Chapitre 8 : Les Heuristiques : Méthodes de trajectoire

8.1 Les heuristiques

Une heuristique est une méthode de calcul basé sur un raisonnement intuitif utilisée pour se déplacer intelligemment dans l'espace des solutions, afin d'obtenir une solution pas nécessairement optimale, dans un délai de temps raisonnable. Les heuristiques sont favorisées dans le cas où la résolution exacte est couteuse voir impraticable.

Deux types d'heuristiques sont principalement utilisées : les heuristiques de construction (ou méthodes gloutonnes), qui construisent progressivement une solution, et les heuristiques de descente ou de trajectoire, qui à partir d'une solution donnée cherchent un optimum local.

Il faut noter que les heuristiques sont spécifiques au problème à résoudre, principalement dans le choix du voisinage ou la sélection de l'étape suivante.

Dans ce chapitre, on s'intéresse beaucoup plus aux heuristiques de trajectoire. Ces méthodes sont divisées en deux catégories : celles qui permettent de déterminer un minimum local, et celles qui s'efforcent de déterminer un optimum global.

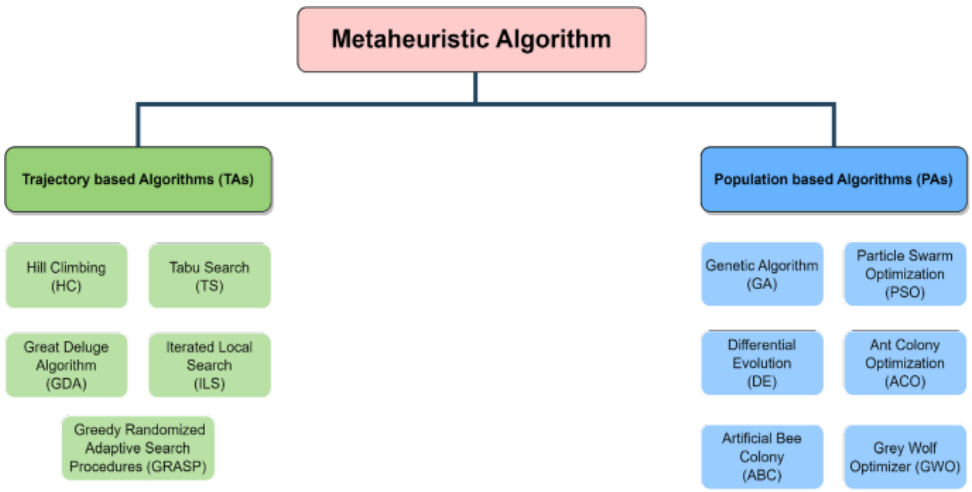

Figure 8.1 – Classification des heuristiques.

8.4 Méthodes de recherche locale

En informatique, la recherche locale est une méthode d'optimisation utilisée pour résoudre des problèmes dans plusieurs domaines comme l'intelligence artificielle, mathématiques, la recherche opérationnelle, l'ingénierie et la bioinformatique. Elle peut être utilisée sur des problèmes de recherche d'une solution maximisant (minimisant) un critère parmi un ensemble de solutions candidates. Ceci est dû à sa rapidité et à son principe relativement simple. Le principe de la recherche locale est le suivant : l'algorithme débute avec une solution initiale réalisable. Sur cette solution initiale, on applique une série de modifications locales (définissant un voisinage de la solution courante), tant que celles-ci améliorent la qualité de la fonction objectif. La figure suivante illustre bien le fonctionnement de l'algorithme général des méthodes de recherche locale.

Le passage d'une solution vers une autre se fait grâce à la définition de structure de voisinage qui est un élément très important dans la définition de ce type de méthode. Le voisinage d'une solution est défini en fonction du problème à résoudre. On définit l'espace de recherche comme l'espace dans lequel la recherche locale s'effectue. Cet espace peut correspondre à l'espace des solutions possibles du problème étudié. Habituellement, chaque solution candidate a plus d'une solution voisine, le choix de celle vers laquelle se déplacer est est basé sur un certain critère généralement la valeur de la fonction objectif.

