Agent de résolution de problèmes

Recherche loclale

Aziz KHAMJANE

Plan

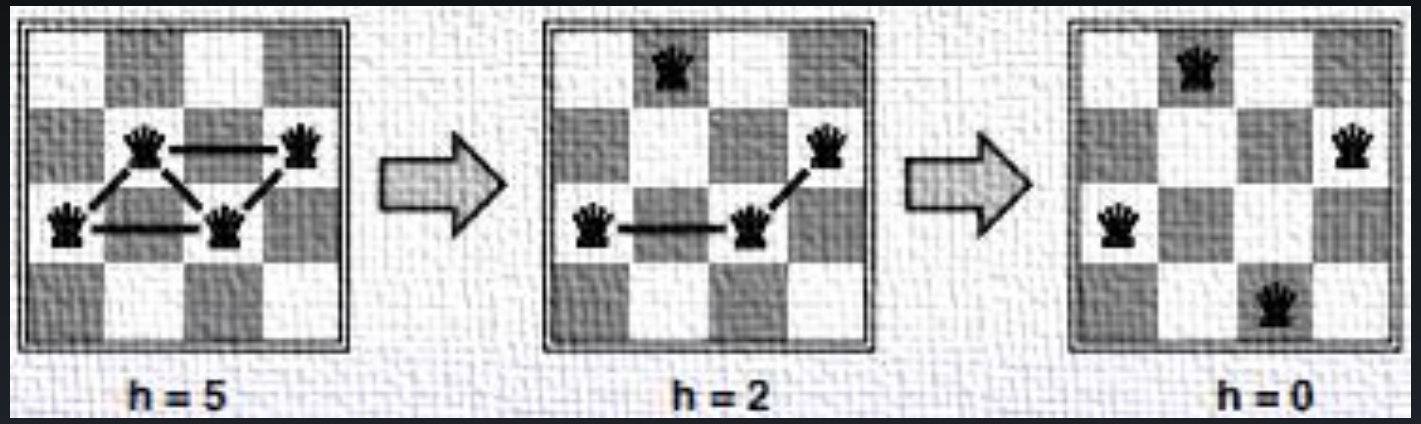
- Introduction
- Principe des algorithmes de recherche locale
- Hill-climbing
- Variants de hill-climbing
- Recuit simulé (simulated annealing)
- Recherche locale par faisceau
- Algorithmes génétiques

Introduction

- Dans de nombreux problèmes d'optimisation, le chemin qui mène vers une solution n'est pas important.
- L'état lui-même est la solution.
- Idée : Modifier l'état en l'améliorant au fur et à mesure.
- Espace d'états : ensemble des configurations possible des états.
- Besoin de définir une fonction qui mesure l'utilité d'un état.

Exemple : les n reines

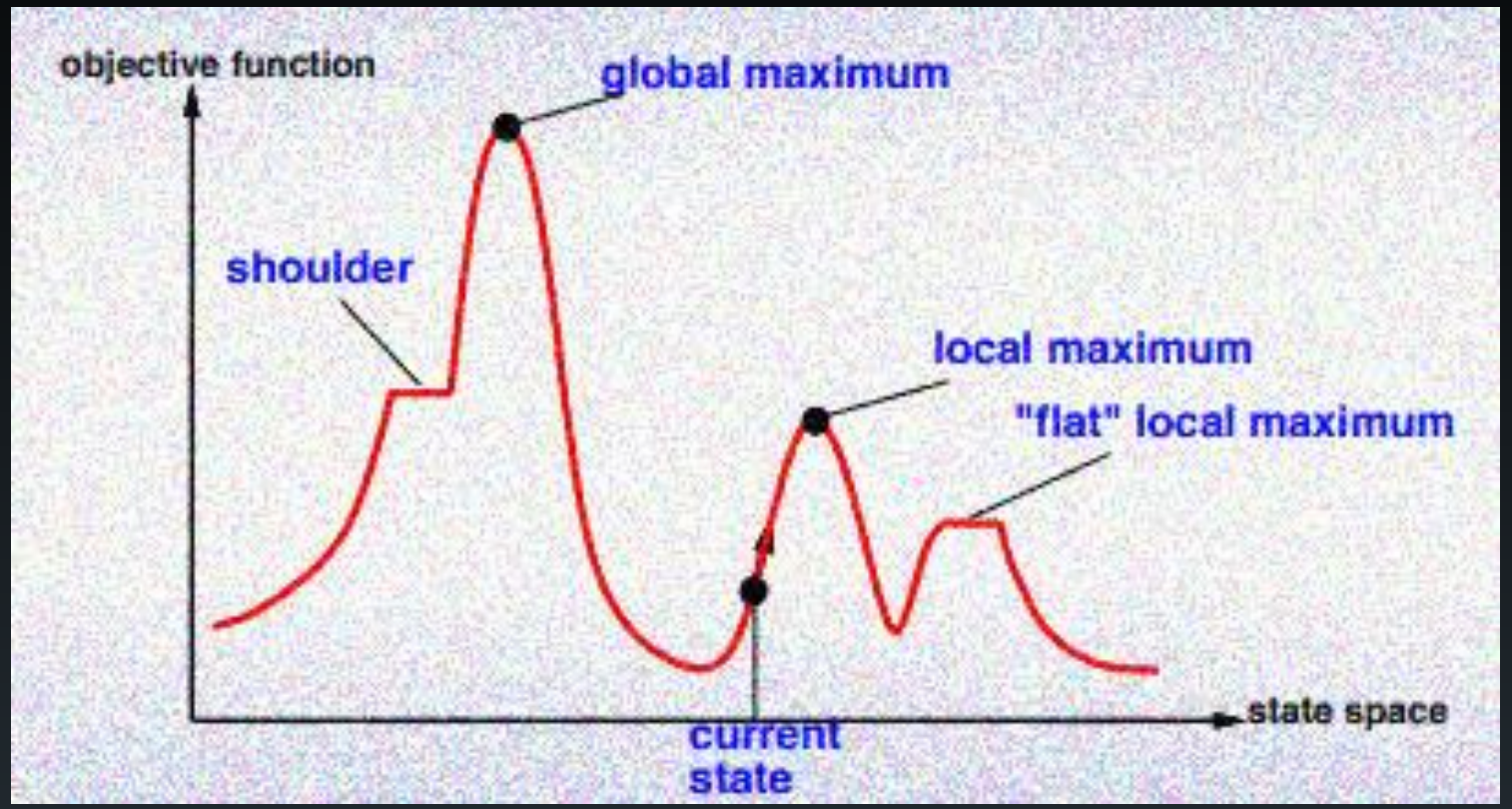
- Placer n reines sur un échiquier de taille $n \times n$, sans que deux reines se trouvent sur la même ligne, colonne ou diagonale.
- Déplacer une reine pour réduire le nombre de conflits.



