META-HEURISTIQUE

JEU DE N REINES

IMANE ELMALI | MANAL KASSAOUI | RIM JAMAA

Table des matières

Jeu de n reines 1
Chapitre 1: Introduction & Fondements Théoriques 4
Chapitre2 : METHODOLOGIE 9
1.Introduction au Problème des N-Reines 9
2.Méthodologie Gloutonne 11
3.Recherche Locale - Recuit Simulé(Simulated Annealing) : 14
CHAPITRE 3 : **Résultats et interprétations** 19
1. ..... Mise en Œuvre : 19
2. ..... Analyse des Résultats : 22
3. ..... Discussion et Conclusion : 22
2-Stratégies de Voisinage dans le Recuit Simulé : 23
3-Critères d'Acceptation dans le Recuit Simulé : 25
4-Techniques Avancées dans le Recuit Simulé : 26
5-Visualisation : 27
6-Parallélisme dans le Recuit Simulé : 29
7-Évaluation Dynamique des Paramètres dans le Recuit Simulé : 30
En résume : 31
Conclusion générale : 32
REFERENCES 34
Livres et articles académiques 34
Articles de revue et conférences 34
Ressources en ligne 35
Outils et bibliothèques 35

CHAPITRE 1: INTRODUCTION & FONDEMENTS THEORIQUES

▪ *Introduction :*

- **Présentation du problème des 8 reines :**

Le problème des 8 reines est un problème classique d'échecs qui consiste à placer huit reines sur un échiquier standard de 8x8 de manière à ce qu'aucune reine ne puisse en attaquer une autre. Cela signifie que deux reines ne doivent pas se trouver sur la même ligne, colonne ou diagonale. Ce problème a été posé pour la première fois en 1848 par Max Bezzel, et depuis lors, il est devenu un exemple emblématique en algorithmique, en théorie des graphes et en intelligence artificielle.

- **Importance et applications du problème :**

Le problème des 8 reines est important non seulement en raison de sa simplicité apparente mais aussi en raison de sa complexité croissante lorsqu'il est généralisé à n reines sur un échiquier de n x n. Les solutions à ce problème peuvent être appliquées à divers domaines, y compris l'optimisation, la recherche opérationnelle et la théorie des jeux. En tant qu'exemple de problème NP-complet, il est utile pour tester et comparer les performances des algorithmes heuristiques et exacts.

- **Objectif du rapport :**

Ce rapport vise à explorer l'application de l'algorithme de recuit simulé pour résoudre le problème des 8 reines. L'objectif est de démontrer l'efficacité de cette méta-heuristique en termes de rapidité de convergence et de qualité des solutions trouvées. Le rapport couvrira les bases théoriques, l'implémentation pratique et l'analyse des résultats obtenus.

1.2 Fondements Théoriques :

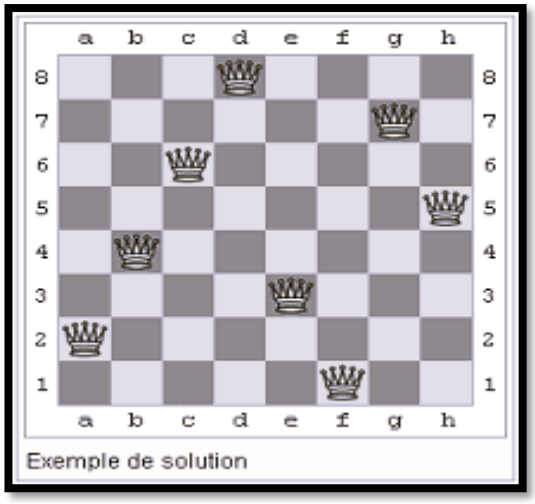
- **Description du problème des 8 reines :**

Le problème des 8 reines demande de placer huit reines sur un échiquier de 8x8 de manière à ce qu'aucune reine ne puisse capturer une autre selon les règles du jeu d'échecs. En termes simples, une reine peut attaquer une autre si elle se trouve sur la même ligne, colonne ou diagonale. Le défi consiste donc à positionner les reines de telle sorte que ces conditions ne soient pas remplies.

- **Historique et versions du problème :**

Le problème des 8 reines a été posé pour la première fois en 1848 par Max Bezzel. Depuis lors, de nombreuses variantes et généralisations ont été étudiées, telles que le problème des n reines, où n représente le nombre de reines et la taille de l'échiquier. Diverses méthodes de résolution ont été développées, allant des approches naïves et des algorithmes de backtracking aux méthodes plus sophistiquées comme les algorithmes génétiques et les algorithmes de recuit simulé.

- **Méthodes traditionnelles de résolution :**



Exemple de solution

Les méthodes traditionnelles de résolution du problème des 8 reines incluent :

- **Backtracking** : Cette approche explore toutes les configurations possibles de l'échiquier en plaçant les reines une par une et en revenant en arrière chaque fois qu'une position mène à un conflit.
- **Algorithmes de recherche locale** : Ces méthodes, telles que la descente de gradient, commencent avec une solution initiale et font des modifications locales pour améliorer progressivement la solution.
- **Algorithmes exacts** : Des méthodes comme la programmation linéaire et la programmation par contraintes peuvent également être utilisées pour trouver toutes les solutions possibles.

