CHAPITRE 5: MÉTHODES APPROCHÉES, HEURISTIQUES SPÉCIALISÉES

COURS : COMPLEXITÉ ET OPTIMISATION

RÉALISÉ Par : DR. S. SLATNIA

UNIVERSITÉ DE BISKRA
MASTER D'INFORMATIQUE
2021-2022

MÉTHODES APPROCHÉES, ALGORITHME GLOUTON

Algorithme Glouton ou avide (en anglais, **Greedy**)

- Pour les problèmes de grandes tailles, pas de temps de calculs raisonnables avec les méthodes exactes.

- Il faut donc, rechercher de bonnes solutions approchées.

- Un algorithme Glouton se déroule selon les phases à chaque phase : on espère que choisir un optimum local à chaque phase convergera vers une solution optimale globale
  - Une solution locale → une solution globale.

- Un algorithme avide emploie de simples stratégies, faciles à implanter et qui requièrent un minimum de ressources.

- Construction d'une solution réalisable en se ramenant à une suite de décisions qu'on prend à chaque fois au mieux en fonction d'un critère local sans remettre en question les décisions déjà prises, la solution obtenue est approchée.

❖ **Avantage** : algorithmes simples à implémenter.
❖ **Inconvénients** : solutions approchées obtenues plus ou moins bonnes, critère local.

❑ **Exemple** : Placement optimal de pièces 2D (Bin Packing).

On dispose de plaques rectangulaires toutes identiques dans lesquelles on veut placer des pièces rectangulaires sans chevauchement. Les pièces à placer ont des dimensions différentes.

❖ On veut trouver le placement pour minimiser le nombre de plaques utilisées.

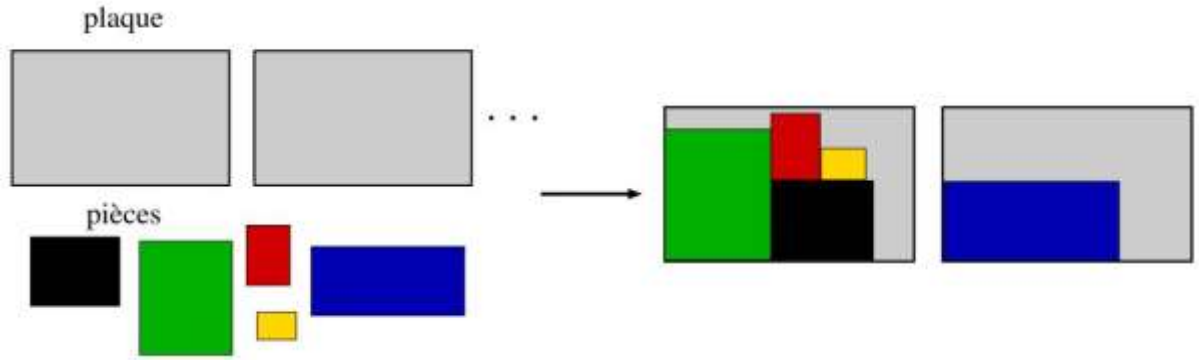

Figure. Exemple d'un placement optimal de pièces 2D (Bin Packing)

❖ Algorithme glouton : trier les pièces en fonction de leur taille et placer d'abord les pièces les plus grandes.

❑ Quelques Méthodes heuristiques

Heuristiques

Une heuristique est une technique qui améliore l'efficacité d'un processus de recherche, en sacrifiant éventuellement l'exactitude ou l'optimalité de la solution.

- L'optimalité est difficile (coût exponentiel), on peut se contenter sur :
  - ➢ Une solution satisfaisante donnée par une heuristique avec un coût plus faible
  - ➢ La solution non optimale mais une solution approchée.

- **Qu'est-ce qu'une heuristique**

• Dans divers domaines, dont en informatique, une **heuristique** est une méthode (~algorithme) qui calcule **rapidement** (ex: en temps constant, linéaire ou polynomiale) une solution pouvant être approximative et incomplète à un problème généralement trop complexe [38].