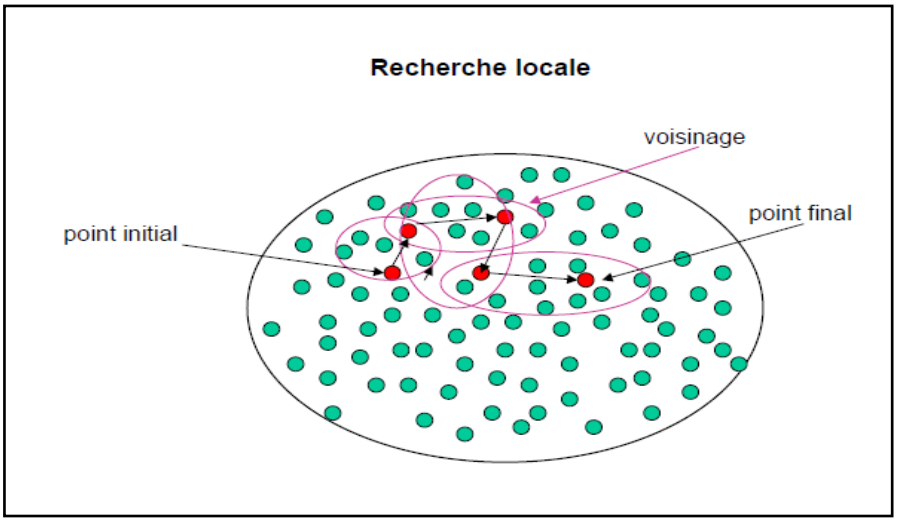

Figure 8.2 – Représentation de la procédure générale de la recherche locale.

D'une manière abstraite, une recherche locale peut se résumer de la façon suivante :

(i) Démarrer avec une solution.

(ii) Améliorer la solution.

(iii) Répéter le processus jusqu'à la satisfaction des critères d'arrêt.

Les méthodes de recherche locales utilisent principalement trois fonctions:

- Une fonction d'évaluation
- Une fonction recherche de voisinage et,
- Une fonction d'amélioration basée sur la fonction d'évaluation.

D'un autre côté, le critère d'arrêt de la recherche locale peut être le nombre d'itérations maximal, une limite en durée, une autre possibilité est de s'arrêter quand la meilleure solution trouvée par l'algorithme n'a pas été améliorée depuis un nombre donné d'itérations.

Par ailleurs, les algorithmes de recherche locale sont des algorithmes sous-optimaux puisque la recherche peut s'arrêter alors que la meilleure solution trouvée par l'algorithme n'est pas la meilleure de l'espace de recherche. Cette situation peut se produire même si l'arrêt est provoqué par l'impossibilité d'améliorer la solution courante car la solution optimale peut se trouver loin du voisinage des solutions parcourues par l'algorithme.

Un autre problème rencontré lors de l'utilisation des méthodes de recherche locale est la présence des minimas locaux. En effet, les méthodes de recherche locale sont souvent piégées dans l'un des minimas et ainsi elles donnent des solutions sous optimales.

On trouve dans la littérature un grand nombre de méthodes de recherche locale, nous évoquerons dans la suite trois méthodes générales largement utilisées, qui sont : méthode de la descente, le recuit simulé et la recherche tabou. La procédure générale de la recherche locale est décrite par l'algorithme ci-dessous. Les bases de la recherche locale peuvent être définies ainsi. Soient :

– f () la fonction qu'on l'on souhaite maximiser,
– S : la solution courante,
– S* la meilleure solution connue,
– N(S) le voisinage de S

Algorithme 8.1 : Algorithme de recherche locale simple

Étape 1 (initialisation)

a) choisir une solution initial s ∈S.

b) s* ←s (i.e. s* mémorise la meilleure solution trouvée)

Étape 2 *(choix et terminaison)*

a) *choisir* s'∈ N(s) s* ←s si f(s) < f(s*)

b) s ← s'(i.e. remplacer s par s')