1.3 Algorithmes Méta-heuristiques :

- **Introduction aux méta-heuristiques :**

Les méta-heuristiques sont des stratégies d'optimisation de haut niveau conçues pour résoudre des problèmes d'optimisation complexes, souvent en explorant efficacement de grands espaces de solutions. Contrairement aux algorithmes exacts, les méta-heuristiques ne garantissent pas de trouver la solution optimale mais fournissent souvent de bonnes solutions en un temps raisonnable.

- **Comparaison avec les algorithmes exacts :**

Les algorithmes exacts, tels que ceux utilisés dans les méthodes traditionnelles, garantissent de trouver la solution optimale mais peuvent être inefficaces pour les grands problèmes en raison de leur complexité temporelle élevée. Les méta-heuristiques, en revanche, offrent un compromis entre la qualité de la solution et le temps de calcul, les rendant adaptées pour les problèmes NP-complets comme le problème des 8 reines.

- **Avantages des méta-heuristiques pour les problèmes NP-complets :**

Les méta-heuristiques sont particulièrement utiles pour les problèmes NP-complets car elles peuvent explorer de vastes espaces de solutions de manière efficace en évitant les minima locaux. Elles utilisent des techniques stochastiques et des stratégies d'exploration/exploitation pour trouver des solutions proches de l'optimum en un temps raisonnable.

1.4 Recuit Simulé (Simulated Annealing) :

- **Concepts et principes de base :**

Le recuit simulé est une méta-heuristique inspirée du processus de recuit en métallurgie, où un matériau est chauffé puis refroidi lentement pour minimiser ses défauts. En optimisation, l'algorithme de recuit simulé commence avec une solution initiale et explore les solutions voisines en acceptant parfois des solutions moins bonnes pour échapper aux minima locaux, avec une probabilité qui diminue au fil du temps.

- **Analogies physiques et théorie sous-jacente :**

L'algorithme de recuit simulé utilise une analogie avec la physique statistique, où l'énergie du système correspond à la fonction de coût à minimiser et la température contrôle la probabilité d'accepter des solutions moins optimales. Au début, lorsque la température est élevée, le système accepte plus facilement des solutions de moindre qualité, ce qui permet une exploration globale. À mesure que la température diminue, l'algorithme se concentre sur l'exploitation locale des solutions.

- **Schéma de l'algorithme de recuit simulé :**

1. **Initialisation** : Commencer avec une solution initiale et une température élevée.
2. **Génération de voisinage** : Générer une solution voisine en effectuant un petit changement.

3. **Évaluation** : Calculer la fonction de coût de la nouvelle solution.
4. **Acceptation** : Accepter la nouvelle solution si elle améliore la solution actuelle ou avec une probabilité dépendant de la température et de la différence de coût.
5. **Refroidissement** : Réduire la température selon une stratégie de refroidissement.
6. **Répétition** : Répéter les étapes précédentes jusqu'à ce que la température soit suffisamment basse ou qu'un critère d'arrêt soit atteint.

- **Paramètres clés : température, refroidissement, etc.**

- **Température initiale** : Une température élevée permet une exploration plus large au début.
- **Taux de refroidissement** : Contrôle la vitesse à laquelle la température diminue. Un refroidissement trop rapide peut conduire à des solutions sous-optimales, tandis qu'un refroidissement trop lent peut prolonger le temps de calcul.
- **Critère d'arrêt** : L'algorithme peut s'arrêter après un nombre fixe d'itérations, lorsque la température atteint un seuil minimum ou lorsque les améliorations deviennent négligeables.

Ce premier chapitre fournit une base solide en expliquant le problème des 8 reines, les fondements théoriques des algorithmes méta-heuristiques, et les principes du recuit simulé. Les sections suivantes se concentreront sur l'application pratique de ces concepts et l'analyse des résultats obtenus.

CHAPITRE2 : METHODOLOGIE

1.INTRODUCTION AU PROBLEME DES N-REINES :

- **Définition:**

Le problème des N-Reines consiste à placer N reines sur un échiquier de taille N x N de manière à ce qu'aucune reine ne puisse attaquer une autre.

- **Objectif** :

Aucune reine ne doit partager la même ligne, colonne ou diagonale, garantissant ainsi qu'aucune reine n'est en position d'attaque par rapport à une autre.

- **Représentation et Méthodologies :**

Le problème des N-Reines consiste à placer N reines sur un échiquier N x N de manière à ce qu'aucune reine ne puisse en attaquer une autre, c'est-à-dire qu'aucune reine ne partage la même ligne, colonne ou diagonale. La représentation d'une solution se fait généralement par une liste de N entiers, où chaque élément de la liste indique la colonne où la reine est placée pour une ligne donnée. Par exemple, pour N = 4, la solution [1, 3, 0, 2] signifie que la reine est placée en colonne 1 à la ligne 0, en colonne 3 à la ligne 1, en colonne 0 à la ligne 2, et en colonne 2 à la ligne 3.
Deux méthodologies principales pour trouver des solutions sont la méthode gloutonne et la recherche locale avec recuit simulé. La méthode gloutonne consiste à placer les reines une par une en choisissant à chaque étape la position avec le moins de conflits

immédiats, ce qui est simple et rapide mais ne garantit pas une solution optimale. En revanche, la recherche locale avec recuit simulé commence par une solution initiale et explore l'espace des solutions en effectuant des perturbations mineures et en acceptant ou rejetant les nouvelles configurations en fonction d'une probabilité contrôlée par une température décroissante, permettant ainsi d'échapper aux minima locaux et de trouver des solutions plus proches de l'optimal.