- Avec N=4 : 256 configurations.
- N=8 : 16 777 216
- N= 16 : 18,446,744,073,709,551,616 configurations

Algorithmes de recherche locale

On cherche un maximum global



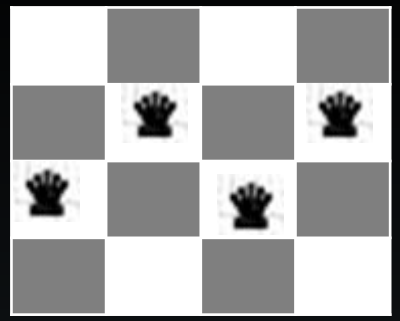
Principe d'une recherche locale

- Une recherche locale garde juste certains états visités en mémoire:
  - Le cas le plus simple est hill-climbing qui garde juste **un état** (l'état courant) et l'améliore itérativement jusqu'à converger à une solution.
  - Le cas le plus élaboré est celui des algorithmes génétiques qui gardent **un ensemble d'états** (appelé population) et le font évoluer jusqu'à obtenir une solution.
- En général, il y a une fonction objective à optimiser (maximiser ou minimiser)
  - Dans le cas de hill-climbing, elle permet de choisir l'état successeur.
  - Dans le cas des algorithmes génétiques, on l'appelle la fonction de fitness.

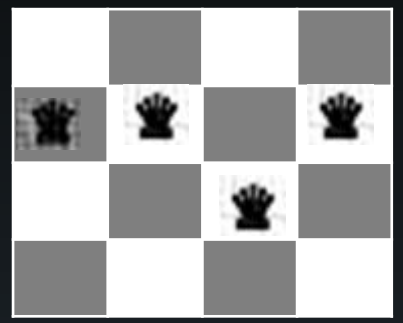
Méthode Hill-Climbing

- Entrée :
  - État initial.
  - Fonction à optimiser:
    - ✓ noté VALUE dans l'algorithme;
    - ✓ parfois noté *h* aussi.
- Méthode
  - Le nœud courant est initialisé à l'état initial.
  - Itérativement, le nœud courant est comparé à ses successeurs immédiats.
    - Le meilleur voisin immédiat et ayant la plus grande valeur (selon VALUE) que le nœud courant, devient le nœud courant.
    - Si un tel voisin n'existe pas, on arrête et on retourne le nœud courant comme solution.

