insérer le noeud *i* sur l'arc a_1-a_2 = c_{a1i} + c_{a2i} – c_{a1a2}

où c_{ij} représente le coût pour circuler du noeud *i* au noeud *j*

3. Insérer le noeud non visité le moins dispendieux à intégrer à la tournée actuelle.

4. Répéter les étapes 2 et 3 jusqu'à ce que tous les noeuds soient visités.



Heuristiques constructives (exemple)

Problème du TSP: minimiser la distance totale parcourue sur la tournée

DIVERSES MÉTHODES D'INSERTION

Partir d'un circuit réduit à une boucle, choisir un sommet libre et l'insérer entre 2 sommets en minimisant l'augmentation de coût

$$\Delta C = Cij + Cjk - Cik$$

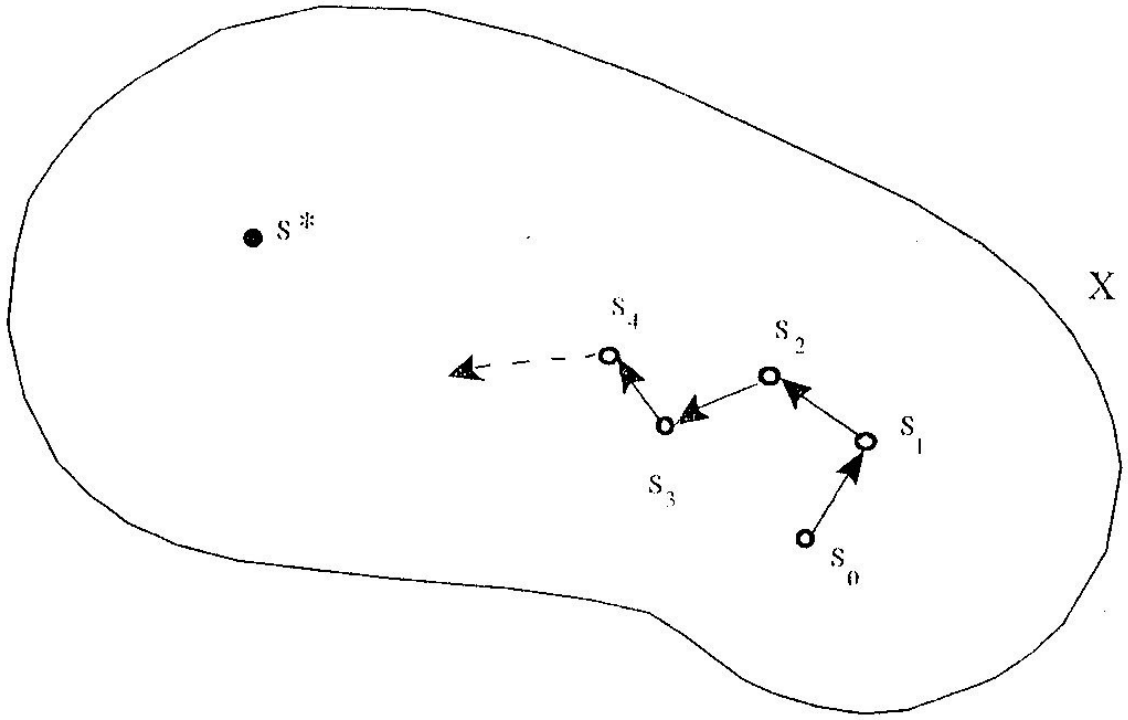
- Plus Proche Insertion
  choisir le sommet le plus proche de la tournée construite et l'insérer au meilleur endroit
- Plus Lointaine Insertion
  choisir le sommet le plus loin de la tournée construite et l'insérer au meilleur endroit
- Meilleure Insertion
  tester tous les sommets sur toutes positions de la tournée et choisir le meilleur couple (sommet, position)

Heuristiques de recherche locale

recherche itérative d'une solution meilleure

dans le « voisinage » V(s) de la solution courante

Exploration de X



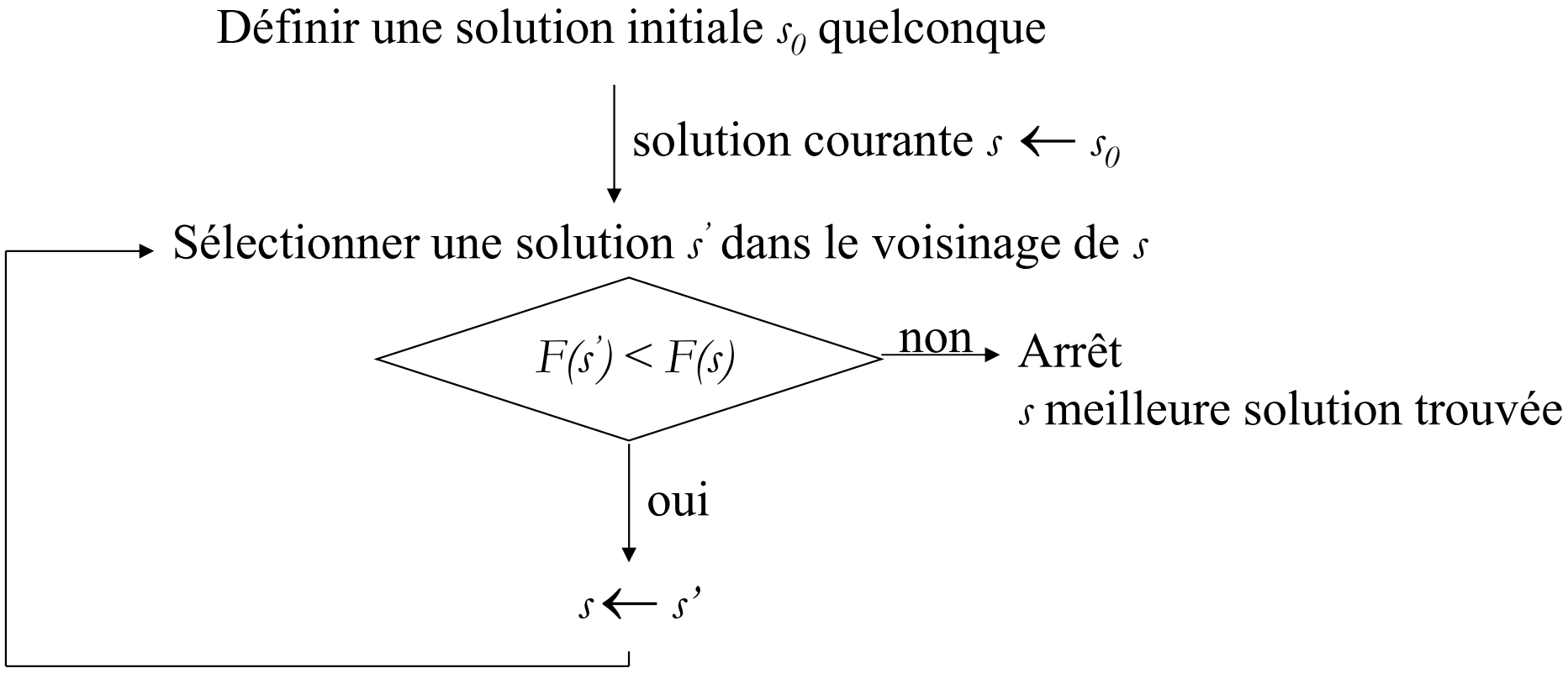
Exemple :

méthodes de gradient dans le domaine continu, de descente dans le cas discret

Descente : principe

Problème : $\min F(s)$
$s \in X$

Choisit toujours une solution voisine meilleure que la solution courante :



Heuristiques de recherche locale : mise en œuvre

Nécessite de disposer de (et/ou de choisir)

- un « codage »

  (modélisation du problème, définition de X)

- une solution initiale

  (peut-être générée aléatoirement ou par une heuristique constructive)

-une notion de voisinage $V(s)$

  (ensemble de solutions obtenues à partir d'une solution courante en lui faisant subir des transformations élémentaires, on parle aussi de passage d'un état à l'autre à la suite de mouvements)

- un moyen efficace pour trouver la meilleure (ou à défaut une bonne) solution dans le voisinage d'une solution quelconque

Heuristiques de recherche locale : exemple du TSP

- exemple de « codage » pour modéliser un TSP à 4 villes :

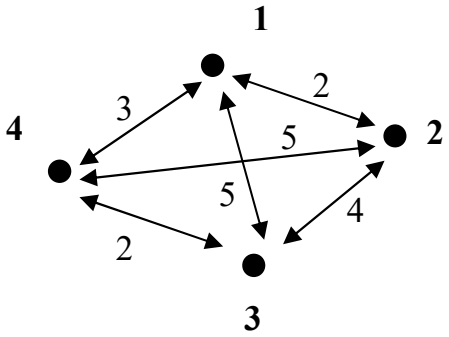
liste des villes dans leur ordre de visite

- exemple de solution initiale : $S_0 = (1,2,3,4)$
- exemple de voisinage

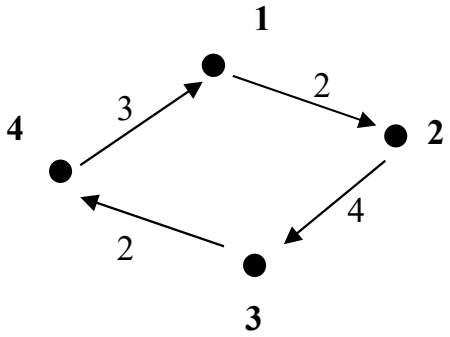
1 mouvement = échange de deux villes au hasard