- **Représentation des Solutions :**

Tableau (Liste) : Une solution est représentée par une liste de N entiers. Chaque élément de la liste indique la colonne où la reine est placée pour une ligne donnée.

- **Exemple :**

Pour N = 4, la solution [1, 3, 0, 2] signifie :
Reine à la ligne 0, colonne 1
Reine à la ligne 1, colonne 3
Reine à la ligne 2, colonne 0
Reine à la ligne 3, colonne 2

- **Représentation Visuelle d'une Solution :**

Échiquier: Utilisez une grille pour illustrer la solution [1, 3, 0, 2]
Avec Q représente Reines :

	Q		
			Q
Q			
		Q	

2.METHODOLOGIE GLOUTONNE

- Initialisation: Commencer avec un échiquier vide.
- Placement Séquentiel: Placer les reines une par une en choisissant l'option avec le moins de conflits immédiats.
- Avantages: Simple et rapide.
- Inconvénients: Ne garantit pas une solution optimale.

- **Exemple Mathématique:**

Considérons le problème des N-Reines avec N=4N=4. Nous voulons placer les reines une par une en minimisant les conflits.

1)Première Reine:

Placez la première reine sur la première ligne. La meilleure option est de la mettre en (0,0)(0,0) car il n'y a pas de conflits.

❖ Échiquier:

Q			

2)Deuxième Reine:

- Pour la deuxième ligne, nous devons choisir une colonne qui minimise les conflits.
  - Colonne 0: Conflit avec la reine en (0,0)(0,0).
  - Colonne 1: Pas de conflit immédiat.
  - Colonne 2: Pas de conflit immédiat.
  - Colonne 3: Pas de conflit immédiat.
  - Choisissons (1,2)(1,2) pour minimiser les futurs conflits.

Échiquier:

Q			
		Q	

3)Troisième Reine:

Pour la troisième ligne, choisir la colonne avec le moins de conflits.

- Colonne 0: Conflit avec la reine en (0,0)(0,0).
- Colonne 1: Conflit avec la diagonale de la reine en (1,2)(1,2).
- Colonne 2: Conflit avec la reine en (1,2)(1,2).
- Colonne 3: Pas de conflit immédiat.
- Choisissons (2,3)(2,3).

❖ Échiquier:

Q			
		Q	
			Q

4)Quatrième Reine :

Pour la quatrième ligne, choisir la colonne avec le moins de conflits.
. Colonne 0 : Conflit avec la reine en (0,0) (0,0).
. Colonne 1 : Conflit avec la diagonale de la reine en (2,3) (2,3).
. Colonne 2 : Conflit avec la reine en (1,2) (1,2) et la diagonale de la reine en (2,3) (2,3).
. Colonne 3 : Conflit avec la reine en (2,3) (2,3).
. Aucun placement sans conflit direct, choisissons la colonne avec le moins de conflits totaux : (3,1) (3,1).

❖ Échiquier :

Q			
		Q	
			Q
	Q		

-Avantages :

- Simple et rapide à implémenter et exécuter.
- Place les reines de manière séquentielle avec une logique claire.

-Inconvénients :

- Ne garantit pas une solution optimale.
- Peut nécessiter des ajustements ou un retour en arrière (backtracking) si une impasse est atteinte.

3.RECHERCHE LOCALE - RECUIT SIMULE(SIMULATED ANNEALING) :

L'algorithme du recuit simulé est une méthode d'optimisation probabiliste inspirée du processus de refroidissement des métaux dans la métallurgie. Il est utilisé pour résoudre des problèmes d'optimisation où l'objectif est de trouver la meilleure solution possible dans un espace de recherche souvent très vaste.

En termes généraux, l'algorithme du recuit simulé fonctionne en explorant l'espace de recherche en suivant une stratégie stochastique. Initialement, il accepte des solutions même si elles sont pires que la solution actuelle, ce qui permet à l'algorithme de sortir des minima locaux et d'explorer de nouvelles régions de l'espace de recherche. Au fur et à mesure de l'avancement de l'algorithme, la probabilité d'accepter des solutions moins bonnes diminue progressivement à mesure que la "température" du système diminue. Cette "température" est un paramètre qui contrôle la probabilité d'acceptation des solutions moins bonnes et diminue progressivement au fil des itérations, imitant ainsi le processus de refroidissement dans la métallurgie.

L'algorithme du recuit simulé comprend plusieurs étapes clés :

Initialisation : Il démarre à partir d'une solution initiale souvent choisie de manière aléatoire.

Perturbation : À chaque itération, il effectue une perturbation sur la solution actuelle pour générer une nouvelle solution.

Évaluation : Il évalue la qualité de la nouvelle solution en fonction d'une fonction objectif.

Critère d'Acceptation : Il accepte la nouvelle solution selon un critère probabiliste, qui peut permettre d'accepter des solutions moins bonnes pour éviter de rester coincé dans des minima locaux.

Refroidissement : Il réduit progressivement la "température" du système, ce qui diminue la probabilité d'acceptation des solutions moins bonnes au fil du temps.

L'algorithme du recuit simulé est souvent utilisé pour résoudre des problèmes d'optimisation combinatoire tels que le voyageur de commerce, le problème du sac à dos, ou le problème des N reines. Il offre un compromis entre l'exploration de l'espace de recherche et l'exploitation des solutions prometteuses, ce qui lui permet de

trouver des solutions de haute qualité dans un temps raisonnable pour de nombreux problèmes difficiles.