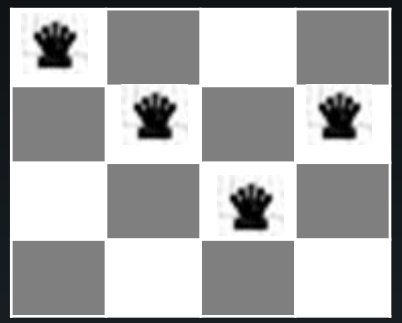
Hill-Climbing avec 8 reines



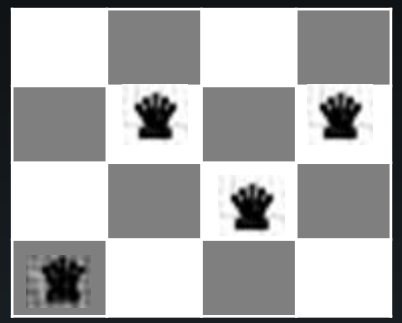
$h = 5$



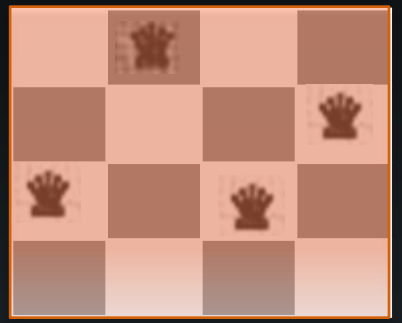
$h = 5$



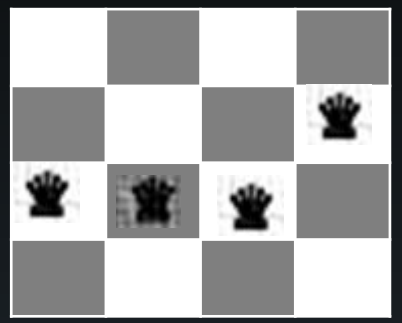
$h = 5$



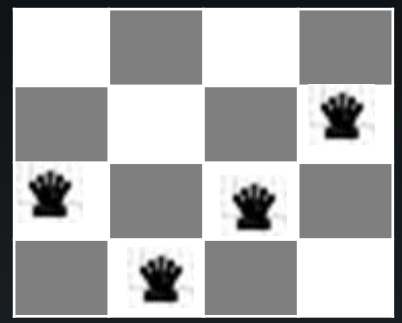
$h = 3$



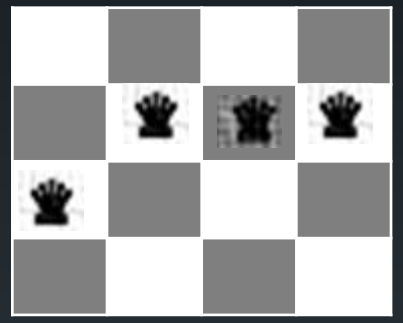
$h = 2$



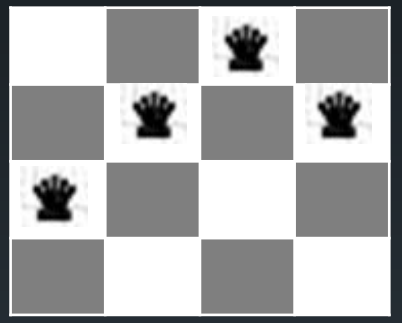
$h = 4$



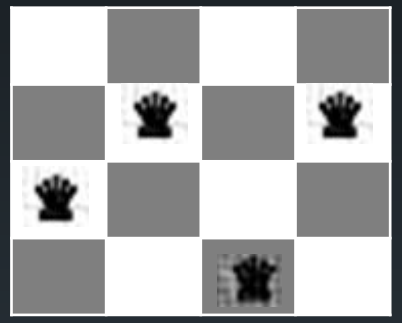
$h = 5$



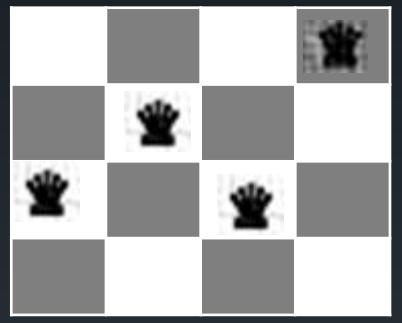
$h = 4$



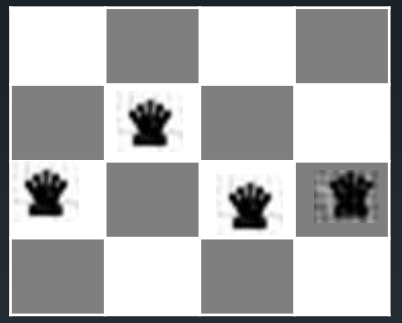
$h = 5$



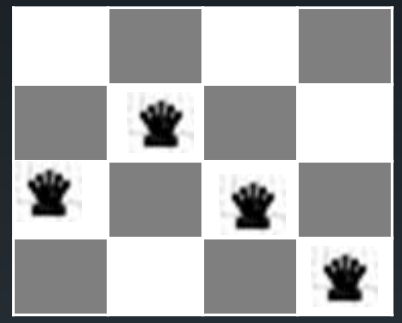
$h = 2$



$h = 4$

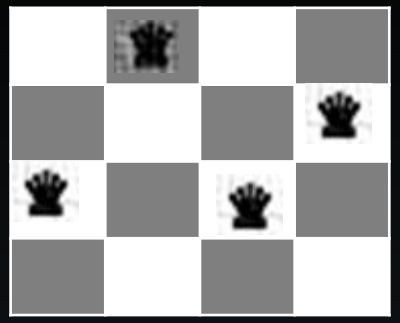


$h = 5$



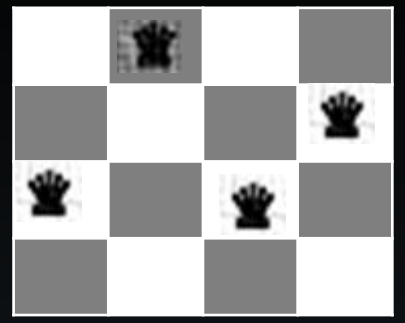
$h = 5$

Hill-Climbing avec 8 reines

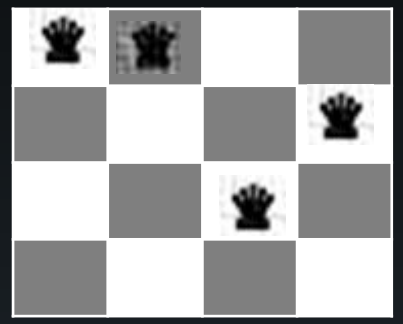


$h = 2$

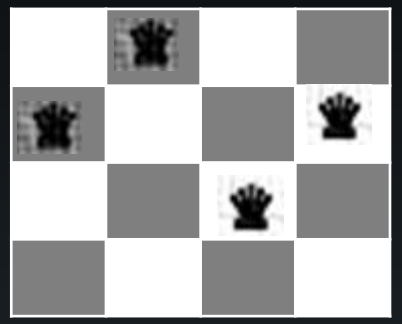