$S = (1,4,3,2)$ solution voisine de S_0 obtenue en permutant *2* et *4*

Données du problème

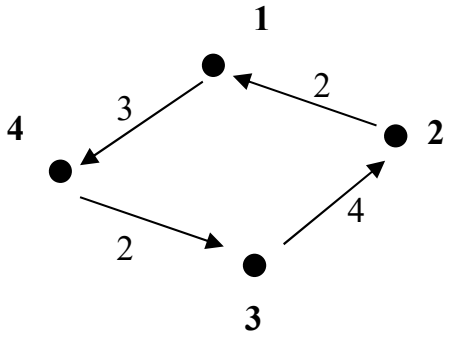


$S_0 = (1,2,3,4)$



$F(S_0) = 2+4+2+3=11$

$S = (1,4,3,2)$



$F(S) = 3+2+4+2=11$

Heuristiques de recherche locale : exemple des reines

Exemple de voisinage pour le problème des reines précédent

- placer aléatoirement une reine sur un échiquier vide et marquer les cases correspondantes

- replacer les reines de l'état courant pour lesquelles c'est possible (dont l'ancienne place dans la solution courante correspond à une case non marquée du nouvel échiquier) et marquer les cases correspondantes

- si il ne reste plus de case libre (non occupée et non marquée) le voisin a été obtenu et sinon tant qu'il reste des cases libres placer aléatoirement une reine et marquer les cases correspondantes

Descente : piège des optima locaux

Principe : analogie entre la valeur de la fonction objectif $F(s)$ pour une solution s,
et l'altitude d'un point dans un paysage vallonné

Chaque fond de vallée = 1 optimum (minimum)

La vallée la plus profonde = optimum globale

Autres vallées = optima locaux

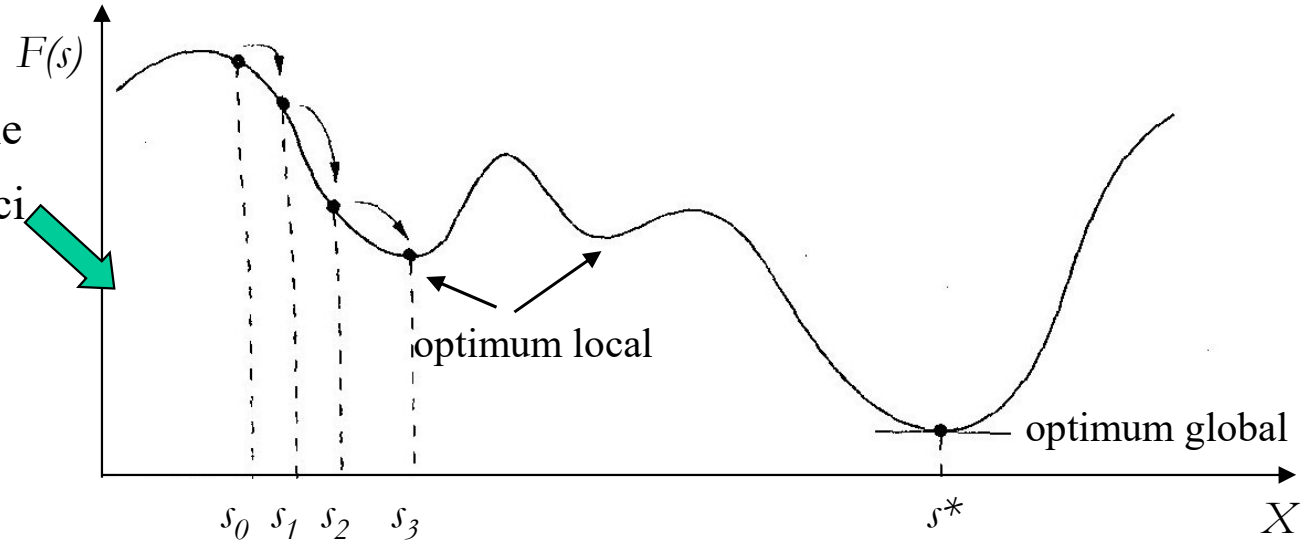
Inconvénient majeur de la descente : arrêt sur optimum local et non global

L'algorithme de descente ne sait pas « remonter » pour sortir d'un optimum local

Réversibilité non vérifiée :
la descente ne saurait pas faire « marche arrière » pour atteindre s^* si elle était ici

Accessibilité non vérifiée :
il n'existe pas de chemin entre s_0 et s^*,

↳ la descente rend la solution s_3
optimum local



Heuristiques de recherche locale : autres algorithmes

Principe :

acceptent des solutions moins bonnes que la solution courante pour s'extraire d'un optimum local, tout en maintenant une « pression » pour favoriser les solutions qui améliorent l'objectif

Méthodes « récentes » (années 1980), très utilisées

Avantage : + performantes que la descente

Inconvénient : + complexes (notamment à régler) et + coûteuses

≠ manières de sortir d'un optimum local :

- réglage d'un paramètre de « température » : recuit simulé
- gestion d'une liste des états interdits : méthode tabou
- …

Heuristiques de recherche locale : autres algorithmes

Ces méthodes sont :

- + simples et + rapides à mettre en œuvre qu'une méthode exacte
  quand on n'exige pas l'optimum global mais une « bonne » solution

- + souples dans le cas de problèmes réels : s'adaptent + facilement
  - à des contraintes additionnelles venant du domaine d'application
  - à des contraintes qu'un client exprimerait en cours de projet

    (parce qu'il ne savait pas les exprimer au départ, souvent fatal pour une méthode exacte!)

Elles fournissent généralement

des bonnes solutions

des bornes utilisables par des méthodes exactes

Le recuit simulé: idée de base

Analogie avec l'opération de « recuit » (*annealing*) en sidérurgie

Recuit : réchauffement puis refroidissement lent de la matière après déformation,
pour faire disparaître les tensions internes dues à la déformation

Phénomènes physiques observés :

l'énergie fournie par le réchauffement permet des mouvements des atomes et le refroidissement lent fige le système dans un état d'équilibre d'énergie minimale

Travaux de (Metropolis et al., 1953) : 1 algorithme simple pour simuler ces phénomènes

- état du système = position des atomes
- passage d'1 état à l'autre : déplacement infinitésimal aléatoire d'1 atome quelconque
- ΔE : variation d'énergie du système provoquée par ce mouvement
- nouvel état accepté si $\Delta E < 0$, et sinon accepté avec une probabilité

$P(\Delta E, T) = exp(-\Delta E / (k_B . T))$ avec T : *température du système* et k_B : *constante physique (de Boltzmann)*

on génère 1 nombre aléatoire $q \in [0,1[$, si $q < P(\Delta E, T)$ nouvel état accepté

sinon conservation état courant

Le recuit simulé: idée de base (2)

(Metropolis et al., 1953) ont montré que

l'utilisation répétée de cette règle d'acceptation des mouvements fait évoluer le système vers un état d'équilibre d'énergie minimale

Si aucun état nouveau n'a été accepté à une température T donnée, on suppose avoir atteint cet état d'énergie minimale

Rôle de la température :