Algorithme de recherche en IA / Fonction heuristique [38]

- Estimation de la distance (coût restant) entre un état **n** et un but **g.**
- Le but **g** peut être implicite.
- Généralement notée **h** ou **h(n).**
  - **h(n)** : estime le coût restant pour atteindre le but implicite **g** à partir du nœud (état) **n.**
- Exemple de fonctions heuristiques pour la navigation dans sur une carte :
  - Distance euclidienne (aussi appelée à vol d'oiseau).
  - Distance de Manhattan (ville quadrillée).

HEURISTIQUE, ALGORITHME RECHERCHE LOCALE

- Partir d'une solution sinon approchée du moins potentiellement bonne et d'essayer de l'améliorer itérativement [35].
- Pour améliorer une solution on ne fait que de légers changements

  On parle de changement local (solution voisine).
- Relancer la méthode plusieurs fois en changeant le point de départ pour avoir plus de couverture.

HEURISTIQUE, ALGORITHME RECHERCHE LOCALE

```
A* ← creer une solution()
for all t=1 a max essais do
      A ← nouvelle solution()
      for all m=1 a max mouvements do
            A' ← choisir voisin(A)
            d ← (f(A')-f(A))
            if acceptable(d) then   A ← A'
      end for
      if f(A*)>f(A) then  A* ← A
end for
retourner A*,f(A*)
f : fonction à optimiser
```

HEURISTIQUE, ALGORITHME RECHERCHE LOCALE

L'insuffisance des méthodes de recherche locale

- incapables de progresser au-delà du premier optimum local rencontré

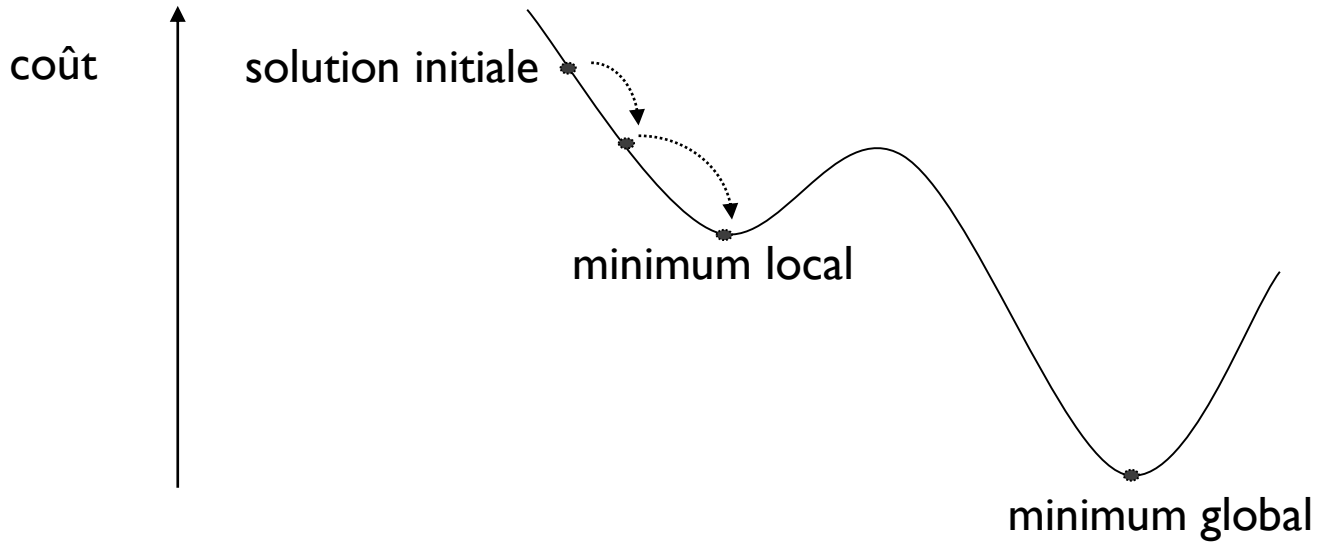

Fig. Problème de minimisation

HEURISTIQUE, ALGORITHME A*

- L'algorithme A* a été créé pour que la première solution trouvée soit l'une des meilleures. Cet algorithme a été proposé pour la première fois par Peter E. Hart (en) [wiki], Nils John Nilsson et Bertram Raphael en 1968. Il s'agit d'une extension de l'algorithme de Dijkstra de 1959. Il s'agit d'un algorithme heuristique de programmation dynamique qui fournit généralement une solution approchée [36].