Hill-Climbing avec 8 reines



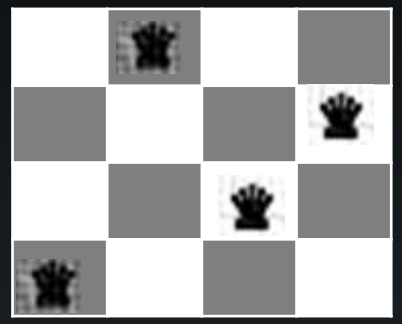
$h = 2$



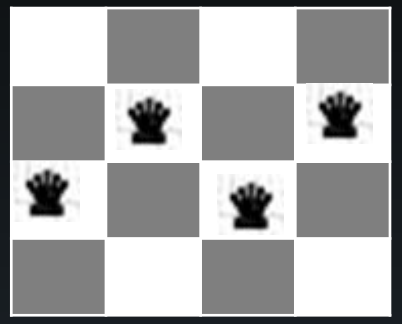
$h = 3$



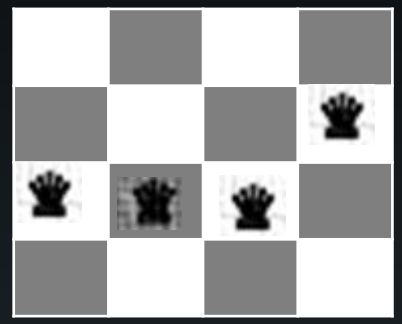
$h = 3$



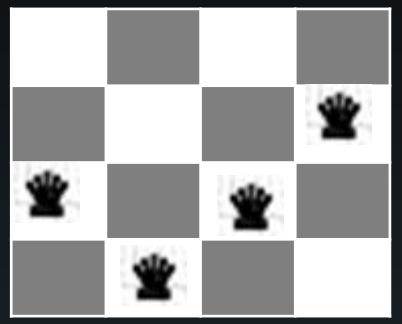
$h = 1$



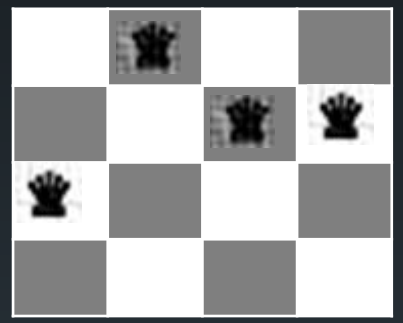
$h = 5$



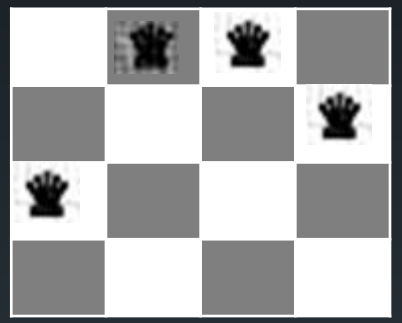
$h = 4$



$h = 5$

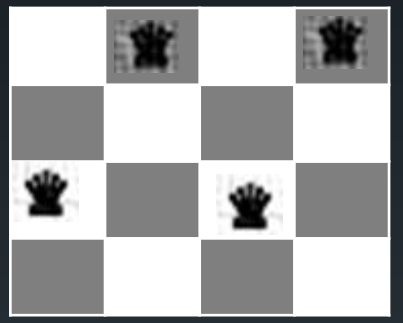


$h = 2$

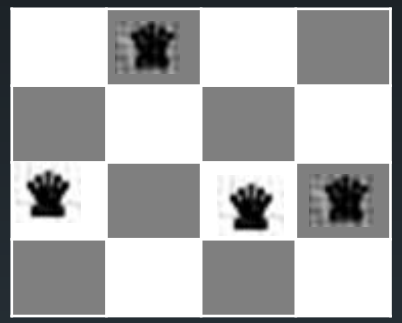


$h = 2$

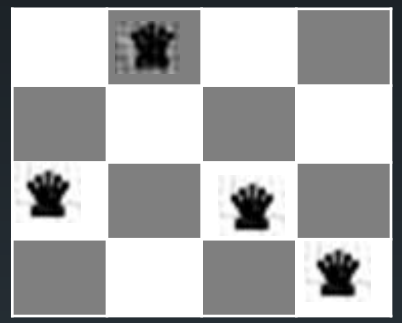
$h = 0$



$h = 2$



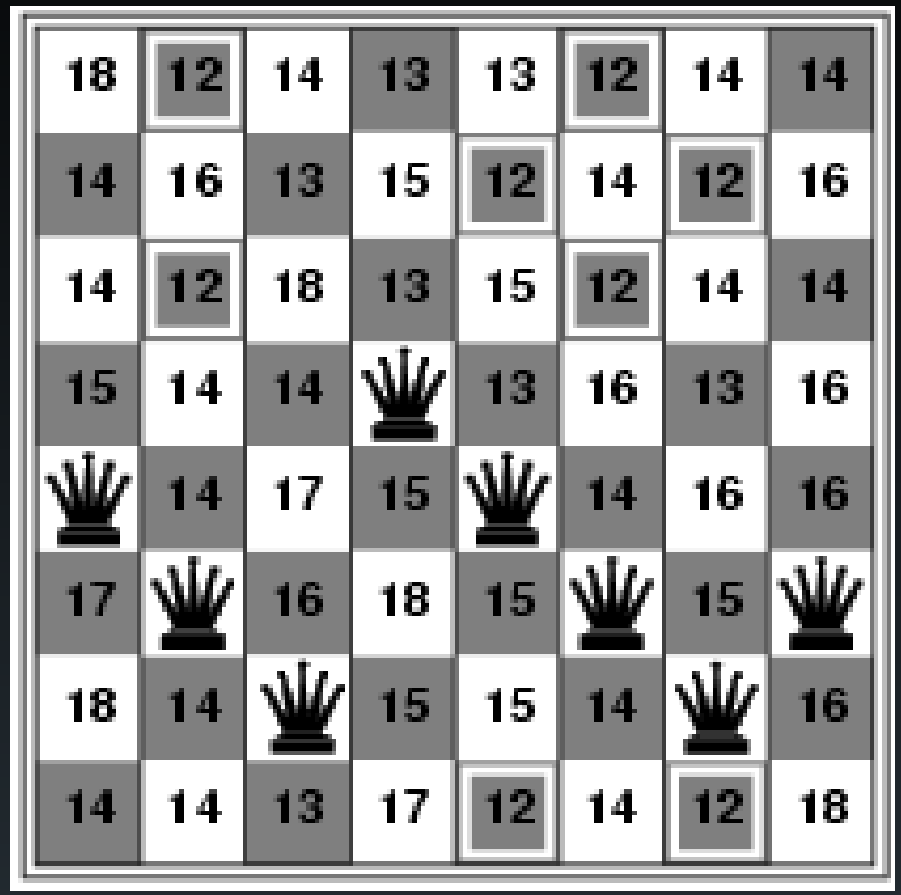
$h = 3$



$h = 2$

Hill-Climbing avec 8 reines

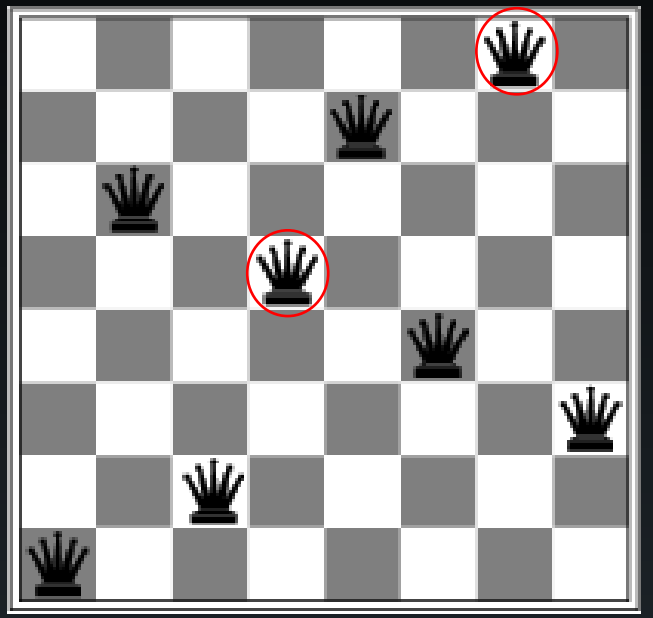
- h (VALUE): nombre de paires de reines qui s'attaquent mutuellement directement ou indirectement.
- On veut le minimiser.



- pour l'état affiché: h=17
- Encadrés: les meilleurs successeurs.