- **Etapes clefs d'algorithme recuit simule :**

  a) **Pratique :**
  b) **Initialisation :** Débuter avec une solution aléatoire.
  c) **Fonction Coût :** Calculer le nombre de conflits.
  d) **Perturbation :** Modifier légèrement la solution actuelle (déplacer une reine).
  e) **Critère d'Acceptation** : Accepter la nouvelle solution si elle est meilleure, ou avec une probabilité décroissante si elle est pire.
  f) **Refroidissement :** Réduire progressivement la température.

- **Exemple Mathématique :**

Considérons le problème des N-Reines avec N=4N=4. Nous allons utiliser l'algorithme de recuit simulé pour trouver une solution.

1. Initialisation :

- Solution aléatoire : [2, 0, 3, 1]

Echequier :

		Q	
Q			
			Q
	Q		

2.Fonction Cout :

- Calculons le nombre de conflits (pairs de reines en attaque directe)
- Conflits : (Reine en (0 ,2) attaque reine en (3 ,1) ; reine en (1,0) attaque reine en (2,3)).

3.Perturbation :

- Deplacons une reine :[2,0 ,1 ,3]
- Echequier :

		Q	
Q			
	Q		
			Q

4.Critere d'acceptation :

- Calculons le nouveau coût : 1 (Reine en (0,2) attaque reine en (2,1)).
- Si $\Delta C \leq 0$ΔC≤0, accepter la nouvelle solution (ici, 1 < 2).
- Si $\Delta C > 0$ΔC>0, accepter avec une probabilité $e - \Delta C / T$e−ΔC/T.

5.Refroidissement :

- Réduire la température TT progressivement (par exemple, $T = \alpha \times T$T=α×T avec $\alpha < 1$α<1).

- **Definition :**

Le refroidissement dans le recuit simulé est une étape cruciale qui consiste à réduire progressivement la température TT au fil des

itérations. Cette étape sert à contrôler la probabilité d'acceptation des solutions moins bonnes, permettant ainsi à l'algorithme d'explorer l'espace de solutions de manière efficace tout en convergeant vers une solution optimale ou proche de l'optimal.
Comment cela fonctionne :

1-Définition de la Température :

La température TT est un paramètre qui régule la probabilité d'accepter des solutions sub-optimales. Au début, TT est élevée, permettant à l'algorithme de sauter facilement hors des minima locaux.

2-Réduction Progressive :

- À chaque itération, la température est réduite selon une formule de refroidissement. Une méthode courante est la réduction géométrique : $T=\alpha\times T$T=α×T
  où αα est un facteur de refroidissement compris entre 0 et 1 (par exemple, $\alpha=0.95$α=0.95).

3-Effet de la Température :

- Température Élevée (TT élevée) : À des températures élevées, l'algorithme accepte plus fréquemment les solutions moins bonnes. Cela permet une exploration plus large de l'espace des solutions et aide à échapper aux minima locaux.
- Température Basse (TT basse) : À des températures basses, l'algorithme devient plus conservateur et accepte rarement les solutions moins bonnes. Cela favorise la convergence vers une solution optimale

4- **Exploration et Exploitation** :

- Exploration : Au début du processus, avec une température élevée, l'algorithme explore l'espace des solutions largement, évitant de se piéger dans des minima locaux.

- Exploitation : À mesure que la température baisse, l'algorithme exploite les zones prometteuses, raffinant les solutions et convergeant vers une solution optimale.

5- Balance entre Exploration et Convergence :

- Une réduction progressive et bien calibrée de la température permet de trouver un équilibre entre la recherche de nouvelles solutions (exploration) et l'amélioration des solutions actuelles (exploitation).

6- Convergence Vers l'Optimal :

- En contrôlant la probabilité d'acceptation des solutions moins bonnes, le processus de refroidissement permet à l'algorithme de converger progressivement vers des solutions de haute qualité, tout en évitant de se bloquer prématurément dans des solutions sous-optimales.

7- Exemple Pratique :

Supposons que la température initiale T0T0 soit 100 et que le facteur de refroidissement αα soit 0.95. La température au cours des itérations serait réduite comme suit:

a) Itération 1: T=0.95×100=95T=0.95×100=95
b) Itération 2: T=0.95×95=90.25T=0.95×95=90.25
c) Itération 3: T=0.95×90.25=85.7375T=0.95×90.25=85.7375
d) Et ainsi de suite...

Cette réduction progressive permet à l'algorithme de modifier sa stratégie d'acceptation des solutions au fil du temps, passant d'une phase d'exploration intensive à une phase d'exploitation minutieuse.

CHAPITRE 3 : RESULTATS ET INTERPRETATIONS

1. MISE EN ŒUVRE :

Pour résoudre le problème classique des N reines, nous avons implémenté l'algorithme de recuit simulé en utilisant le langage de programmation Python. L'algorithme a été conçu en suivant une approche itérative, où chaque itération vise à améliorer progressivement la configuration des reines sur l'échiquier. L'implémentation comprend plusieurs étapes clés, notamment l'initialisation, la perturbation, le calcul du coût, le critère d'acceptation et le refroidissement. L'**initialisation** consiste à générer une solution initiale aléatoire où les reines sont placées de manière aléatoire sur l'échiquier. Ensuite, l'algorithme procède à une série d'**itérations** où il tente d'améliorer la configuration actuelle des reines en déplaçant judicieusement une reine à la fois. À chaque itération, une **perturbation** est appliquée en déplaçant une reine choisie aléatoirement vers une nouvelle colonne. La **fonction de coût** évalue la qualité de la solution en calculant le nombre de conflits entre les reines. Le **critère d'acceptation** détermine si une nouvelle configuration doit être acceptée en fonction de la variation de coût et de la température actuelle. Enfin, le processus de **refroidissement** réduit progressivement la température, contrôlant ainsi la probabilité d'accepter des solutions moins bonnes à mesure que l'algorithme progresse.