c) fin si la condition d'arrêt est vérifiée sinon aller à l'Étape 2.a

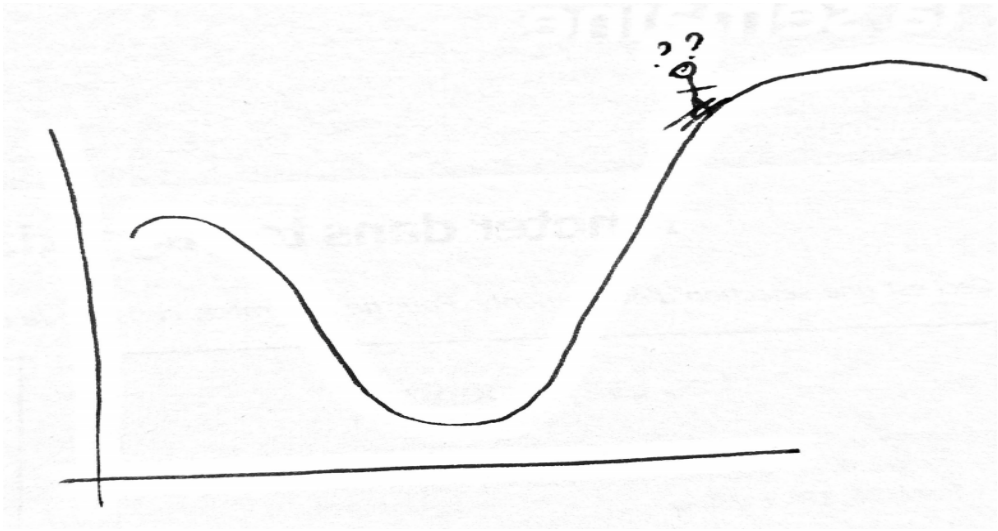

Figure 8.3– La Méthode de la Descente

8.4.1. La Méthode de la Descente (Hill-Climbing)

Cette méthode appelée aussi Hill-Climbing est assez ancienne, mais d'une grande simplicité. Elle consiste à se déplacer dans l'espace de recherche en choisissant toujours la meilleure solution parmi le voisinage de la solution courante. On part d'une solution si possible bonne et on balaie l'ensemble des voisins de cette solution. S'il n'existe pas de voisin meilleur que notre solution, on a trouvé un optimum local et on s'arrête; sinon, on choisit le meilleur des voisins et on recommence (L'algorithme 8.2). La convergence

vers un optimum local pouvant être très lente, on peut éventuellement fixer le nombre d'itérations maximum, si on veut limiter le temps d'exécution. Cette méthode a l'inconvénient de rester bloquée dans un optimum local. En effet, une fois un optimum local trouvé, on s'arrête, même si ce n'est pas l'optimum global. Selon le paysage des solutions, l'optimum local peut être très bon ou très mauvais par rapport à l'optimum global. Si la solution de départ est donnée par une heuristique déterministe, l'algorithme sera déterministe. Si elle est tirée au hasard, on a un algorithme non déterministe et donc plusieurs exécutions différentes sur la même instance pourront donner des solutions différentes et de qualités différentes.

Il est tout à fait clair que dans ce type de méthodes, la notion de voisinage est primordiale. Si les voisins sont très nombreux, on a de fortes chances de trouver l'optimum global. Malgré cela, visiter un voisinage peut être très long par ce qu'on visitera une grande partie de l'espace des solutions. D'un autre côté, si le voisinage est très restreint, on risque fort de rester bloqué dans un optimum local de mauvaise qualité. Par conséquent, le choix de la notion de voisinage est un compromis entre efficacité et qualité.

Algorithme 8.2 : Schéma général du Hill-Climbing

```
Début
Générer et évaluer une solution initiale s
 Tant que La condition d'arrêt n'est pas vérifiée faire
  Modifier s pour obtenir s' et évaluer s'
  si (s' est meilleure que s) alors
   Remplacer s par s'
  finsi
 Fin tant que
 retourner s
Fin
```

On distingue différents types de descente en fonction de la stratégie de génération de la solution de départ et du parcours du voisinage : la descente déterministe, la descente stochastique et la descente vers le premier meilleur. On peut par exemple remarquer que dans la formulation précédente, il n'est pas mentionné que l'aléatoire est mis en jeu pour la génération de la solution initiale ou sa modification. L'algorithme 8.3 décrit la version stochastique *Random Hill-Climbing* (RHC). Il faut alors définir un pas de recherche qui détermine la taille du voisinage pouvant être exploré.

Algorithme 8.3 : Schéma général du Random Hill-Climbing

```
Début
  Générer aléatoirement une solution initiale s et l'évaluer
  tant que La condition d'arrêt n'est pas vérifiée faire
    Modifier aléatoirement s pour obtenir s' et évaluer s'
    si (s' est meilleure que s) alors
      Remplacer s par s'
    fin si
  fin tantque
  retourner s
Fin
```

La méthode hill-climbing possède plusieurs avantages. C'est une méthode intuitive, simple à implémenter par rapport à d'autres méthodes de recherche locale. Typiquement, Il n'existe pas de notion d'arbre de recherche ni de retour en arrière. La méthode hill-climbing donne souvent de bonnes solutions. Cependant, cette méthode présente quelques limitations comme le problème du minimum local et la lenteur de la méthode.