Hill-Climbing avec 8 reines

- Un exemple de minimum local avec h(n)=1



À partir d'un état généré aléatoirement, hill climbing est bloquée dans 86% des cas. Elle réussit dans 14% des cas. Elle fonctionne rapidement, en prenant juste 4 pas en moyenne quand il réussit et 3 quand il est bloqué –

- pas mal pour un espace d'états avec $N = 8^8 \approx 17\ millions\ d'états$.
- Solution 1 : autoriser les mouvements latéraux
  - Il faut fixer le nombre de mouvements possibles
  - Pour le problème de 8-reines, l'autorisation des mouvements latéraux permet d'améliorer le taux de réussite de hill climbing de 14% à 94%

Variants de hill-climbing

- **Hill-climbing stochastique :**
  - Choisir aléatoirement parmi les états successeurs qui améliore la situation. la probabilité de sélection peut varier en fonction de l'amélioration.
  - Cette version converge généralement plus lentement, mais dans certains espaces d'états, elle trouve de meilleures solutions.
- **First-choice hill climbing :**
  - Générer des successeurs de manière aléatoire jusqu'à ce qu'un soit généré soit meilleur que l'état actuel.
  - C'est une bonne stratégie lorsqu'un état a plusieurs (par exemple, des milliers) successeurs.
- **Random-restart hill climbing :**
  - Répéter plusieurs hill-climbing avec plusieurs états initiaux générés aléatoirement
  - Si la probabilité de réussite de l'algorithme Hill-climbing est de 1/q, il faut répéter l'algorithme au moins q fois.

Méthode *simulated annealing* (recuit simulé)

- C'est une amélioration de l'algorithme hill-climbing pour minimiser le risque d'être piégé dans des maxima/minima locaux.
  - au lieu de regarder le meilleur voisin immédiat du nœud courant, on va regarder, avec une certaine probabilité, un moins bon voisin immédiat.
    - on espère ainsi s'échapper des optima locaux
  - au début de la recherche, la probabilité de prendre un moins bon voisin est plus élevée et diminue graduellement.
- Le nombre d'itérations et la diminution des probabilités sont définis à l'aide d'un schéma de « températures », en ordre décroissant
  - ex.: schéma [2^{-0}, 2^{-1}, 2^{-2}, 2^{-3}, ... , 2^{-99}], pour un total de 100 itérations
  - la meilleure définition du schéma va varier d'un problème à l'autre

Méthode *simulated annealing* (recuit simulé)


```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling the probability of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next] – VALUE[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^(ΔE/T)
```

Exploration locale par faisceau (local beam search)

- Conserver k états au lieu d'un seul état.
  Commencer par k états générés de manière aléatoire

  - A chaque étape, tous les successeurs des k états sont générés
  - Si l'un d'eux est un but, l'algorithme s'arrête.
  - Sinon, il sélectionne les k meilleurs successeurs parmi la liste complète des successeurs et recommence.

Algorithmes génétiques

- Un algorithme génétique est une variante de la recherche de faisceau stochastique dans laquelle les états successeurs sont générés en combinant deux états parents,
- Les algorithmes génétiques utilisent la théorie de Darwin sur l’évolution des espèces.
- Elle repose sur trois principes : le principe de *variation*, le principe *d'adaptation* et le principe *d'héritage*.

Algorithmes génétiques

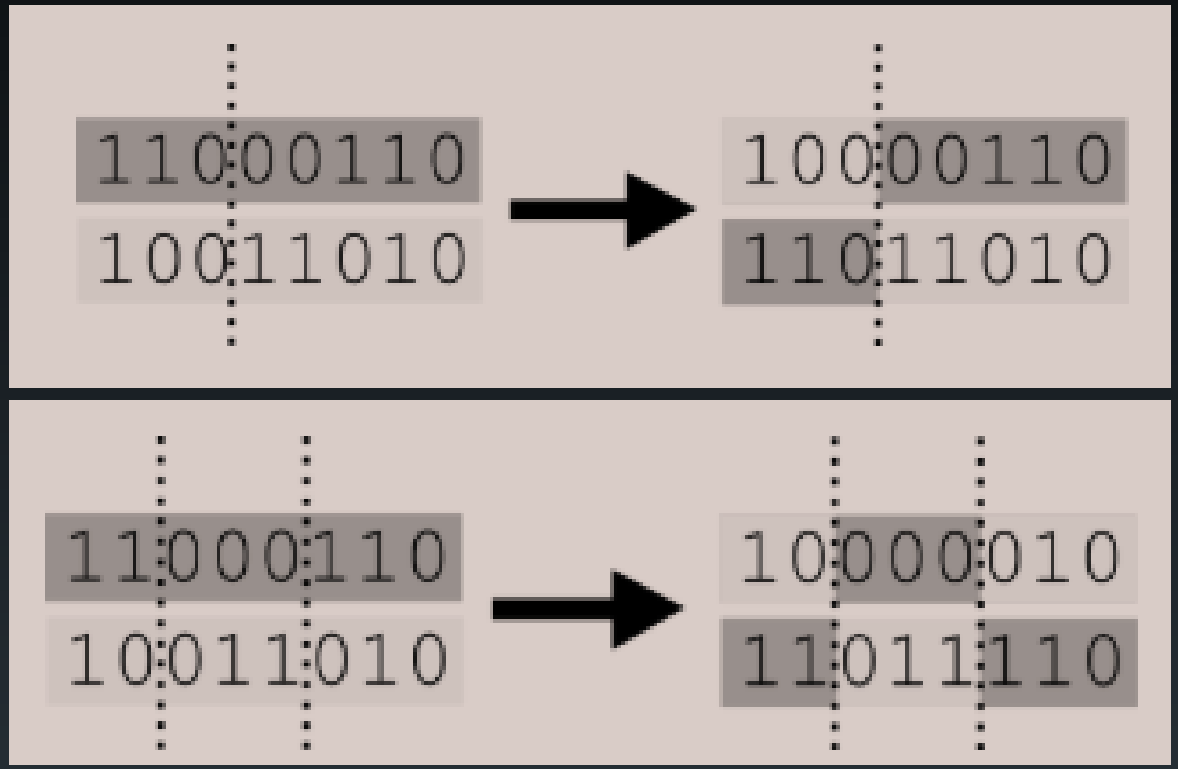
- On représente l'espace des solutions d'un problème à résoudre par une *population* (ensemble de *chromosomes*).
  - Un *chromosome* est une chaîne de *gènes* de taille fixe.
  - Par exemple : 101101001
- Une population génère des enfants par un ensemble de procédures simples qui manipulent les chromosomes
  - *La sélection* **:** Choix des individus les mieux adaptés.
  - *Le croisement* **:** Mélange par la reproduction des particularités des individus choisis.
  - *La mutation :* Altération aléatoire des particularités d'un individu.
- Les enfants sont conservés en fonction de leur *adaptabilité* (*fitness*) déterminée par une fonction d'adaptabilité donnée, *f(x).*