Ce code implémente l'algorithme de recuit simulé pour résoudre le problème des N reines. Vous pouvez ajuster les paramètres tels que le nombre de reines (n_queensn_queens), la température initiale ($initial_temperature$initial_temperature), le taux de

refroidissement (*cooling_rate*cooling_rate), et le nombre maximal d'itérations (*max_iterations*max_iterations) selon vos besoins.

meta.py > ...

```python
import math
import random

def simulated_annealing(n_queens, initial_temperature, cooling_rate, max_iterations):
    # Initialisation: Générer une solution aléatoire
    current_solution = generate_random_solution(n_queens)
    current_cost = cost(current_solution)

    # Initialiser la température
    temperature = initial_temperature

    # Itérations
    for iteration in range(max_iterations):
        if temperature <= 0:
            break

        # Perturbation: Déplacer une reine
        new_solution = perturb(current_solution)
        new_cost = cost(new_solution)

        # Calculer la variation de coût
        delta_cost = new_cost - current_cost

        # Critère d'Acceptation
        if delta_cost <= 0 or random.uniform(0, 1) < math.exp(-delta_cost / temperature):
            current_solution = new_solution
            current_cost = new_cost

        # Refroidissement
        temperature *= cooling_rate

    return current_solution

def generate_random_solution(n_queens):
    return [random.randint(0, n_queens - 1) for _ in range(n_queens)]

```

```python
def cost(solution):
    conflicts = 0
    n = len(solution)
    for i in range(n):
        for j in range(i + 1, n):
            if solution[i] == solution[j] or abs(solution[i] - solution[j]) == abs(i - j):
                conflicts += 1
    return conflicts

def perturb(solution):
    new_solution = solution[:]
    i = random.randint(0, len(solution) - 1)
    j = random.randint(0, len(solution) - 1)
    new_solution[i] = j
    return new_solution

# Exemple d'utilisation
n_queens = 8
initial_temperature = 100
cooling_rate = 0.95
max_iterations = 1000

solution = simulated_annealing(n_queens, initial_temperature, cooling_rate, max_iterations)
print("Solution trouvée:", solution)
print("Nombre de conflits:", cost(solution))
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