- L'algorithme A* est un algorithme de recherche de chemin dans un graphe entre un nœud initial et un nœud final. Il utilise une **évaluation heuristique** sur chaque nœud pour estimer le meilleur chemin y passant, et visite ensuite les nœuds par ordre de cette évaluation heuristique.
  - **Algorithme simple,**
  - **Ne nécessitant pas de prétraitement,** et
  - **Ne consommant que peu de mémoire.**

- Garantit de toujours trouver le chemin le plus court à un but s'appelle « **algorithme admissible** ».
- On peut démontrer que A* ne considère pas plus de nœuds que tous les autres algorithmes admissibles de recherche, à condition que l'algorithme alternatif n'ait pas une évaluation heuristique plus précise. Dans ce cas, A* est l'algorithme informatique le plus efficace garantissant **de trouver le chemin le plus court.**
- Si l'évaluation renvoie simplement toujours **zéro**, qui n'est jamais une surestimation, alors, A* exécutera une implémentation possible de l'algorithme de Dijkstra et trouvera toujours la solution optimale.

HEURISTIQUE, ALGORITHME A*

- **Variables importantes : open et closed [38]**

• **open** :

– contient les nœuds à traiter;

– c'est à dire à la **frontière de la partie explorée (successeur)** jusqu'à présent dans le graphe.

• **closed** :

– contient les nœuds déjà **traités (visité**);

– c'est à dire à l'intérieur de la frontière délimitée par open.

HEURISTIQUE, ALGORITHME A* [38]

- **Insertion des nœuds dans open**

• Les nœuds dans **open** sont ordonnés selon une estimation **f(n)** de la qualité d'une solution passant par ce nœud.

• Une fonction **f(n)** donne ou estime la qualité de la meilleure solution passant par le nœud **n**.

• Pour chaque nœud **n**, **f(n)** est un nombre réel positif ou nul, estimant le coût pour un chemin partant de l'état initial **n0** , passant par **n**, et arrivant dans un état **n'** satisfaisant le but **g**.

HEURISTIQUE, ALGORITHME A* [38]

- **Définition de la fonction f(n)**

- **La fonction f désigne la distance entre le nœud initial et le but.**
- En pratique on ne connaît pas cette distance : c'est ce qu'on cherche !
- Par contre on connaît la distance optimale dans la partie explorée entre le nœud initial n0 et un nœud déjà exploré.
- Ainsi, on peut séparer **f(n)** en deux parties: **f(n) = g(n) + h(n)**
  - **g(n)** : coût réel du chemin optimal partant du nœud initial **n0** à **n** dans la partie déjà explorée.
  - **h(n)** : coût estimé du reste du chemin partant de **n** jusqu'à un état satisfaisant le **but**.
  - **h(n)** est une fonction heuristique.

HEURISTIQUE, ALGORITHME A* [38]

Algorithme générique de recherche dans un graphe

Algorithme rechercheDansGraphe(n0)

1. **open** ← Créer un ensemble ordonnées par f(n) // vide au départ
2. **closed** ← Créer un ensemble // vide au départ
3. insérer **n0** dans **open**
4. **while** (true) // la condition de sortie (exit) est déterminée dans la boucle
   5. **Si** open est vide **alors** sortir de la boucle avec échec (aucune solution n'existe)
   6. n1 = noeud au début de open avec le plus petit f(n)
   7. enlever n1 de open et l'ajouter dans closed
   8. **Si** n1 est le but **alors** sortir de la boucle avec succès en retournant le chemin;
   9. **Pour** chaque noeud successeur n2 de n1
      10. Initialiser la valeur g(n2) = g(n1) + coût de la transition (n1,n2)
      11. mettre parent(n2) = n1
      12. Si open ou closed* contient un nœud n3 équivalent à n2 (même état) avec **f(n2)< f(n3)** enlever n3 de open ou closed* et insérer n2 dansopen.
      13. sinon (c-à-d., n2 n'est est ni dans open ni dans closed)
          14. insèrer n2 dans open en triant les nœuds en ordre croissant selon f(n)