Algorithmes génétiques : Sélection

- *Sélection*
- La sélection consiste à choisir les individus les mieux adaptés.
- Il existe plusieurs techniques de sélection:
  - *Sélection par rang* : Cette technique de sélection choisit toujours les individus possédant les meilleurs scores d'adaptation.
  - *Probabilité de sélection proportionnelle à l'adaptation :* pour chaque individu, la probabilité d'être sélectionné est proportionnelle à son adaptation au problème.
  - *Sélection uniforme* : La sélection se fait aléatoirement, uniformément et sans intervention de la valeur d'adaptation.

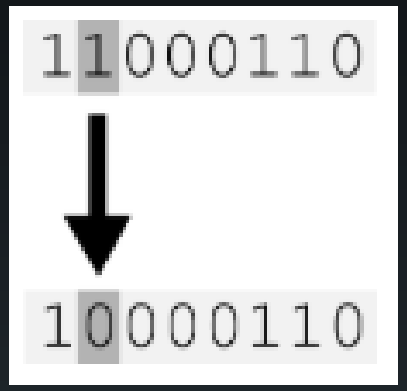
Algorithmes génétiques : Croisement

- *Croisement*
- Le croisement, crossing-over, est le résultat obtenu lorsque deux chromosomes partagent leurs particularités.



Algorithmes génétiques : Mutation

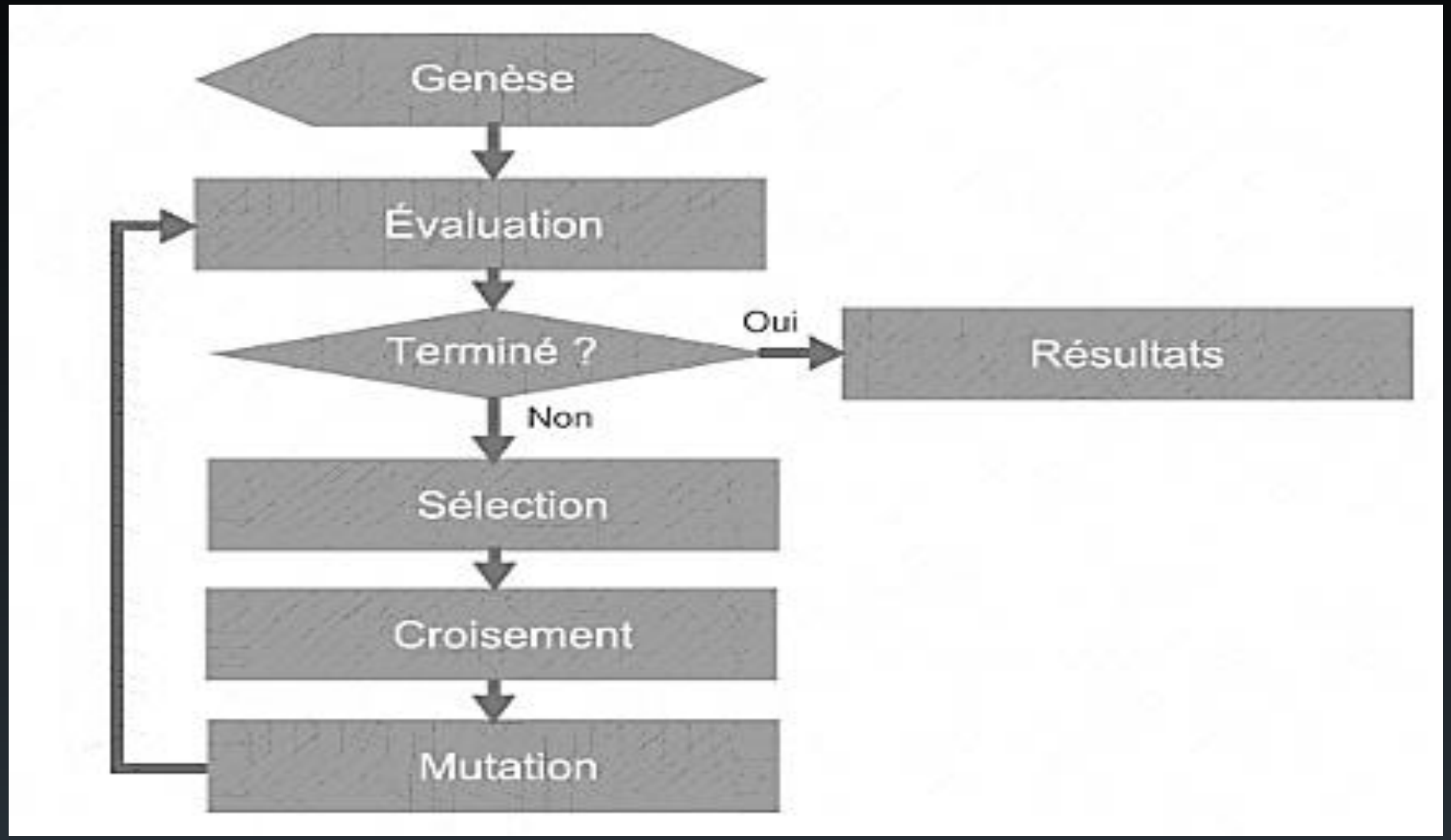
- *Mutation*
- La mutation consiste à altérer un gène dans un chromosome selon un facteur de mutation. Ce facteur est la probabilité qu'une mutation soit effectuée sur un individu.