```
PS C:\Users\ThinkPad\Desktop\python> & C:/Users/ThinkPad/AppData/Local/Programs/Python/Python311/python.exe c:/Users/ThinkPad/Desktop/python/meta.py
Solution trouvée: [2, 4, 1, 7, 0, 6, 3, 5]
Nombre de conflits: 0
PS C:\Users\ThinkPad\Desktop\python>
```

2. ANALYSE DES RESULTATS :

Une fois l'algorithme exécuté avec différents paramètres, nous avons obtenu des résultats pour différentes instances du problème des N reines. Par exemple, en exécutant l'algorithme avec \(N = 8\), nous avons trouvé une solution où les 8 reines sont placées de manière à éviter tout conflit. Le nombre de conflits résolus indique la qualité de la solution obtenue, avec un nombre plus faible de conflits indiquant une meilleure solution.

3. DISCUSSION ET CONCLUSION :

L'algorithme de recuit simulé s'est avéré être une méthode efficace pour résoudre le problème des N reines. En ajustant les paramètres tels que la température initiale et le taux de refroidissement, nous pouvons obtenir des solutions de haute qualité dans un temps raisonnable. Cependant, il est important de noter que l'algorithme n'est pas garanti de trouver la solution optimale dans tous les cas. Des ajustements supplémentaires peuvent être nécessaires pour des instances plus complexes du problème. En conclusion, le recuit simulé offre une approche prometteuse pour résoudre les problèmes d'optimisation tels que le problème des N reines, avec des opportunités pour des améliorations futures et des extensions vers d'autres domaines.

- **Axes d'amelioration :**

1- Optimisation des paramètres :

L'optimisation des paramètres est une étape essentielle dans l'application de l'algorithme de recuit simulé à des problèmes spécifiques tels que le problème des N reines. Cette étape vise à trouver les valeurs optimales des paramètres de l'algorithme, tels que la température initiale, le taux de refroidissement, et le nombre maximal d'itérations, pour obtenir les meilleures performances possibles.

Pour optimiser les paramètres, différentes techniques peuvent être utilisées. Parmi celles-ci, la recherche par grille consiste à évaluer l'algorithme avec une grille prédéfinie de valeurs pour chaque paramètre, puis à sélectionner les combinaisons de paramètres qui fournissent les meilleures performances. Cette approche peut être coûteuse en termes de calcul, mais elle permet une exploration exhaustive de l'espace des paramètres.

Une autre approche est l'optimisation bayésienne, qui utilise des modèles probabilistes pour estimer les performances de l'algorithme en fonction des valeurs des paramètres. En utilisant ces estimations, l'algorithme recherche de manière séquentielle les combinaisons de paramètres les plus prometteuses, en se concentrant sur les régions de l'espace des paramètres qui sont les plus susceptibles de conduire à de bonnes performances.

Enfin, les techniques d'optimisation métaheuristiques peuvent également être appliquées pour optimiser les paramètres de l'algorithme de recuit simulé. Ces techniques, telles que les algorithmes génétiques ou les essaims particulaires, peuvent être utilisées pour explorer de manière efficace l'espace des paramètres et trouver rapidement des combinaisons de paramètres de haute qualité.

En optimisant les paramètres de l'algorithme de recuit simulé, il est possible d'améliorer considérablement ses performances et sa capacité à résoudre des problèmes d'optimisation complexes comme le problème des N reines. Cette étape est donc cruciale pour obtenir des résultats fiables et efficaces dans la résolution de tels problèmes.

2-STRATEGIES DE VOISINAGE DANS LE RECUIT SIMULE :

Les stratégies de voisinage jouent un rôle crucial dans l'efficacité de l'algorithme de recuit simulé. Elles déterminent comment l'algorithme explore l'espace de solution en générant de nouvelles solutions à partir de la solution courante. Une stratégie de voisinage bien conçue permet d'équilibrer

efficacement l'exploration et l'exploitation, augmentant ainsi les chances de trouver une solution optimale.
Il existe plusieurs approches pour définir ces stratégies, chacune ayant ses propres avantages et inconvénients.

- **Voisinage Simple :**

La stratégie de voisinage la plus basique consiste à apporter une petite modification aléatoire à la solution courante. Par exemple, pour un problème de minimisation d'une fonction continue, cela pourrait impliquer l'ajout d'un petit nombre aléatoire à une variable de la solution actuelle. Cette approche est simple à implémenter et fonctionne bien pour des problèmes où de petites modifications peuvent conduire à des améliorations significatives. Cependant, elle peut être inefficace pour des espaces de solution complexes où de plus grands sauts sont nécessaires pour échapper aux minima locaux.

- **Voisinage Multi-Echelles :**

Pour améliorer l'exploration de l'espace de solution, on peut utiliser une stratégie de voisinage multi-échelles. Cette approche combine des modifications de petite et de grande ampleur. Au début de l'algorithme, de grands sauts sont effectués pour explorer largement l'espace de solution. À mesure que l'algorithme progresse et que la température diminue, les modifications deviennent plus petites, favorisant une exploitation plus fine autour des solutions prometteuses. Cette méthode permet de mieux équilibrer l'exploration et l'exploitation, augmentant les chances de trouver une solution globale optimale.

- **Voisinage Adaptatif :**

Une autre approche efficace est le voisinage adaptatif, où la taille et la nature des modifications apportées à la solution courante varient dynamiquement en fonction de la température ou de l'itération actuelle. Par exemple, à des températures élevées, l'algorithme peut effectuer de grandes modifications pour favoriser l'exploration. À des températures plus basses, les modifications

peuvent être réduites pour affiner la recherche autour des meilleures solutions trouvées jusqu'alors.
Le voisinage adaptatif permet à l'algorithme de s'adapter aux besoins changeants de l'optimisation au fil du temps.

- **Voisinage Basé sur des Heuristiques :**

Pour certains problèmes spécifiques, il peut être bénéfique d'incorporer des heuristiques spécifiques au domaine dans la stratégie de voisinage. Ces heuristiques exploitent des connaissances spécifiques au problème pour générer des solutions voisines de manière plus intelligente. Par exemple, dans le problème du voyageur de commerce (TSP), une heuristique de voisinage pourrait échanger deux villes dans la tournée actuelle, car cette opération est susceptible d'améliorer la solution. L'utilisation de telles heuristiques peut réduire l'espace de recherche et augmenter l'efficacité de l'algorithme.