HEURISTIQUE, ALGORITHME HILL CLIMBING

- La **méthode *hill-climbing*** est une méthode d'optimisation permettant de trouver un optimum local parmi un ensemble de configurations.

- Le *hill-climbing* est une méthode générale qui prend en entrée trois objets :
  - Une configuration,
  - Une fonction qui pour chaque configuration donne un ensemble de configurations voisines, et
  - Une fonction-objectif qui permet d'évaluer chaque configuration.

- La méthode consiste simplement à partir de la configuration initiale, à évaluer les solutions voisines, et à choisir la meilleure de celles-ci, et à recommencer l'opération jusqu'à arriver à une position meilleure que les positions voisines (un optimum local) [36].

HEURISTIQUE, ALGORITHME HILL CLIMBING

- Le *hill-climbing* est une **méthode,**
  - **Itérative**
  - **Locale, et**
  - **Gloutonne.**

- Elle est similaire à la méthode de la **descente de gradient** mais est plutôt utilisée dans un contexte discret où le gradient n'est pas défini. Certains auteurs donnent une définition plus restreinte du *hill-climbing*, où chaque configuration correspond à un vecteur, et chaque voisinage considéré ne fait varier qu'une seule coordonnée.

Méta-heuristiques

- Algorithmes d'optimisation généralement de type stochastique combinant plusieurs approches heuristiques.

- L'intérêt et domaines d'applications des méthodes heuristiques
  - Est dans les problèmes d'optimisation de la forme :

$$\min_{x \in X} \mathbf{f(x)}$$
$$\begin{cases} \text{sous des contraintes} \\ \mathbf{g(x)} \leq \mathbf{b} \end{cases}$$

Figure. La fonction d'optimisation

- La fonction f peut être optimisation multi-objectifs et la fonction objectif f et les contraintes g sont non linéaires.

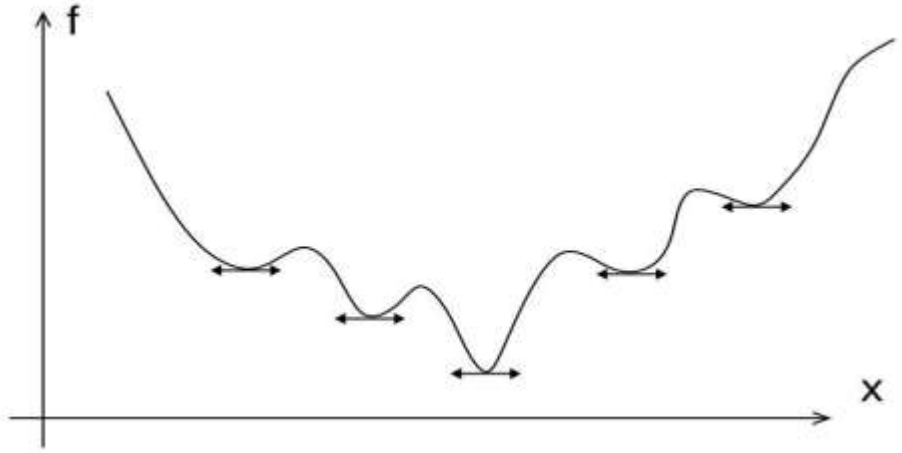

Figure. Exemple de fonction d'optimisation

❖ **Problème** : Plusieurs minima locaux possibles

⇒ Les méthodes classiques d'optimisation couteuses et incapables de capturer la solution globale.

❖ **Solution** : Méthodes méta-heuristiques, capacité à s'extraire d'un minimum local pour aller vers un minimum global.