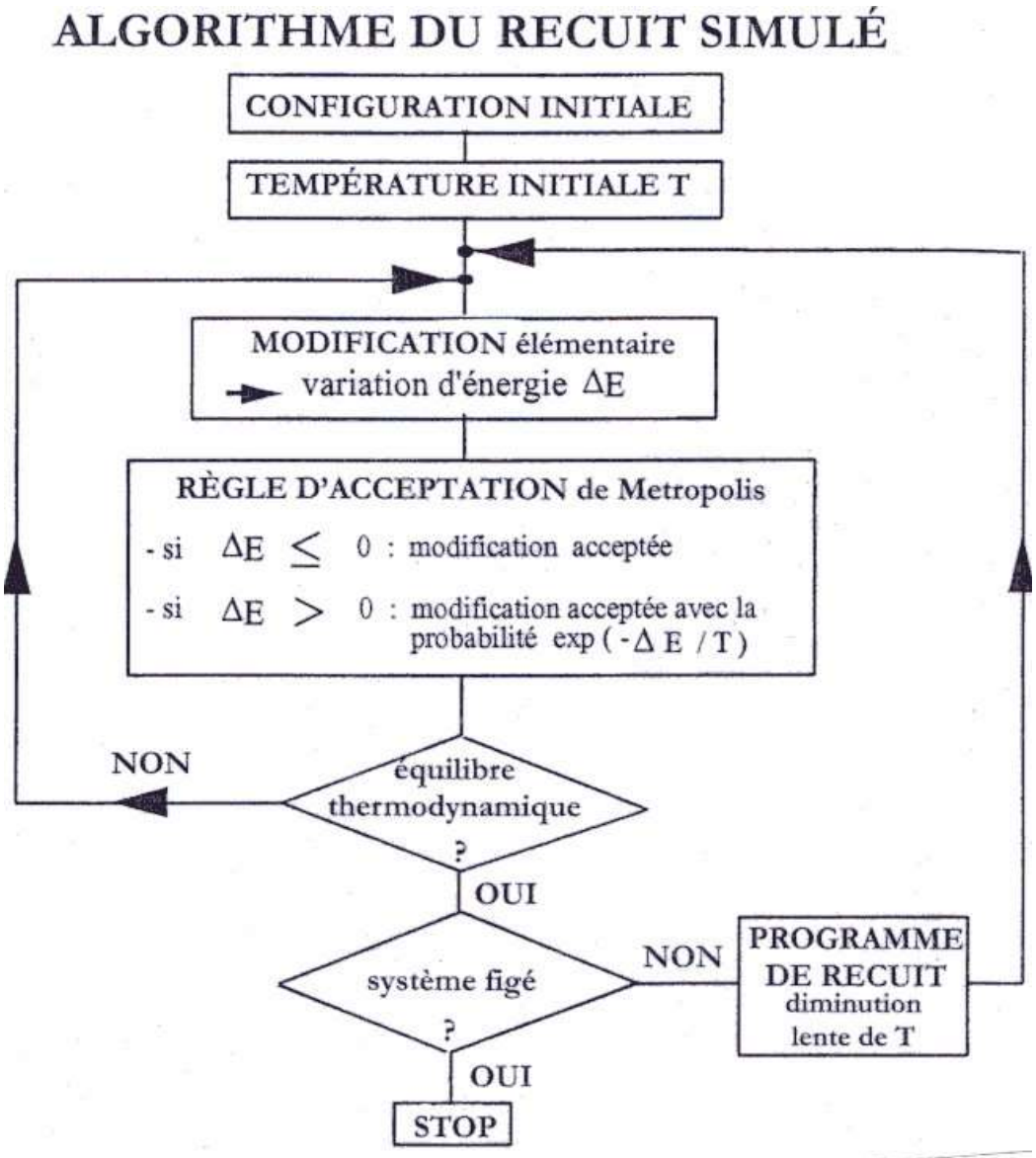
+ T est grand, + les mouvements aléatoires des atomes sont importants

(fortes transformations)

refroidissement : mouvements de + en + faibles

grande lenteur du refroidissement garantit un état d'équilibre à chaque T

Le recuit simulé: schéma général



Le recuit simulé: analogie avec l'optimisation

(Kirkpatrick et al., 1953) : transpose le principe à la descente

≠ états du système	solutions admissibles $s \in X$
énergie du système	fonction à minimiser $F(s)$
mouvements d'atomes	mouvements/transformations

Principe : au lieu de n'accepter que des voisins qui diminuent $F(s)$

on autorise des augmentations (même importantes) de $F(s)$

au début de l'algorithme puis au fur et à mesure que le temps passe

on autorise ces augmentations de plus en plus rarement

(la température baisse)

Le recuit simulé: analogie avec l'optimisation (2)

($V(s)$) : ensemble des états (solutions) atteignables à partir de s

A chaque itération, 1 solution voisine s' est générée

elle est acceptée si $\Delta F = F(s') - F(s) < 0$,

et sinon avec une probabilité

$P(\Delta F, T) = exp(-\Delta F / T)$ avec T : *paramètre de température à régler*

Changements de température : selon un schéma précis (le plus souvent par paliers)

Meilleure solution trouvée est mémorisée ($s^\wedge$)

Critère d'arrêt : aucune solution voisine n'a été acceptée durant 1 cycle complet d'itérations à température constante

Le recuit simulé : algorithme

Initialisation : soit $x_0 \in X$, une solution initiale ;
$\hat{F} = F(x_0)$

Etape n **:** soit $x_n \in X$, la solution courante ;
tirer au sort une solution $x^\star \in V(x_n)$;
si $F(x^\star) \leqslant F(x_n)$, faire $x_{n+1} = x^\star$;
si $F(x^\star) < \hat{F}$, faire $\hat{F} = F(x^\star)$;
sinon, tirer un nombre q au hasard entre 0 et 1 ;
si $q \leqslant p$, faire $x_{n+1} = x^\star$;
sinon, faire $x_{n+1} = x_n$;
si la règle d'arrêt n'est pas satisfaite,
passer à l'étape $n + 1$;
sinon, stop.

Remarque : la descente est un recuit simulé à température nulle

Initialisation : soit $x_0 \in X$, une solution initiale ;

Etape n **:** soit $x_n \in X$, la solution courante ;
sélectionner une solution $x^\star \in V(x_n)$;
si $F(x^\star) \leqslant F(x_n)$,
faire $x_{n+1} = x^\star$ et passer à l'étape $n + 1$
sinon x_n est la meilleure solution trouvée ; stop.

Le recuit simulé : réglage des paramètres

T_0 : température initiale

dépend du taux d'acceptation voulu pour les solutions dégradant F

qui doit être assez élevé au début de l'algorithme

- fixer le taux moyen d'acceptation (par exemple 50 %)
- simuler la détérioration moyenne $(<\Delta F>)$ de ΔF

  en générant plusieurs solutions à partir d'une solution initiale fixée
- calculer T_0 telle que $exp\ (- <\Delta F> / T_0) = 0{,}5$, soit $T_0 = <\Delta F> / ln2$

Changements de température : schéma de refroidissement par paliers

T_0 : L itérations

$T_1 = \alpha\ T_0$: L itérations $\quad 0 < \alpha < 1$

$\ldots T_k = \alpha^k\ T_0$: de l'itération $kL + 1$ à l'itération $(k + 1)\ L$

Le recuit simulé : réglage des paramètres

L : longueur d'un palier de température

dépend de la taille du voisinage et augmente avec la taille du problème

(il faut plus de temps pour explorer une région plus vaste donc plus d'itérations)

a une grande influence sur la durée d'exécution

α : (généralement 0,9 ou 0,95)

contrôle la lenteur du refroidissement

valeur en lien avec celle de L

(on tolère un refroidissement plus rapide si les paliers sont plus longs)

Le recuit simulé : avantages et limites

Avantages : il existe des études théoriques de convergence

Inconvénients : ne permet pas de traiter des problèmes dont la fonction objectif peut prendre des valeurs infinies

réglage des paramètres souvent empirique (par tâtonnements)

Remarque : une bonne solution initiale (obtenue par algorithme glouton par exemple) ne sera pas forcément utile au recuit puisque c'est au début de l'algorithme que se font les plus grosses transformations

La méthode tabou: idée de base

Développée dans un cadre particulier par Glover en 1986 puis généralisée en 1989-90

Comme la descente, se déplace d'1 solution courante s vers 1 solution voisine s' telle que $F(s') = \min_{s'' \in V(s)} F(s'')$

Si optimum local, déplacement vers le « moins mauvais » des voisins

Mais risque de boucle en faisant l'opération inverse à l'itération suivante !!!

↳ idée : interdire la dernière opération

Mais si la « vallée » de l'optimum local est « profonde » : plusieurs itérations nécessaires pour s'en extraire

↳ idée : garder en mémoire les dernières solutions visitées et interdire le retour vers celles-ci pour 1 nombre fixé d'itérations

La méthode tabou: mise en oeuvre

Dans la pratique : conservation à chaque étape d'une ou plusieurs

listes T de solutions « taboues »

2 alternatives :

une liste contient les solutions interdites (coûteux en place mémoire)

ou

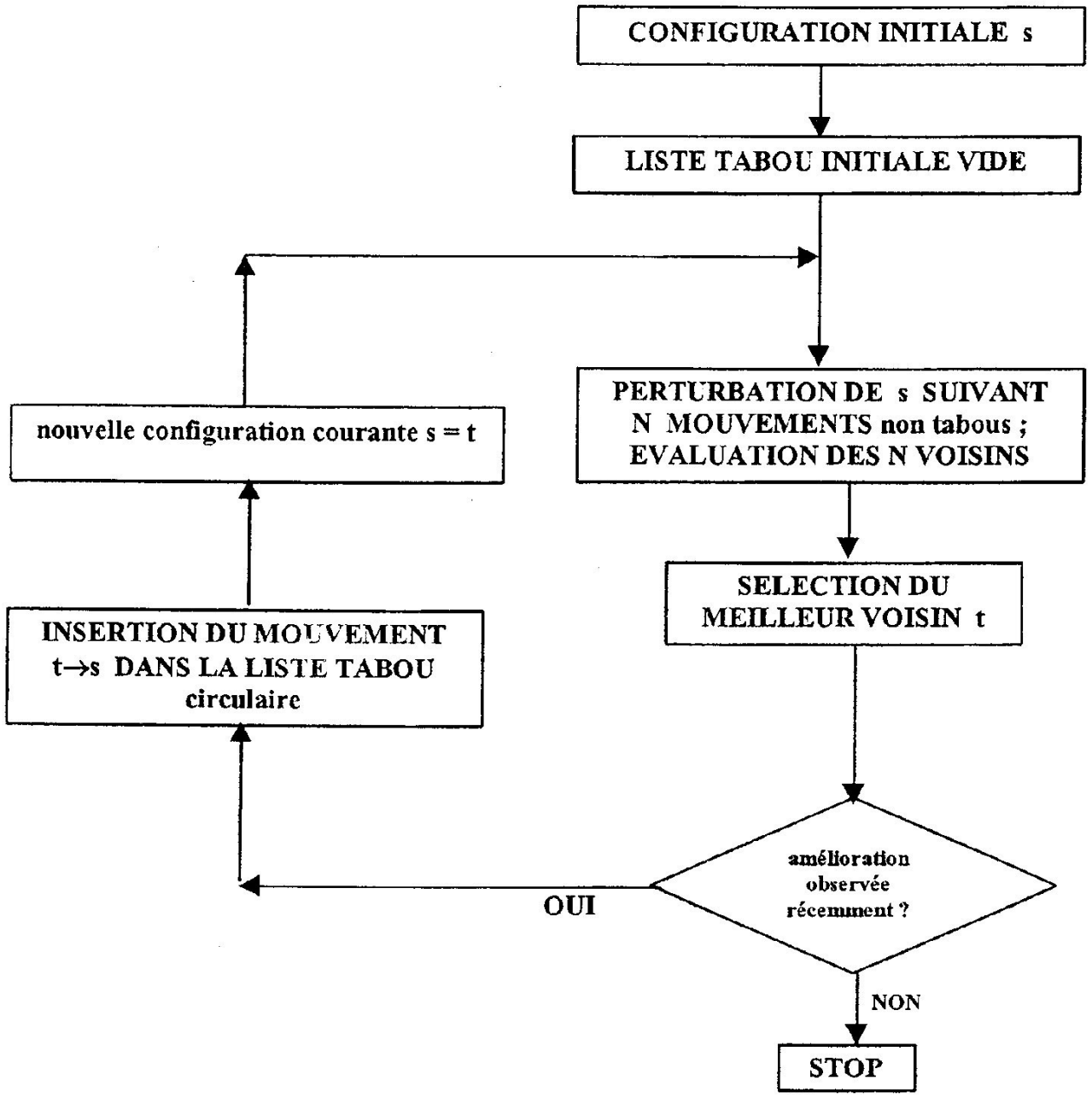
une liste contient les mouvements interdits (qui ramèneraient vers ces solutions déjà visitées)