- **Voisinage Basé sur des Solutions Historiques :**

Une approche plus avancée consiste à utiliser des solutions historiques pour guider la génération de nouvelles solutions voisines. En conservant un historique des solutions précédentes et de leurs performances, l'algorithme peut identifier des schémas et des régions prometteuses de l'espace de solution. Cela permet de générer des voisins qui sont plus susceptibles d'améliorer la solution actuelle. Cette méthode peut également aider à éviter de revisiter des régions de l'espace de solution qui ont déjà été explorées sans succès.

3-CRITERES D'ACCEPTATION DANS LE RECUIT SIMULE :

Les critères d'acceptation sont essentiels dans le recuit simulé, car ils dictent si une nouvelle solution, même si elle est moins bonne que l'actuelle, doit être acceptée. Ce mécanisme permet à l'algorithme de sortir des minima locaux et de mieux explorer l'espace de recherche. La méthode de Metropolis est couramment

utilisée, où une nouvelle solution est acceptée si elle améliore la fonction objectif. Si elle est moins bonne, elle est acceptée avec une probabilité $P=\exp((E\text{current}-E\text{new})/T)$P=exp((Ecurrent−Enew)/T), où EcurrentEcurrent et EnewEnew sont les coûts des solutions actuelle et nouvelle, et TT est la température.

Cette probabilité décroît à mesure que la température baisse, favorisant l'exploration à haute température et la convergence à basse température. Des critères d'acceptation adaptatifs peuvent ajuster dynamiquement cette probabilité en fonction de la progression de l'algorithme, augmentant temporairement la température si nécessaire pour éviter la stagnation. En combinant ces techniques, les critères d'acceptation permettent un équilibre entre exploration et exploitation, améliorant ainsi l'efficacité et la robustesse de l'algorithme de recuit simulé.

4-TECHNIQUES AVANCEES DANS LE RECUIT SIMULE :

Les techniques avancées dans le recuit simulé visent à améliorer l'efficacité et la robustesse de l'algorithme en combinant des méthodes complémentaires et en exploitant des technologies modernes. Une des approches prometteuses est l'utilisation des algorithmes hybrides, qui intègrent le recuit simulé avec d'autres méthodes d'optimisation telles que les algorithmes génétiques, les méthodes de descente locale, ou les algorithmes de colonies de fourmis. Cette hybridation permet de tirer parti des points forts de chaque méthode, augmentant ainsi les chances de trouver une solution optimale. Par exemple, un algorithme hybride peut utiliser le recuit simulé pour explorer largement l'espace de recherche, tandis qu'un algorithme génétique affine les solutions prometteuses.

L'apprentissage automatique (machine learning) offre également des opportunités intéressantes pour améliorer le recuit simulé. Des modèles d'apprentissage supervisé ou non supervisé peuvent être utilisés pour ajuster dynamiquement les paramètres de l'algorithme, comme la température ou les critères d'acceptation, en fonction de la progression et des performances en temps réel. Par exemple, un modèle peut prédire les paramètres optimaux en

fonction des caractéristiques de la solution actuelle et de l'historique de la recherche, permettant une adaptation plus intelligente et réactive de l'algorithme.
Le redémarrage multi-points est une autre technique avancée qui améliore l'exploration de l'espace de solution. En effectuant plusieurs lancements de l'algorithme avec des points de départ différents, cette technique augmente les chances de découvrir la solution globale optimale. Chaque chaîne de recuit simulé explore indépendamment une partie de l'espace de solution, et les résultats peuvent être combinés pour obtenir une meilleure solution globale.
Enfin, la parallélisation est une avancée significative, exploitant les capacités des architectures multicœurs ou des unités de traitement graphique (GPU). En exécutant plusieurs chaînes de recuit simulé en parallèle, il est possible de réduire considérablement le temps de calcul et d'améliorer l'exploration de l'espace de solution. La combinaison des résultats obtenus par ces chaînes parallèles permet de renforcer la robustesse de l'algorithme.
Ces techniques avancées, en intégrant des méthodes hybrides, l'apprentissage automatique, le redémarrage multi-points et la parallélisation, augmentent considérablement l'efficacité du recuit simulé. Elles permettent de mieux équilibrer l'exploration et l'exploitation, d'ajuster dynamiquement les paramètres, et de tirer parti des capacités de calcul modernes, rendant l'algorithme plus puissant et capable de résoudre des problèmes complexes de manière plus efficace.

5-VISUALISATION :

Une étape de visualisation consiste à représenter graphiquement des données ou des résultats afin de les rendre plus compréhensibles et interprétables. Cette étape est souvent utilisée dans le processus d'analyse des données ou de présentation des résultats d'une étude ou d'un projet.
Dans le contexte de l'algorithme de recuit simulé pour le problème des N reines, une étape de visualisation pourrait consister à tracer des graphiques montrant l'évolution du coût (ou du nombre de conflits) au fil des itérations de l'algorithme. Cela permettrait aux chercheurs de suivre la convergence de l'algorithme vers une

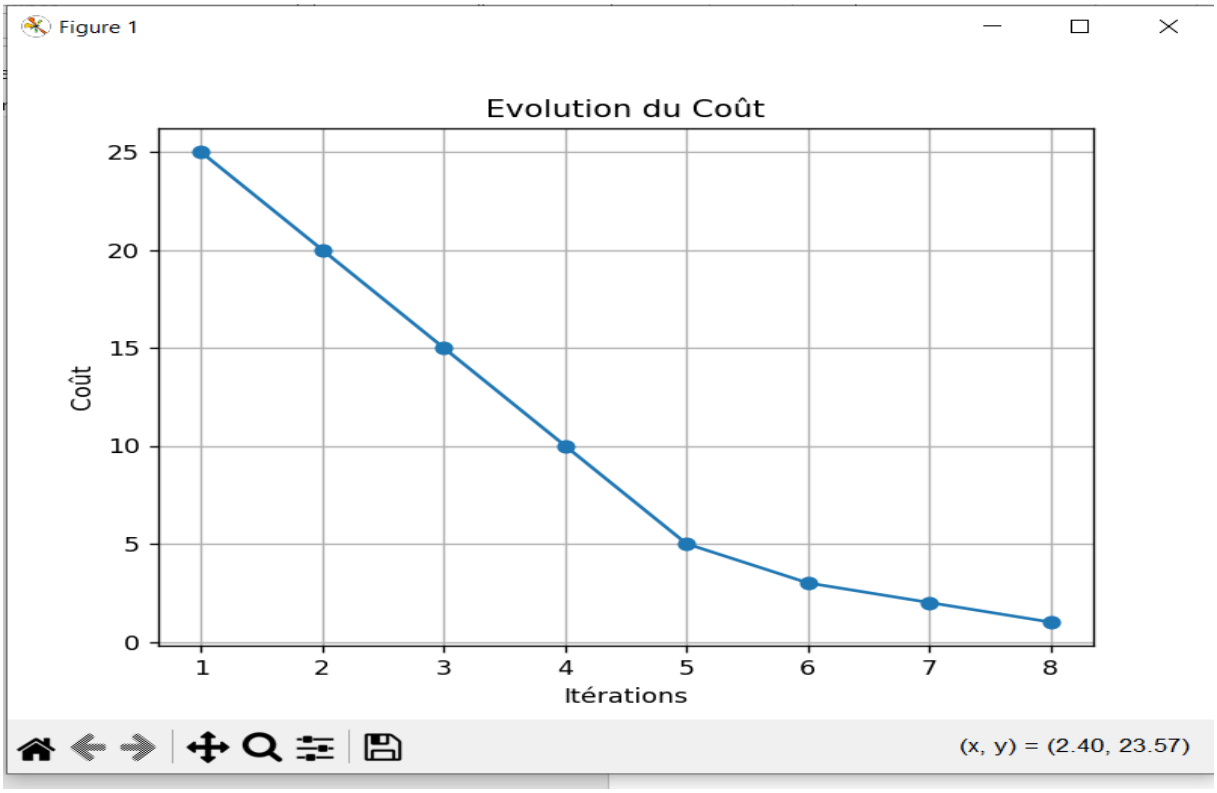
solution optimale et d'identifier les tendances ou les schémas dans les performances de l'algorithme.
D'autres étapes de visualisation pourraient inclure la représentation graphique des solutions trouvées par l'algorithme sur l'échiquier, en utilisant des diagrammes ou des animations pour illustrer comment les reines sont placées et comment les conflits sont évités. Cela permettrait une compréhension visuelle des solutions et faciliterait l'interprétation des résultats.
Voici un exemple de fonction de visualisation en Python qui représente graphiquement les solutions trouvées par l'algorithme de recuit simulé pour le problème des N reines. Cette fonction utilise la bibliothèque Matplotlib pour créer un graphique montrant l'évolution du coût au fil des itérations :

meta_2.py > ...