❑ Algorithme génétique

Les AGs forment une famille très intéressante d'algorithmes d'optimisation. Ils ont été développés en première par John Holland à l'université de Michigan ("Adaptation in Natural and Artificial Systems", 1975).

- **Individu** : une solution potentielle du problème (un élément de l'espace de recherche).
- **Génotype ou chromosome** : c'est une autre façon de dire "individu".
- **Gène** : un chromosome est composé de gènes. Dans le codage binaire, un gène vaut soit 0 soit 1.
- **Phénotype** : chaque génotype représente une solution potentielle à un problème d'optimisation. La valeur de cette solution potentielle est appelée le phénotype.

- Le principe des **AGs** est de coder chaque solution potentielle d'un problème par un "***chromosome***".

- L'ensemble des chromosomes ou "**individus**" forme alors la "**population**" qui va être amenée à évoluer au cours du temps.

- Une "**génération**" est l'état de la population à un instant t.

- La **population** évolue au cours des générations en suivant des lois de **sélection**, de **croisement** et de **mutation**.

❑ Les opérateurs génétiques

❖ **La sélection :** l'opérateur de sélection donne plus de chance aux "**bons**" individus de participer à la génération suivante en fonction d'un certain critère, appelé **la fitness.**

❖**La mutation** : elle effectue une modification des gènes des chromosomes des enfants.

En fonction du problème à résoudre, à chaque individu est associé un 'degré d'adaptation à son environnement', la fitness.

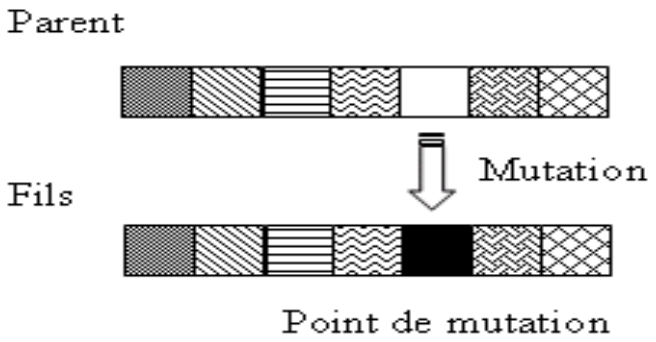

Figure. L'opérateur de mutation.

❑ Les opérateurs génétiques

❖ **Le croisement** : l'objectif de croisement est de combiner des chromosomes à partir de parents présentant des qualités pour obtenir de meilleurs individus.

Le croisement consiste à mélanger, au niveau du génome, des gènes de deux individus parents pour générer deux enfants. La taille de la population reste constante.

➢ **Le croisement par coupure :** le génome des deux parents est sectionné en un même endroit, choisi le plus souvent aléatoirement. Le nombre de points de coupure est un paramètre de l'opérateur. Les fragments sont recombinés pour constituer les enfants.

➢ **Le croisement uniforme :** dans le cas du croisement uniforme, un masque binaire aléatoire de même longueur que le génome d'un chromosome utilisé. Si le bit i du masque est à 1, alors l'enfant 1 prend le bit i du parent 1, et l'enfant 2 prend le bit i du parent 2. Si par contre c'est un 0, c'est l'inverse qui est effectué.