Avantages : prend moins de place mémoire

élimine plus de solutions que celles visitées effectivement

Généralement les listes sont gérées en FIFO (first in first out)

La méthode tabou: schéma général



La méthode tabou

Initialisation : soit $x_0 \in X$, une solution initiale ;
$\hat{F} = F(x_0)$:
$k =$ longueur de la liste tabou.

Etape n **:** soit $x_n \in X$, la solution courante ;
chercher, dans un sous-voisinage V^* de $V(x_n)$,
la meilleure solution x^* qui soit :
non tabou

faire $x_{n+1} = x^*$;
si $F(x^*) < \hat{F}$, faire $\hat{F} = F(x^*)$;
mettre à jour la liste tabou ;
si la règle d'arrêt n'est pas satisfaite,
passer à l'étape $n + 1$;
sinon, stop.

Remarque : comme dans la descente, si trouver la meilleure solution du voisinage est trop coûteux on peut se contenter d'un échantillon du voisinage, voire même de sélectionner la première solution voisine non taboue générée

La méthode tabou: réglage des paramètres

- le nombre et la longueur des listes tabou

  généralement de 5 à 50 environ (7 est un chiffre courant)
  liés au voisinage et à la nature des tabous (solutions ou mouvements) choisis

- le critère d'arrêt peut être, par exemple,

  - un temps de résolution (temps de réponse maximal imposé par l'utilisateur)
  - un nombre d'itérations maximal, soit au total, soit entre deux améliorations de la meilleure solution rencontrée
  - un écart « suffisamment » petit par rapport à l'optimum global (ce qui nécessite de connaître au moins la valeur de l'optimum ou une borne inférieure

Heuristiques de recherche locale : conclusion

- la principale difficulté pour la mise en œuvre de telles méthodes est le réglage des paramètres, fortement lié au problème traité et généralement effectué expérimentalement

- un reproche qui leur est souvent fait est l'absence (pour certaines en tout cas) de preuve de convergence théorique, ou alors de manière asymptotique, sans délimiter le nombre d'itérations nécessaires pour avoir une « bonne » solution

- néanmoins elles sont relativement souples et adaptables à tous types de problèmes

- et surtout elles ont déjà fait leurs preuves en fournissant des résultats impressionnants pour de nombreux problèmes combinatoires

Approches évolutionnaires

Quel est le mécanisme de résolution existant le plus efficace ?

- Le cerveau humain

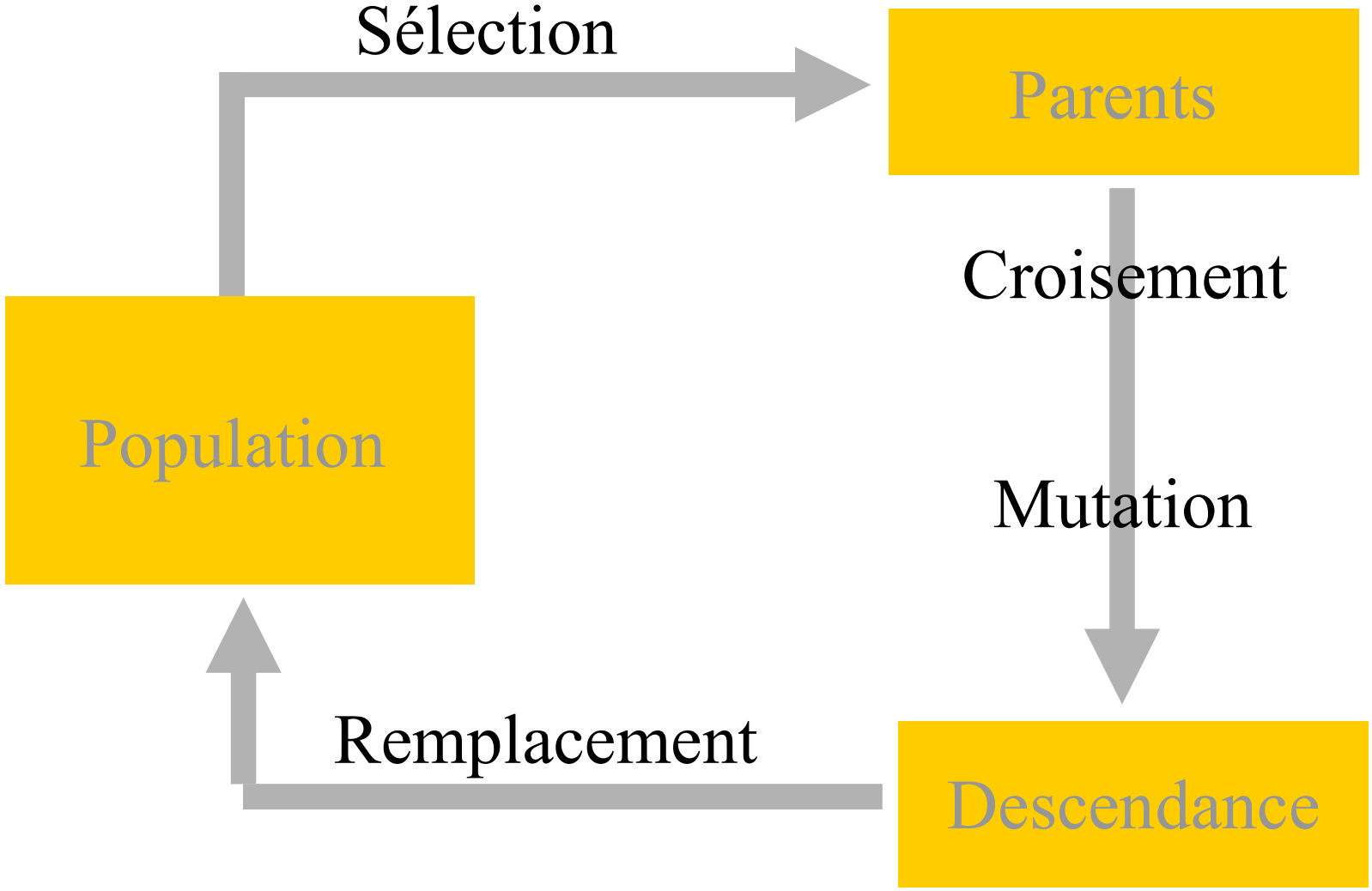
- Le mécanisme d'évolution naturelle
  qui a créé le cerveau humain
  (Darwin et al.)