8.4.2 Le recuit simulé

Le recuit simulé (Simulated annealing SA) est un algorithme de recherche locale présenté par Metropolis et al. En 1953 et popularisé par les trois chercheurs Kirkpatrick, Gelatt et Vecchi en 1983. Il a été appliqué avec succès à une grande variété de problèmes d'optimisation combinatoire très compliqués ainsi qu'à divers problèmes du monde réel comme le Problème de voyageur de commerce (TSP), le problème de satisfiabilité (SAT), le problème de sac à dos …etc.

Le concept de la méthode est adopté à partir du processus du « recuit» utilisé dans l'industrie métallurgique. Le recuit est le processus par lequel un refroidissement lent est appliqué aux métaux pour produire une cristallisation mieux alignée et à faible énergie. En fait, les thermodynamiciens ont constaté qu'un changement brusque de la température conduit à une structure amorphe. Alors qu'une baisse progressive de la température permet d'aboutir une structure solide. Cela montre que la structure du matériau obtenu varie en fonction de la vitesse de refroidissement.

Idem, l'algorithme de recuit simulé commence par une haute température T et une solution initiale s. A chaque itération, une nouvelle solution s' est générée du voisinage de la solution actuelle. La valeur de la fonction objective de cette nouvelle solution est

ensuite comparée à celle de la meilleure solution actuelle afin de déterminer si une amélioration a été réalisée. Si la valeur de la fonction objectif de la nouvelle solution est meilleure, c'est-à-dire plus petite dans le cas de la minimisation la nouvelle solution devient la solution actuelle à partir de laquelle la recherche continue en procédant à une nouvelle itération.

En d'autre terme l'algorithme calcule Δ qui représente la différence entre la qualité de la solution *s* et *s'* tel que Δ= *f*(s') - *f*(s). Pour les problèmes de minimisation, la solution *s'* remplace *s* à chaque fois où Δ>0. La nouveauté dans le recuit simulé est qu'une mauvaise solution avec une valeur de fonction objectif dégradée peut également être acceptée comme nouvelle solution courante mais avec une probabilité déterminée $p = e^{(-\Delta)/T}$. Cette probabilité d'acceptation est comparée à un nombre $r \in [0,1]$ généré aléatoirement, la solution *s* est remplacée par *s'* si et seulement si $p > r$.

Au début de la recherche, la température élevée donne plus de probabilité d'acceptation des solutions de mauvaise qualité. Au fur et à mesure que la température décroît, l'acceptation d'une mauvaise solution sera quasi impossible. La propriété d'acceptation des solutions de mauvaises qualités permet au RS d'explorer bien l'espace de recherche et d'échapper au piège de l'optimum local. Cela peut mener la recherche vers la meilleure solution (l'optimum global) car cette dernière peut faire partie du voisinage d'une mauvaise solution. Pour ne pas perdre la meilleure solution rencontrée au cours de l'exécution, l'algorithme utilise une variable de mémorisation qui consiste à garder la meilleure solution trouvée dans toutes les itérations.

Dans l'algorithme de recuit simulé, plusieurs critères d'arrêts peuvent être établis comme le nombre maximal d'itérations, la température est arrivée à la valeur finale prédéfinie, et si on ne trouve pas une amélioration de la solution après un certain nombre d'itération. Le pseudo code de l'algorithme du recuit simulé est comme suit :

Algorithme 8.4 Algorithme de recuit simulé

Début
Générer une solution initial s ;
Initialiser la valeur de la température T ;
$s_{best}=s$
Répéter
Générer aléatoirement une solution s' voisine de s ;
$\Delta f = f(s') - f(s)$;
Si $\Delta f \leq 0$ **alors** // cas de minimisation
$s_{best} = s'$;
$s=s'$
Sinon Si $(r < p)$ **Alors**
$s=s'$
Fin Si
Décrémenter la température T ;
Jusqu'à la condition d'arrêt est satisfaite ;
Retourner la solution s_{best}.
Fin

Le recuit simulé est un algorithme simple et facile à implémenter. Il offre la souplesse d'emploi en comparant avec les méthodes de la recherche locale classique. En outre, il facilite l'intégration des nouvelles contraintes liées au problème traité. Cependant, l'un des inconvénients du recuit simulé est la convergence lente dans le cas des problèmes de grande taille.