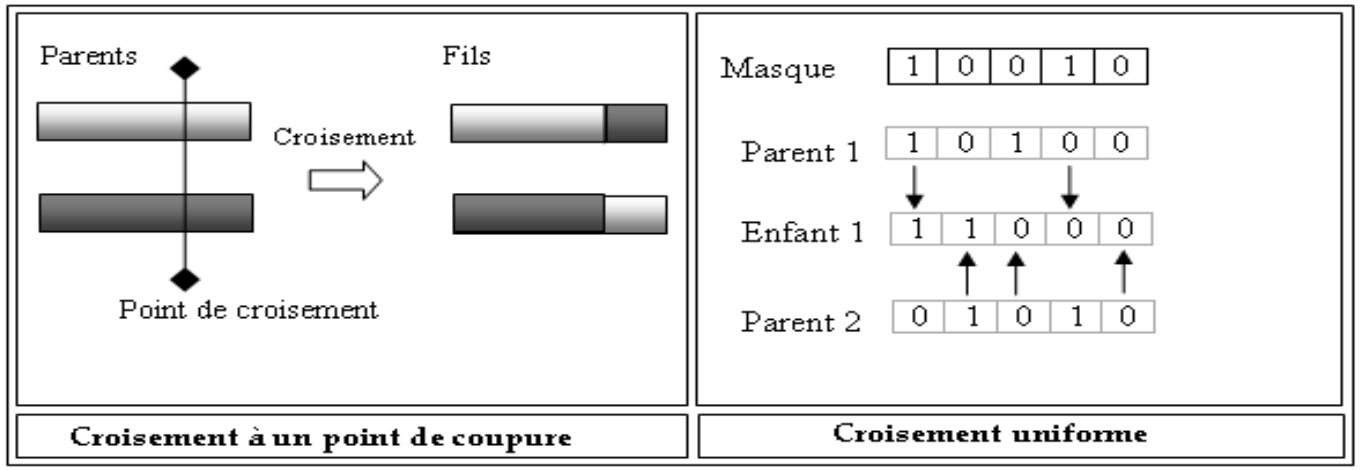

Figure. L'opérateur de croisement.

❖ Après croisement et mutation, une nouvelle génération est construite en conservant les individus de la population précédente ayant une propriété de fitness particulière, jusqu'à. la convergence vers une solution optimale.

Le processus d'optimisation

1. Initialisation : générer un ensemble de solutions initiales.
2. Phase de mise à jour.
   A. Evolution :
   - **Sélection** : choisir avec une probabilité proportionnelle à leur qualité une liste d'individus.
   - **Reproduction** : générer à partir de cette liste de nouveaux individus à l'aide des opérateurs génétiques.
   - **Remplacement** : éliminer avec une probabilité inversement proportionnelle à leur qualité certains individus.