Réaliser un mécanisme de résolution en s'inspirant :

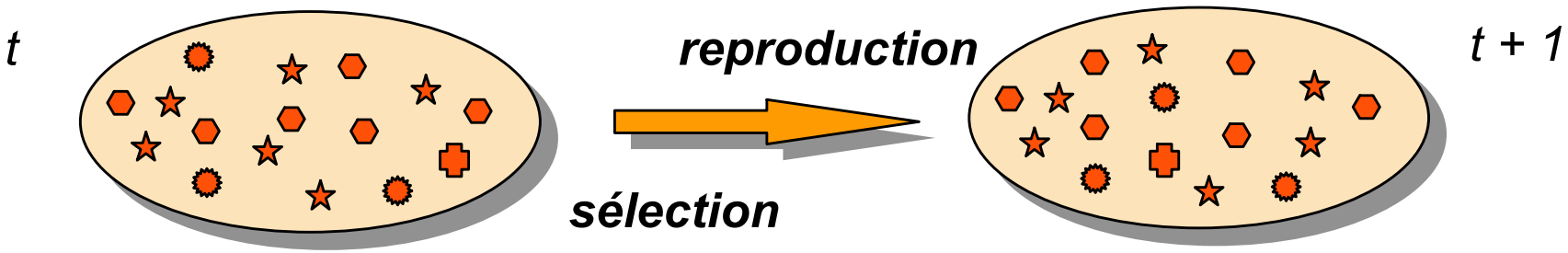
Du cerveau ⇒ Réseaux de neurones (neurocomputing)

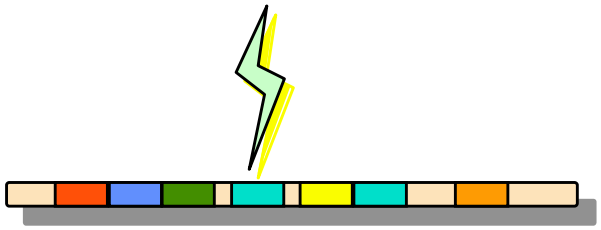
De l'évolution ⇒ Algorithmes évolutionnaires (evolutionary computing)

Cycle d ’évolution

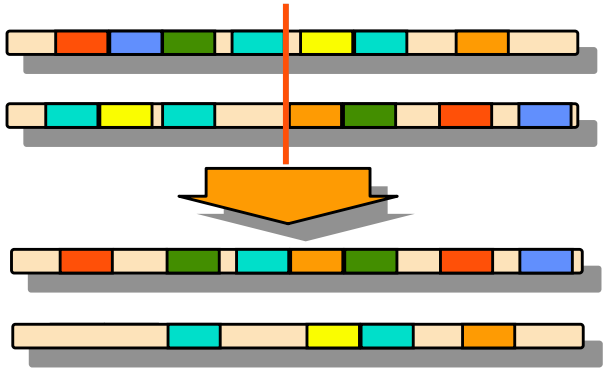


Ingrédients





mutation



Recombinaison

Algorithme général

Générer (P_0)
t←0
tant que non (CritèreTerminaison(P_t)) faire
    Evaluer (P_t)
    P'_t ←Sélectionner(P_t)
    P''_t← OpérateursReproduction (P_t)
    P_{t+1} ← Remplacer (P_t , P''_t)
    t ←t + 1
retourner (MeilleureSolution(P_t))

Mécanisme d ’évolution

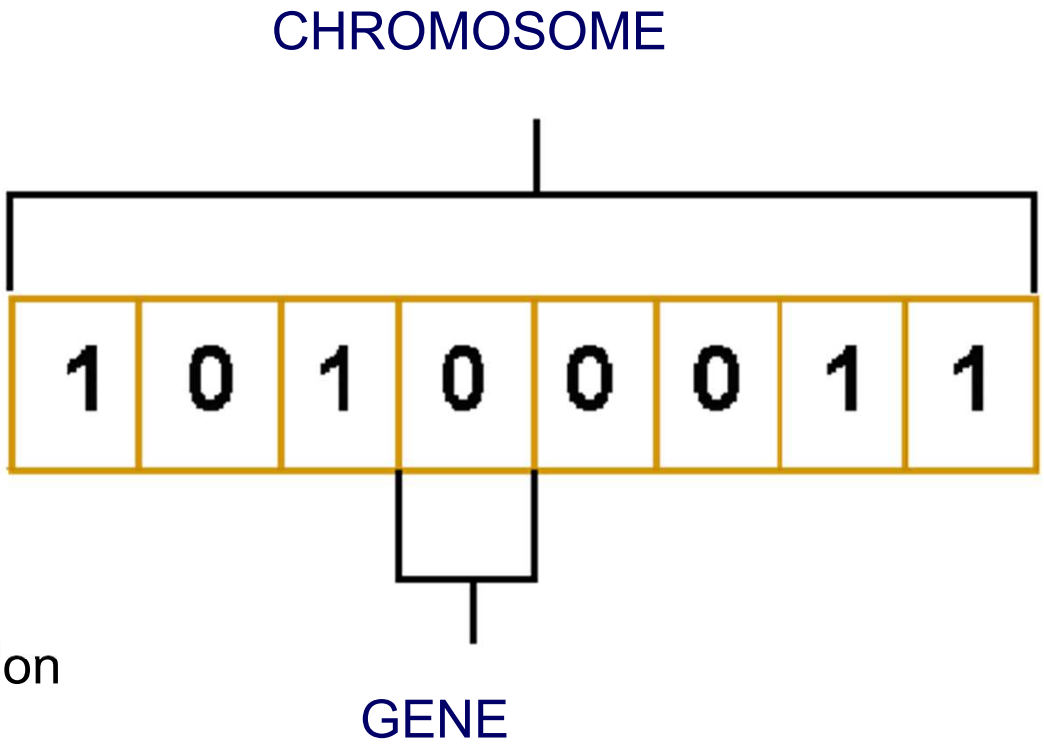
- Augmentation de la diversité par les opérateurs génétiques
  - mutation
  - croisement
- Diminution de la diversité par la sélection
  - des parents
  - des survivants

Résumé

- Fondés sur une métaphore biologique
- Adaptables à de nombreux domaines
- Rapport performance/coût intéressant

Principales étapes de la conception

- Représentation/codage
- Evaluation de l ’individu
- Initialisation de la population
- Sélection des parents
- Opérateurs de croisement et mutation
- Sélection des individus à remplacer
- critère d ’arrêt



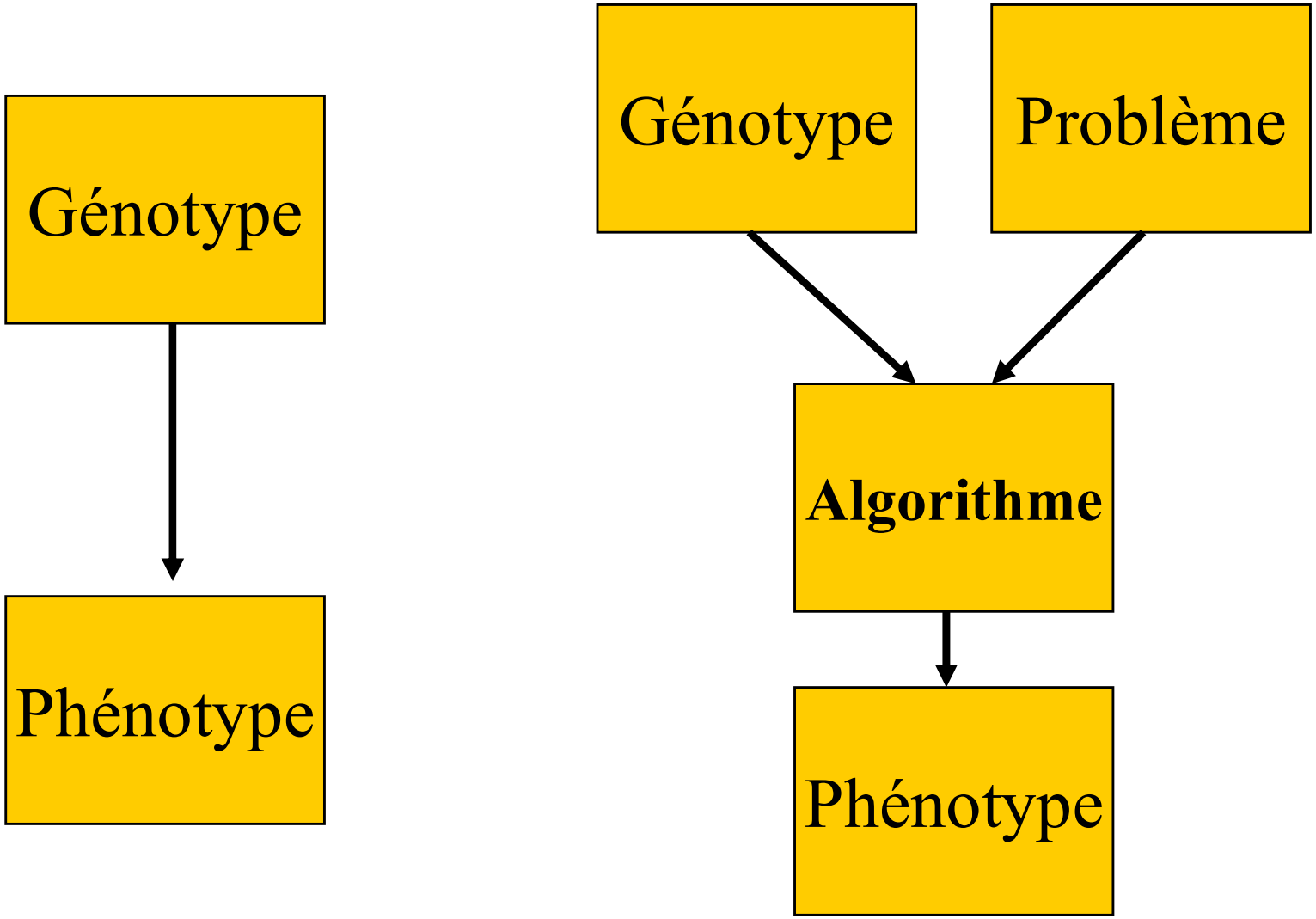
Détermination d'une représentation

Le but est de déterminer le codage (phénotype) représentant le génotype de l'individu.