Algorithmes génétiques

1. Générer aléatoirement la population initiale $P(t)$
2. Evaluer $P(t)$
3. **TantQue** (Critère d'arrêt non atteint) **Faire**
4. $P(t+1) \leftarrow$ Sélection des parents dans $P(t)$
5. $P(t+1) \leftarrow$ Appliquer Croissement et Mutation sur $P(t+1)$
6. $P(t+1) \leftarrow$ Remplacer P(t) par $P(t+1)$
7. Evaluer $P(t)$
8. **FinTantQue**

Algorithmes génétiques



Exemple 1

- Trouver l'entier x qui maximise la fonction : $f(x) = 15x - x^2$
- Supposons que $x \in [0, 15]$:
  - on a besoin de seulement 4 bits pour représenter les individus de la population.

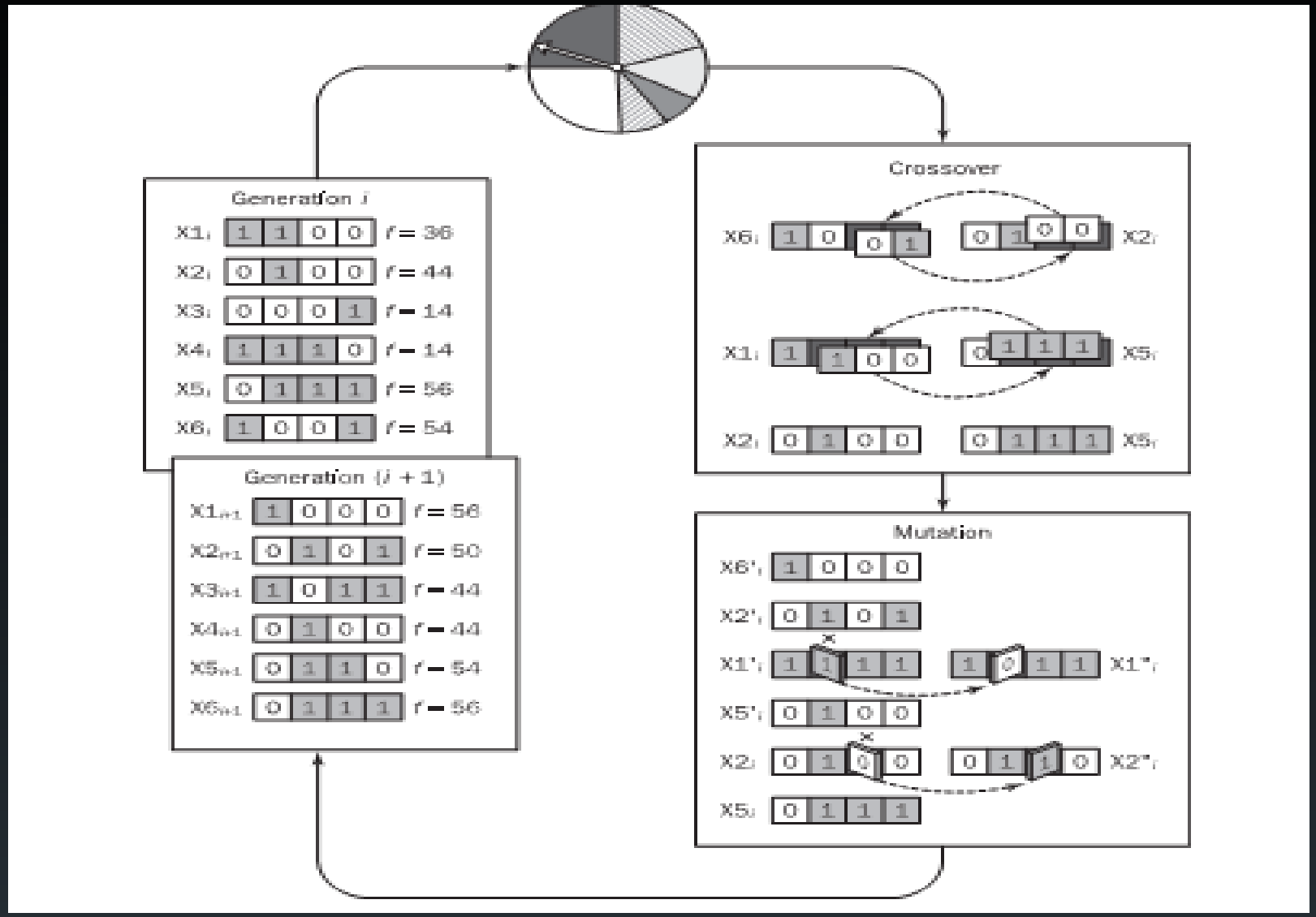
Integer	Binary code	Integer	Binary code	Integer	Binary code
1	0 0 0 1	6	0 1 1 0	11	1 0 1 1
2	0 0 1 0	7	0 1 1 1	12	1 1 0 0
3	0 0 1 1	8	1 0 0 0	13	1 1 0 1
4	0 1 0 0	9	1 0 0 1	14	1 1 1 0
5	0 1 0 1	10	1 0 1 0	15	1 1 1 1

Exemple 1 (suite)

- Fixons la taille de la population à 6.
- La probabilité de mutation à 0.001.
- La fonction d'adaptabilité à $f(x) = 15x - x^2$.
- L'algorithme génétique initialise les 6 chromosomes de la population en les choisissant au hasard.