   B. réactualisation de la meilleure solution.

   C. allez en A tant que Nombre de Générations ≤ Valeur pré-déterminée.

Le processus d'optimisation

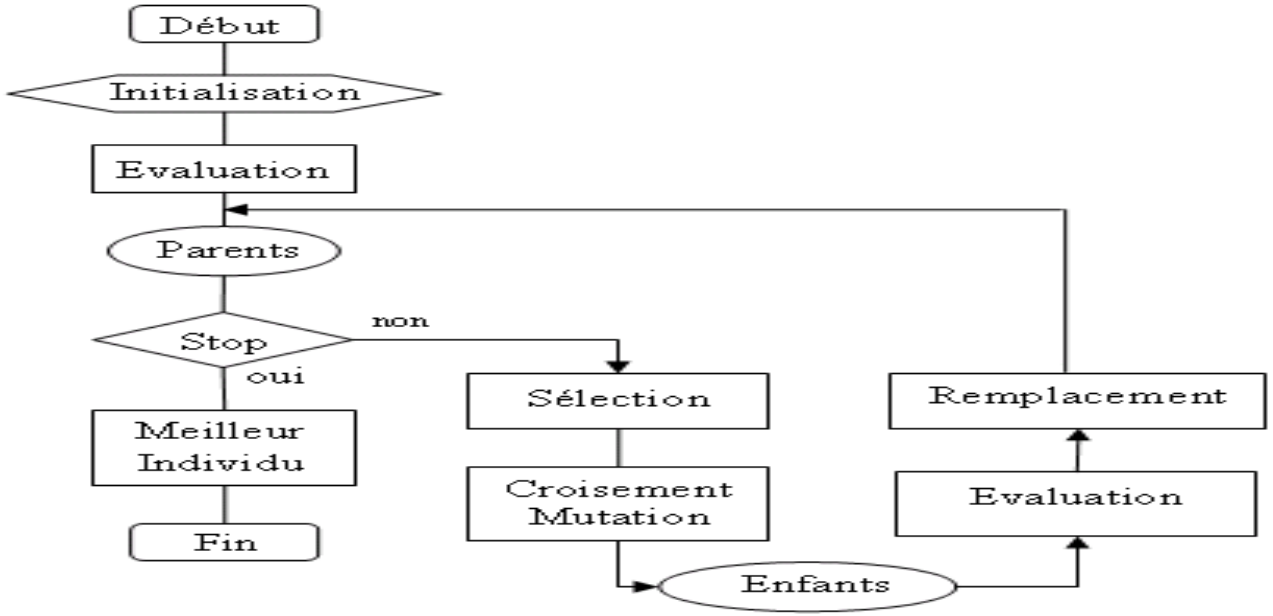

Figure. Principe de base d'un AG.

- La fonction objectif f peut être non-linéaire.
- Temps de calculs souvent importants.
- Difficultés de mise en oeuvre.
- Fournit généralement une solution approchée.
- Problèmes d'extréma locaux.

❑ Exercice 1 - les AG pour les n reines

- On a n reines qu'il faut placer dans un échiquier (matrice n x n) sans qu'aucune d'entres elles ne soit en prise par une autre.
- Donc une reine placée dans la case d'indice (i, j) condamnera la ligne i, la colonne j et les deux diagonales passant par la case (i, j). Les états solutions sont ceux représentant des échiquiers avec n reines déjà placées sans qu'aucune ne soit en prise.

➢ Montrer comment peut-on appliquer les algorithmes génétiques pour la résolution de ce problème.

NB. Deux reines sont en prise si elles se trouvent sur une même ligne, une même colonne ou une même diagonale.

❑ Solution

❖ **Population** :

De taille *n* individus

❖ **Initialisation de l'algorithme génétique** :

Initialement, les *n* individus (chromosomes) sont générés de manière aléatoire

❖ **Chromosome** (individu) :

- Codage binaire (que des 0 et des 1)
- Le chromosome est une matrice de dimension $n \times n$ ou un vecteur de longueur $n \times n$
- Chaque gène indique la présence ou l'absence d'une reine dans la case (i, j)

❖ **Fonction objectif à maximiser (Fitness function)** :

Il s'agit de compter le nombre de reines qui ne sont pas en prise

❑ Solution

❖ **Mutation :**

Choisir avec une probabilité p_m une case de manière aléatoire et inverser sa valeur

❖ **Croisement :**

- Croisement à 1 point avec une probabilité p_c
- Choisir un point de croisement et échanger les parties des deux individus

❖ **Sélection :**

Elitiste ou la roue de la fortune.

❖ **Critère d'arrêt :**

S'arrêter lorsque :

- Les *n* reines sont bien placées (fitness = *n*)
- On atteint un nombre maximum MaxGen d'itérations

❑ Exercice 2 - Algorithme glouton pour les pièces de monnaie

On dispose de pièces en nombre illimité de valeurs.
Pièces={100DA, 50DA, 20DA, 10DA, 5DA, 2DA, 1DA}

- Montrer comment peut-on appliquer l'algorithme glouton pour rendre une somme **S=257 DA** avec un nombre minimum de pièces.

❑ Solution - Algorithme glouton pour les pièces de monnaie

- On veut totaliser une somme d'argent S en utilisant un nombre minimal de pièces appartenant à un ensemble donné.
- A chaque étape on choisit la plus grande pièce <= S tout en mettant à jour la nouvelle somme (S – la pièce choisie):

Pour **S=257 DA** et pièces={**100DA, 50DA, 20DA, 10DA, 5DA, 2DA, 1DA}**

- à l'étape 1 : S1 = 257 , on choisit 1 pièce de 100DA
- à l'étape 2 : S2 = 157 , on choisit 1 pièce de 100DA
- à l'étape 3 : S3 = 57 , on choisit 1 pièce de 50DA
- à l'étape 4 : S4 = 7 , on choisit 1 pièce de 5DA
- à l'étape 5 : S5 = 2 , on choisit 1 pièce de 2DA

➢ La solution trouvée est donc **2x100 DA , 1x50 DA , 1x5 DA et 1x2 DA**