Les choix doivent prendre en compte les aspects relatifs au problème.

Le choix de la représentation doit se faire en fonction de l'évaluation du génotype et de la nature future des opérateurs génétiques.

Passage du génotype au phénotype : évaluation



Initialisation

Répartition uniformément sur l ’espace de recherche…(si possible)

relativement au génotype :

- alphabet binaire : probabilité 0.5
- représentation réelle : uniformément sur les intervalles.

Initialisation à partir de résultats précédents ou au moyen d’heuristiques

à utiliser avec précaution :

- possible perte de diversité génétique
- introduction possible d ’un biais irrécupérable

Stratégie de sélection

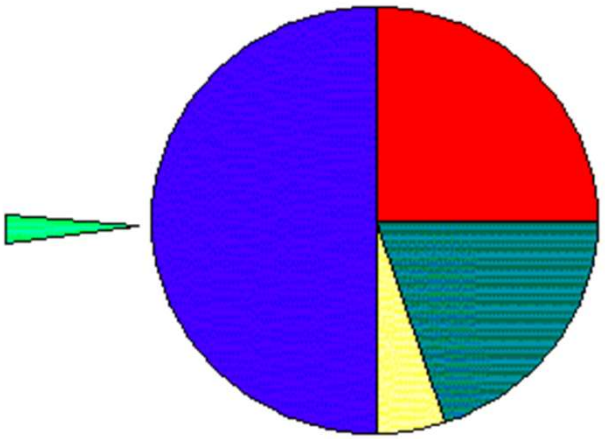
Le but de la sélection est d'assurer que les individus les mieux adaptés ont plus de chances de se reproduire que les individus moins bien adaptés.

- Cela produit une pression sélective qui permet l 'amélioration de la population au fil des générations.

- Il faut cependant faire attention de laisser une opportunité aux individus faibles, qui peuvent être porteurs de matériel génétique utile.

Exemple : sélection proportionnelle par roulette

Réalisation de n tirages : un individu *i* a une probabilité $f_i / \bar{f}$ d 'être sélectionné



Les individus adaptés ont donc plus de chance de se reproduire

Exemple : sélection proportionnelle par roulette

Désavantages :

- Danger de convergence prématurée due à l ’apparition d ’individus exceptionnels prenant rapidement le dessus sur l ’ensemble de la population.
- Faible pression sélective lorsque les fitness des individus sont proches.

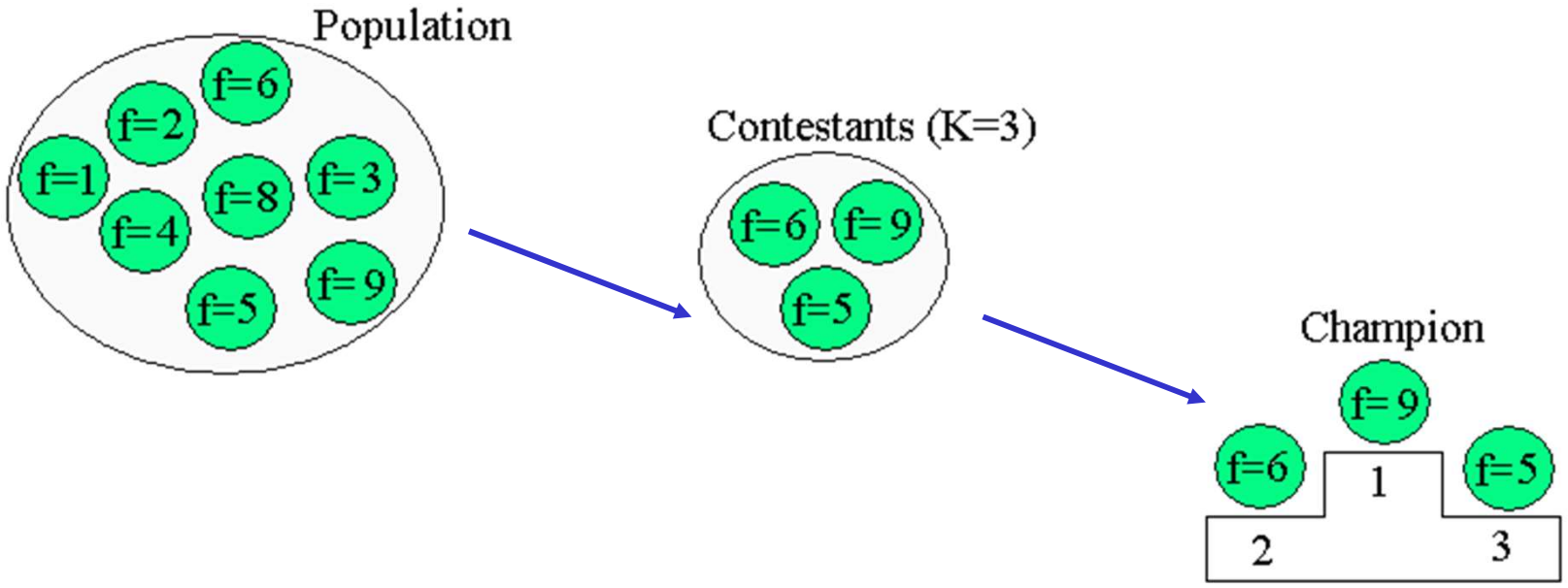
Solution : mise à l’échelle.

- Standardiser pour assurer que les fitness sont comprises entre 0 et 1 et que leur somme est égale à 1

Exemple : sélection par tournoi

Sélectionner aléatoirement *k* individus et prendre le meilleur. (*k* : taille du tournoi)

Recommencer jusqu ’à obtention du nombre d ’individus désirés.



Exemple : sélection par rang (ranking)

Les individus sont classés par ordre décroissant de fitness, associant ainsi un rang à chaque individu.

Au lieu d ’être sélectionnés proportionnellement à leur fitness, les individus sont sélectionnés proportionnellement à leur rang.

⇒ Permet d ’éviter la convergence prématurée due à l ’apparition d ’individus exceptionnels

Croisement et Mutation
Approche Classique

Croisement :

- effet décroissant avec la convergence
- opérateur d ’exploitation

Mutation :

- nécessaire pour s ’échapper de minima locaux
- opérateur d ’exploration

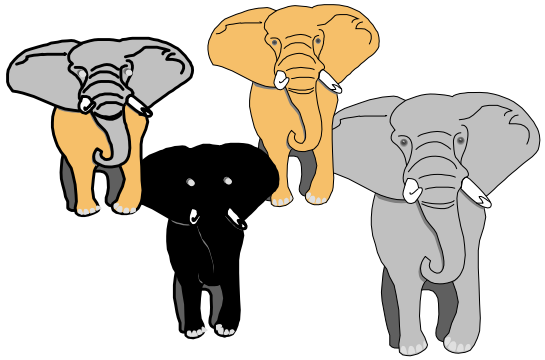
Opérateurs de croisement

Nous avons un ou plusieurs croisement pour la représentation choisie :

- L ’individu fils doit hériter du capital génétique de ses deux parents.
- Le croisement doit prendre en compte la représentation de manière à éviter de produire des individus non valides.

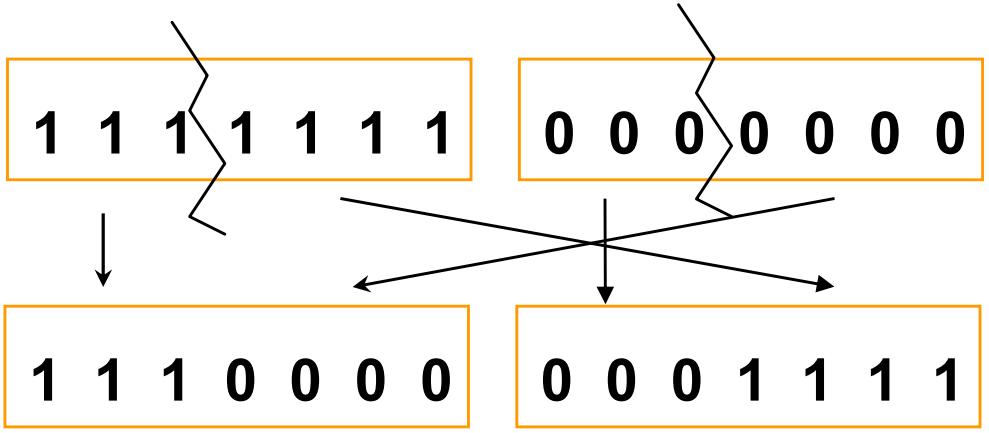
Exemple : croisement pour une représentation discrète

Population entière:

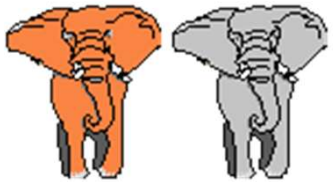


. . .