Néanmoins, Une difficulté spécifique à l'algorithme du recuit simulé est l'ajustement du paramètre de la température T. En effet, Un refroidissement trop rapide mènerait vers un optimum local pouvant être de très mauvaise qualité. Un refroidissement trop lent serait très coûteux en temps de calcul.

8.4.3 La recherche Tabou (tabu search)

La recherche tabou est une méthode de recherche locale avancée qui fait appel à un ensemble de règles et de mécanismes généraux pour guider la recherche de manière intelligente. Le principe de la recherche tabou est d'explorer le voisinage et choisir la position dans ce voisinage qui minimise la fonction objectif à partir d'une position donnée. Lorsque tous les points du voisinage ont une valeur plus élevée, cette opération peut conduire à augmenter la valeur de la fonction. Dans cette méthode le mécanismes d'échappement aux minimax locaux consiste à stocker en mémoire les dernières positions explorées d'où le nom tabou. Les positions sont enregistrées dans une pile FIFO de taille paramétrable. Toutefois, la mémorisation des solutions complètes rend la gestion de la liste Tabou trop coûteuse en temps de calcul et en place mémoire; particulièrement si le nombre de variables est très élevé. On peut réussir à pallier ce problème en ne stockant que les mouvements et la valeur de la fonction à minimiser.

La recherche tabou possède plusieurs avantages tels que la rapidité, facile à implémenter, donne souvent de bonnes solutions, et possède un fonctionnement intuitif. Cependant, la recherche tabou possède certaines limitations comme le manque de modélisation mathématiques. En effet, chaque cas est différent et il faut donc adapter la recherche à chaque problème particulier. D'un autre le grand nombre de paramètre à régler rend l'utilisation de la méthode tabou difficile. Finalement, la recherche tabou a été appliquée avec succès pour résoudre de nombreux problèmes difficiles d'optimisation combinatoire : problèmes de routage de véhicules, problèmes d'affectation quadratique, problèmes d'ordonnancement, problèmes de coloration de graphes, etc. L'algorithme général de la recherche tabou est le suivant :

Algorithme 8.5: Schéma général de la recherche tabou

Début
Générer une solution initiale s ;
Calculer f(s) ;
Initialiser la liste tabou T= {} ;
$s_{best} = s$;
Répéter
Trouver la meilleure solution s' voisine de s qui ne soit pas tabou ou qui vérifie le critère d'aspiration ;
Calculer $f(s')$;
Si $f(s') \geq f(s_{best})$ **alors**
$s_{best} = s'$;
Fin Si
Mettre à jour T ;
$s = s'$;
Jusqu'à le critère d'arrêt est vérifié
Retourner s_{best} ;
Fin

8.5 Discussion des méthodes de recherche locale

Bien que les méthodes de recherche locale aient démontré leurs efficacités dans plusieurs problèmes d'optimisation, la difficulté essentielle liée à l'utilisation de ces méthodes consiste à réussir une exploration de l'espace de recherche sans stagner dans des minimas locaux. La recherche se trouve dans un minimum local lorsque plus aucun des voisins de l'affectation courante ne l'améliore, par exemple, dans la figure suivante, on observe plusieurs minimas locaux. Pour sortir de ces minima, les algorithmes doivent avoir la capacité de détériorer l'affectation courante afin de l'améliorer ultérieurement en atteignant l'optimum global. Cette technique est connue sous le nom du *chemin aléatoire* (Random Walk). Elle consiste à effectuer, de temps en temps, un

mouvement aléatoire pour diversifier la recherche et ainsi espérer se rapprocher de la bonne solution. Le fait d'accepter la détérioration de l'affectation courante permet de sortir de certains minima locaux mais peut induire un allongement du temps d'exécution. On trouve une technique similaire à celle-ci dans l'algorithme de recuit simulé. Une autre technique couramment utilisée est celle de la *relance* (Multiple Start). Elle consiste à répéter une nouvelle recherche à partir d'une autre solution initiale appartenant à une autre région de l'espace de recherche. Cette technique peut être réitérée plusieurs fois dans l'espérance d'atteindre différentes zones prometteuses de l'espace de recherche.