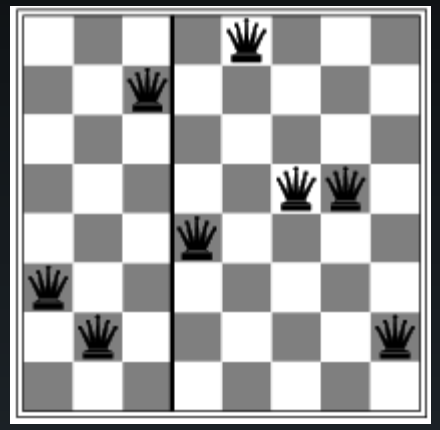
Chromosome label	Chromosome string	Decoded integer	Chromosome fitness	Fitness ratio, %
X1	1 1 0 0	12	36	16.5
X2	0 1 0 0	4	44	20.2
X3	0 0 0 1	1	14	6.4
X4	1 1 1 0	14	14	6.4
X5	0 1 1 1	7	56	25.7
X6	1 0 0 1	9	54	24.8

Exemple 1


Generation i
X1i 1 1 0 0 f = 36
X2i 0 1 0 0 f = 44
X3i 0 0 0 1 f = 14
X4i 1 1 1 0 f = 14
X5i 0 1 1 1 f = 56
X6i 1 0 0 1 f = 54
Generation (i + 1)
X1i+1 1 0 0 0 f = 56
X2i+1 0 1 0 1 f = 50
X3i+1 1 0 1 1 f = 44
X4i+1 0 1 0 0 f = 44
X5i+1 0 1 1 0 f = 54
X6i+1 0 1 1 1 f = 56
Crossover
X6i 1 0 0 1
X2i 0 1 0 0
X1i 1 1 0 0
X5i 0 1 1 1
X2i 0 1 0 0
X5i 0 1 1 1
Mutation
X6'i 1 0 0 0
X2'i 0 1 0 1
X1'i 1 1 1 1
X1"i 1 0 1 1
X5'i 0 1 0 0
X2i 0 1 0 0
X2"i 0 1 1 0
X5i 0 1 1 1

Exemple 2: problème de 8-reines

On peut représenter le chromosome par un nombre de 8 chiffres, chacun allant de 1 à 8, indiquant la ligne occupée par la reine dans chaque colonne.

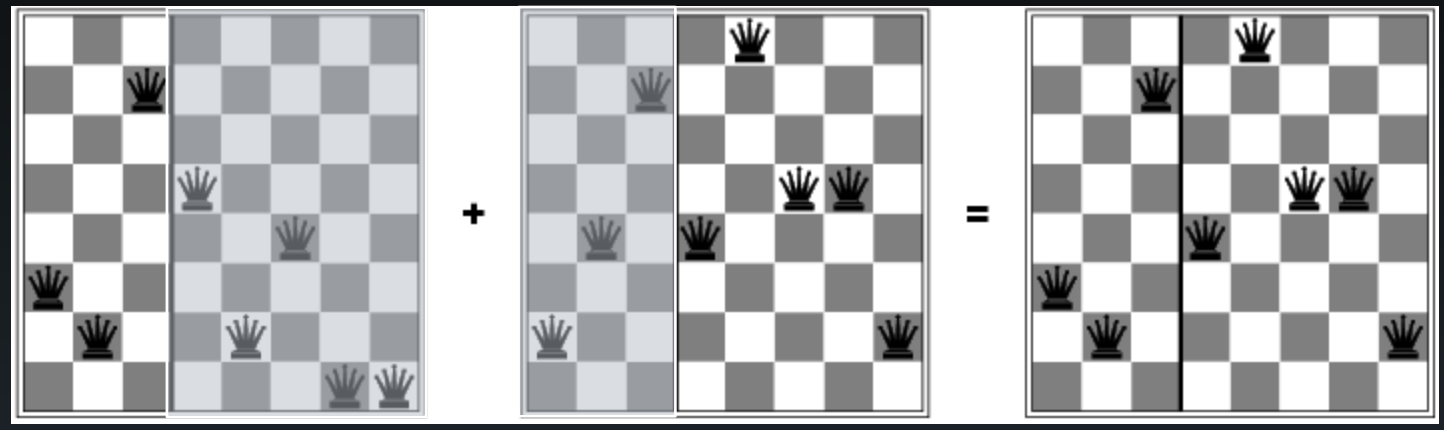


67251448

Fonction de fitness: nombre de pairs de reines qui ne s'attaquent pas (min = 0, max = (8 × 7)/2 = 28)

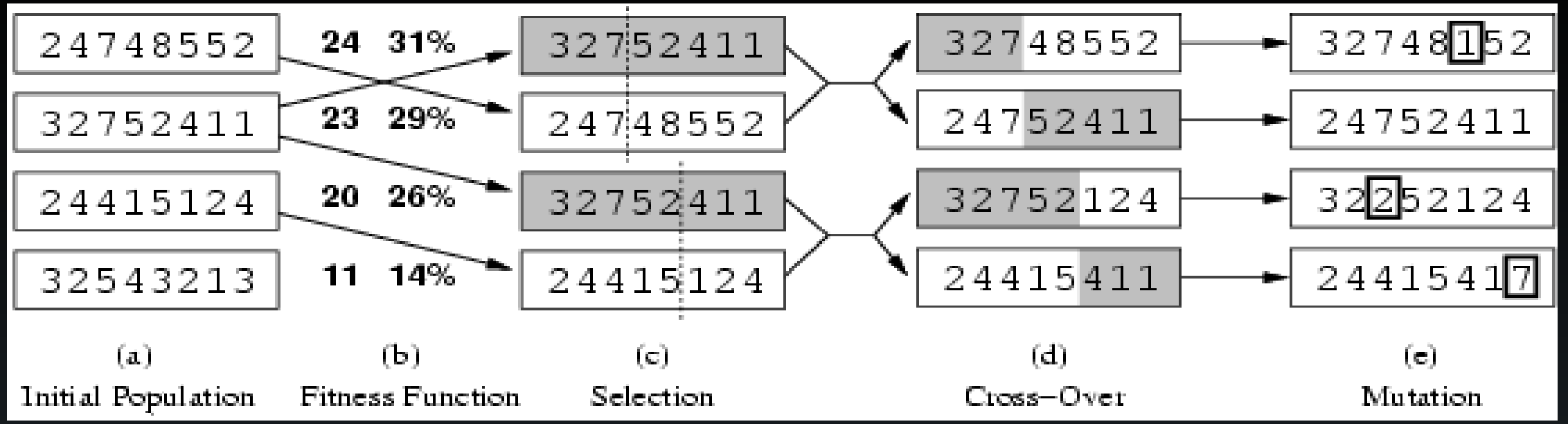
Exemple : 8 reines

Croisement



67247588 + 75251448 = 67251448

Exemple avec 8 reines



- Pourcentage de fitness (c-à-d., probabilité de sélection du chromosome):
  - 24/(24+23+20+11) = 31%
  - 23/(24+23+20+11) = 29%
  - 20/(24+23+20+11) = 26%
  - 11/(24+23+20+11) = 14%

Recherche local vs Recherche global