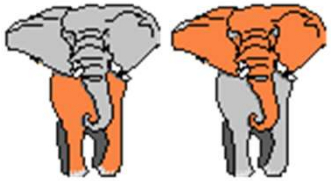
Chaque chromosome est divisé en n parties qui sont alors recombinées. (*n=1*)



parents

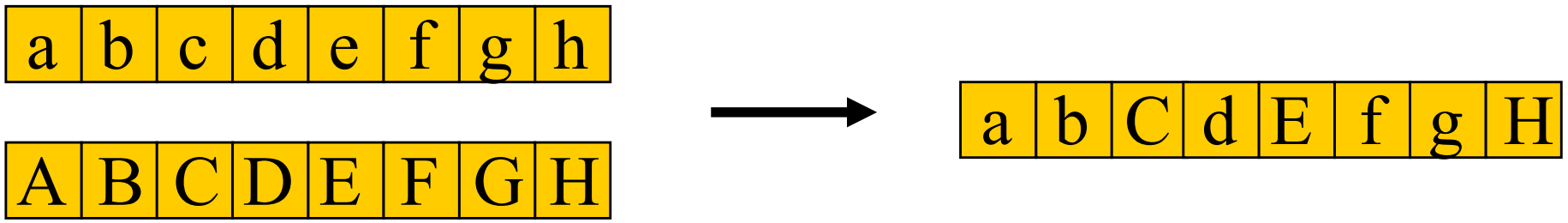


enfants



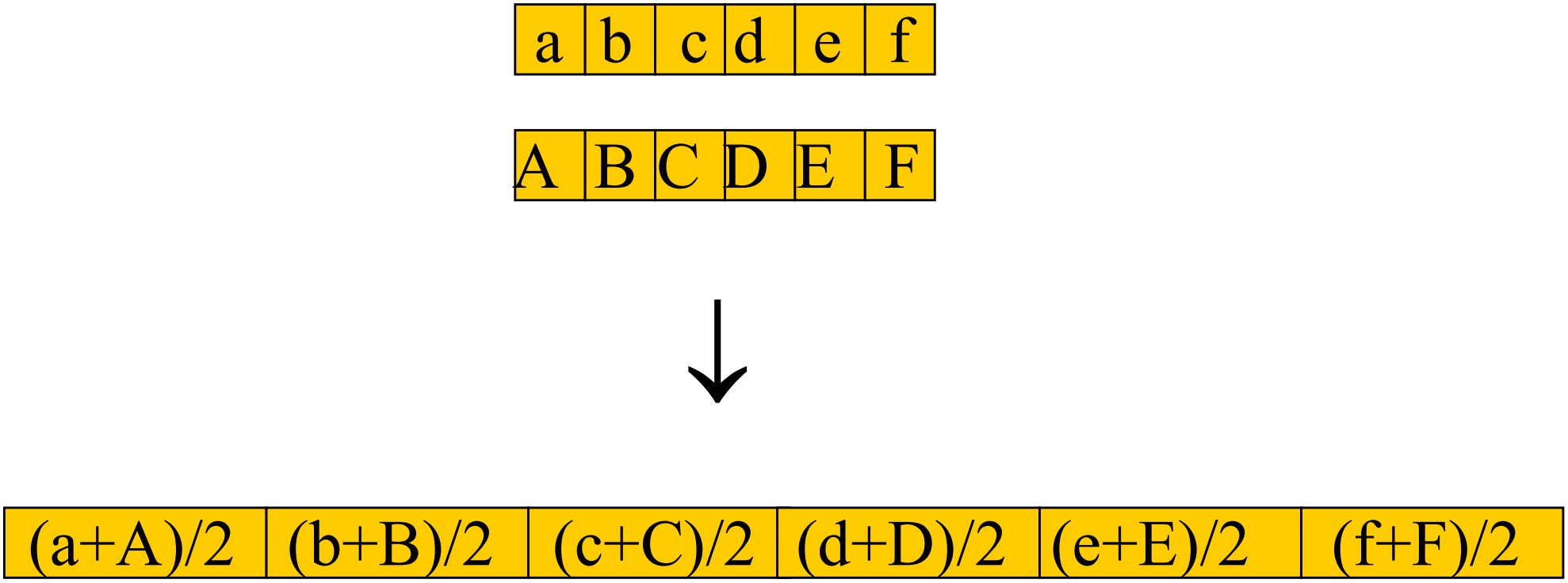
Exemple : croisement pour une représentation réelle

Exemple de croisement uniforme pour une représentation discrète.

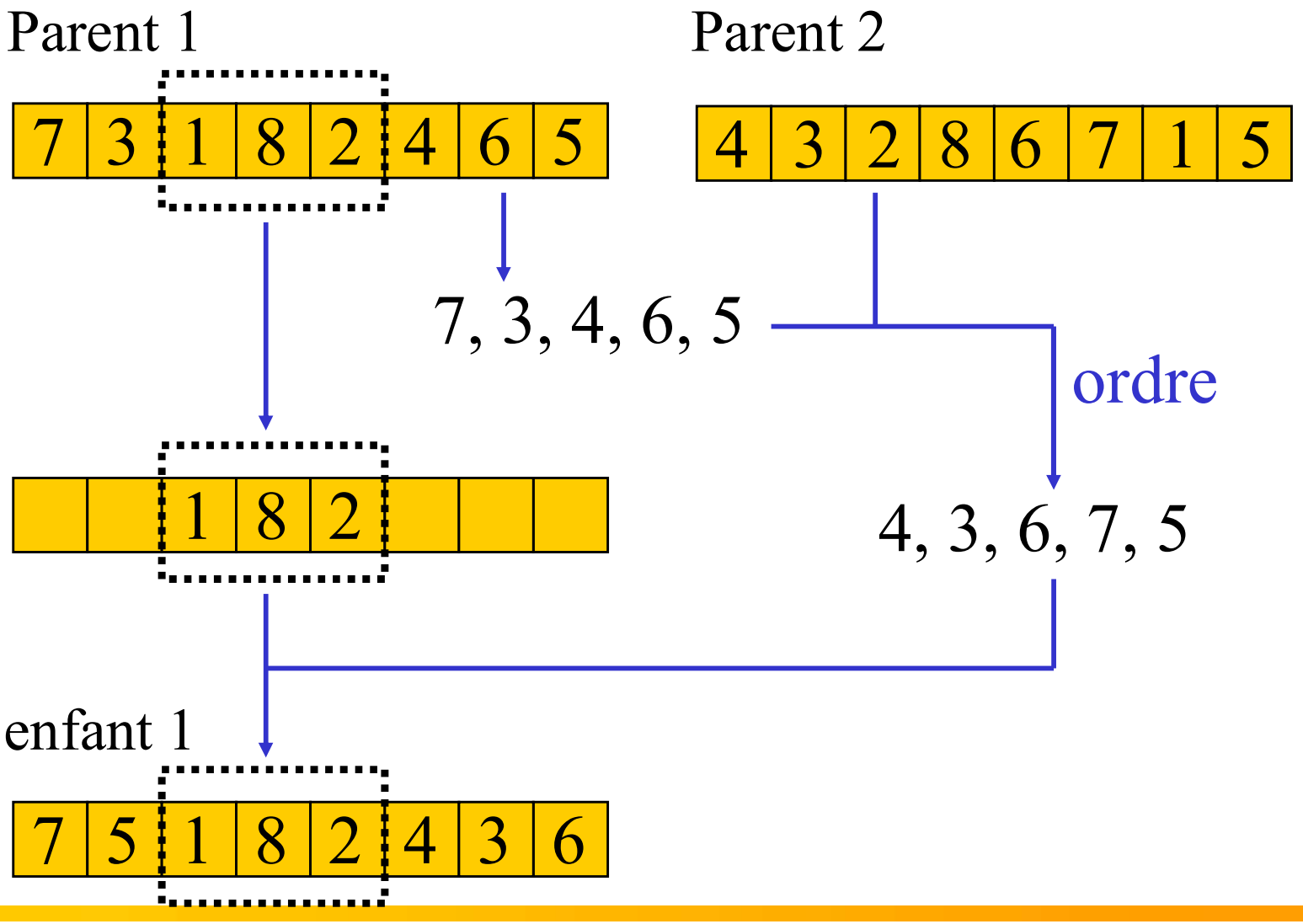


Exemple : croisement pour une représentation réelle (2)

Exemple de croisement arithmétique :



Exemple : croisement pour une représentation ordonnée



Opérateurs de mutation

Nous avons une ou plusieurs mutations pour la représentation choisie :

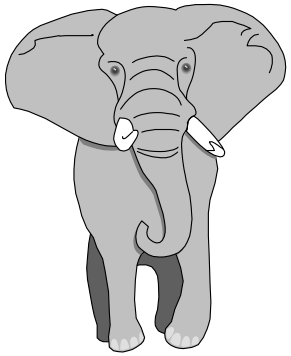
- au moins une mutation doit garantir que tout point de l ’espace est atteignable
- la taille de la mutation est importante et doit être contrôlable
- la mutation doit produire des chromosomes valides
- attention au biais du à l ’introduction d’heuristiques

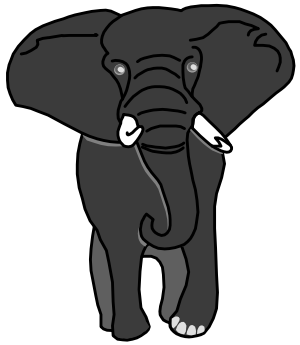
Exemple : mutation pour une représentation discrète

avant 1 1 1 1 1 1 1

après 1 1 1 0 1 1 1

Gène muté

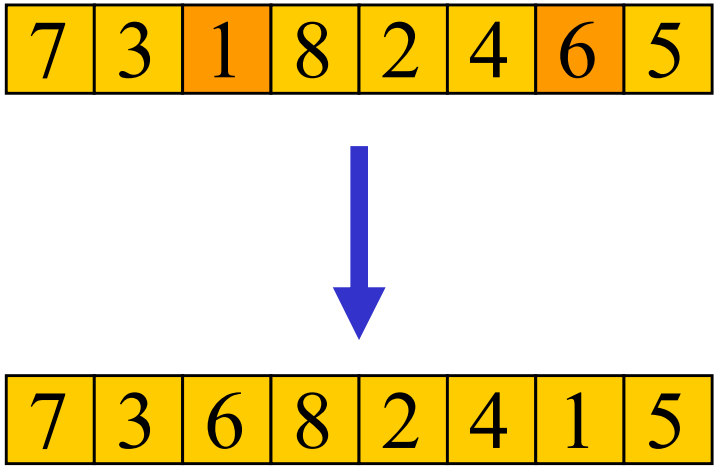




La mutation a généralement lieu avec une probabilité p_m pour chaque gène.