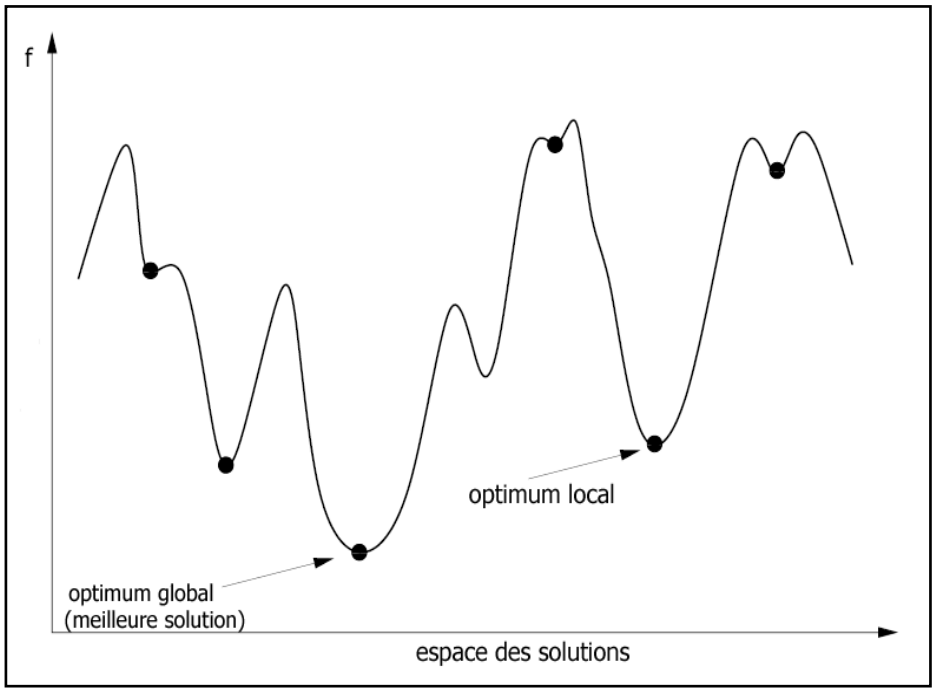

Figure 8.4– Le problème de minimum local dans la recherche locale.

8.6. Exemple d'applications de méthodes de recherche locale

La résolution du problème du sac à dos par le recuit simulé

Nous allons maintenant résoudre le fameux problème di sac à dos par un algorithme du recuit simulé. Tout d'abord, il faut déterminer les paramètres du recuit tels que :

Température = T0

Taux de refroidissement : $0<\alpha<1$

1. on détermine la représentation de la solution utilisée. Pour le problème du sac à dos, la solution est représenté par un vecteur binaire « sol » de taille n où n est le nombre d'objet.

Sol=binaire[n]

2. On détermine la fonction objectif, c'est le profit total des objets sélectionnés :

$$profit = \sum_{i=1}^{n} p(i) * sol(i)$$

3. L'opération de la génération de nouvelles solutions
   l'opérateur de voisinage joue un rôle important dans la convergence de tout algorithme de recherche locale. Dans notre algorithme simple, on utilise un opérateur de flip qui inverse la valeur d'un bit de la solution en cours d'une manière aléatoire. L'exemple suivant montre l'opération de flip :

   Solution en cours

1	0	1	1	0

   Solution après modification

1	1	1	0	1

4. Traitement des solutions infaisables : l'opérateur de voisinage va générer certainement des solutions dont le poids total des objets sélectionnés dépasse la capacité du sac à dos. Pour remédier ce problème, une procédure de réparation doit être utilisée. Trois stratégies de traitement des solutions infaisables peuvent être utilisées :

   - Transformer la solution infaisable en solution valide
   - Rejeter la solution infaisable
   - Introduire une pénalité dans la fonction objectif.

8.7 Les méthodes de recherche globales

Les métaheuristiques de recherche globale, par opposition aux métaheuristiques de recherche locale, sont des techniques d'optimisation qui se concentrent sur l'exploration de l'ensemble de l'espace de recherche pour trouver la meilleure solution possible à un problème donné. Contrairement aux métaheuristiques de recherche locale, qui se concentrent sur l'amélioration progressive d'une solution initiale, les métaheuristiques globales visent à rechercher des solutions dans tout l'espace de recherche, y compris les régions éloignées et non explorées. Voici un aperçu des métaheuristiques de recherche globale et de leurs caractéristiques principales :