Exemple : mutation pour une représentation ordonnée

Sélectionner au hasard deux gènes et les échanger.



Stratégie de remplacement

La pression sélective est aussi affectée par le choix des individus destinés à être remplacés par les individus nouvellement produits.

- Utilisation de méthodes stochastiques ou au contraire de stratégies de remplacement déterministes.

- Possibilité de ne jamais remplacer le meilleur individu de la population : élitisme.

Critère d ’arrêt

- Optimum atteint

- Limite en ressources CPU :

nombre maximum d ’évaluations

- Limite fixée par l ’utilisateur :

nombre donné de générations sans améliorations

Performances de l ’algorithme

Ne pas tirer de conclusions à partir d ’une exécution unique :

- utiliser des mesures statistiques sur un nombre suffisant d ’exécutions.
- tester l ’algorithme sur des données réelles.

Problématiques principales

Diversité génétique

- convergence prématurée et atteinte d ’un optimum local.

Equilibre entre exploration et exploitation

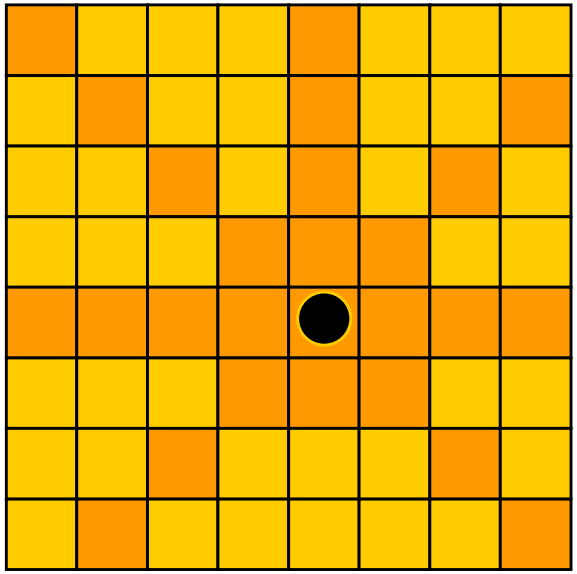
exploration : parcourir des régions inconnues

- trop d ’exploration : recherche aléatoire, pas de convergence.

exploitation : améliorer les meilleurs individus actuels.

- trop d ’exploitation : recherche locale seulement, convergence vers un optimum local.

Exemple : les 8 reines

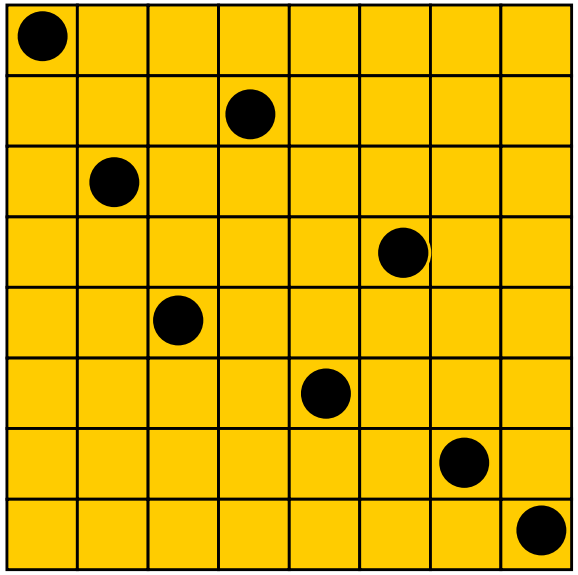


Problème: placer 8 reines sur un échiquier 8x8 sans que celles-ci puissent se prendre mutuellement.

8 reines : représentation

Génotype : permutation
de nombres entre 1 et 8

1	3	5	2	6	4	7	8

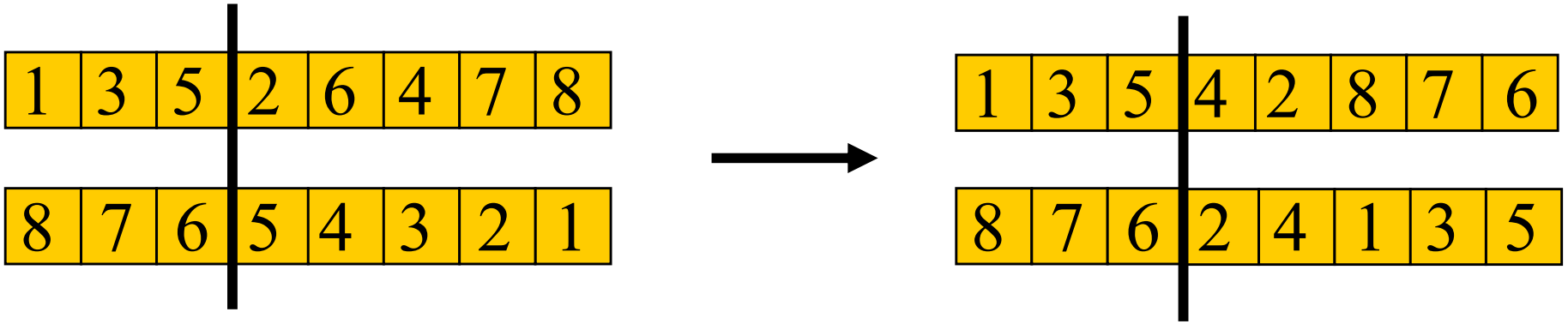


Génotype : configuration

8 reines : représentation (2)

Mutation : echange de 2 nombres

Croisement : combinaison de 2 parents



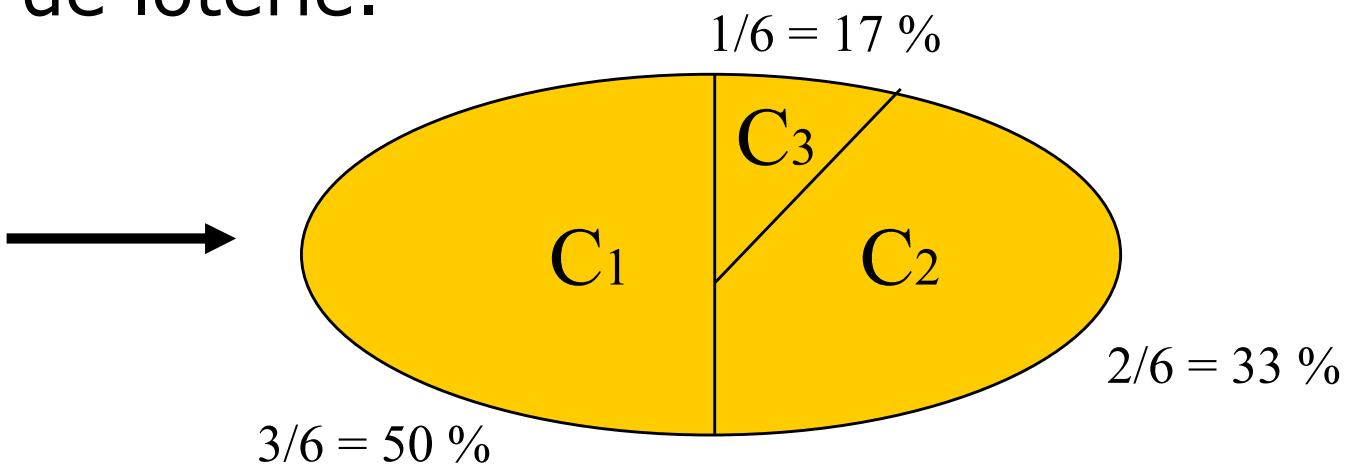
8 reines : fitness et sélection

Fitness : la pénalité d ’une reine est égale au nombre de reines qu’elle peut prendre.

La fitness d ’une configuration est égale à la somme des pénalités de toutes les reines.

Sélection par roue de loterie.

Fitness(C_1) = 1
Fitness(C_2) = 2
Fitness(C_3) = 3



Domaines d ’application

- Ordonnancement
- Transport, logistique
- Apprentissage, identification de modèles, de paramètres
- Contrôle commande et planification
- Ingénierie, Design
- Electronique
- Data Mining
- …