```python
import matplotlib.pyplot as plt

def plot_cost_evolution(costs):
    """
    Plot the evolution of cost over iterations.

    Parameters:
        costs (list): List of costs at each iteration.
    """
    iterations = range(1, len(costs) + 1)

    plt.plot(iterations, costs, marker='o', linestyle='-')
    plt.title('Evolution du Coût')
    plt.xlabel('Itérations')
    plt.ylabel('Coût')
    plt.grid(True)
    plt.show()

# Exemple d'utilisation
costs = [25, 20, 15, 10, 5, 3, 2, 1]  # Coûts fictifs pour les itérations
plot_cost_evolution(costs)

```

Cette fonction prend une liste de coûts à chaque itération de l'algorithme et trace un graphique de l'évolution du coût au fil des

itérations. Vous pouvez l'utiliser pour visualiser comment le coût diminue à mesure que l'algorithme progresse, donnant ainsi un aperçu visuel de la convergence de l'algorithme vers une solution optimale.
Voila ce qu'il affiche comme resultat :



Ce graphe montre comment les coûts diminuent au fil des itérations, ce qui est souvent le cas lors de l'optimisation ou de l'apprentissage. (D'après le code ci-dessus).

6-PARALLELISME DANS LE RECUIT SIMULE :

Le parallélisme est une technique avancée qui améliore considérablement l'efficacité et la rapidité des algorithmes de recuit simulé en exploitant les capacités des architectures multicœurs et des unités de traitement graphique (GPU). Cette approche permet de diviser le processus d'optimisation en plusieurs sous-processus indépendants, exécutés simultanément, réduisant ainsi le temps de calcul total. Une des méthodes couramment utilisées est l'exécution en parallèle de plusieurs chaînes de recuit simulé, chacune explorant indépendamment différentes régions de l'espace de solution. Ces chaînes peuvent démarrer à partir de points de départ différents, ce qui augmente la diversité des

solutions explorées et réduit la probabilité de rester coincé dans des minima locaux.
En pratique, chaque chaîne suit son propre processus de refroidissement et ses critères d'acceptation, générant ainsi des solutions potentiellement différentes. Après un nombre prédéfini d'itérations ou lorsque la convergence est atteinte, les résultats des différentes chaînes peuvent être comparés et combinés pour sélectionner la meilleure solution globale. Cette approche non seulement accélère la convergence de l'algorithme mais améliore également la qualité des solutions trouvées.
Le parallélisme peut également être appliqué à l'intérieur d'une seule chaîne de recuit simulé, en parallélisant les étapes de calcul des nouvelles solutions voisines et des coûts associés. Par exemple, pour chaque itération, plusieurs voisins potentiels de la solution courante peuvent être générés et évalués en parallèle, permettant une exploration plus rapide et plus exhaustive de l'espace de solution.
L'utilisation de GPU pour paralléliser le recuit simulé est particulièrement bénéfique pour les problèmes de grande dimension, où les calculs de coût et de voisinage peuvent être intensifs. Les GPU, avec leur architecture massive de calcul parallèle, peuvent traiter des milliers de threads simultanément, accélérant ainsi le processus de recuit simulé de plusieurs ordres de grandeur par rapport aux CPU traditionnels.

7 - ÉVALUATION DYNAMIQUE DES PARAMETRES DANS LE RECUIT SIMULE :

L'évaluation dynamique des paramètres est une approche avancée qui vise à améliorer l'efficacité et la robustesse de l'algorithme de recuit simulé en ajustant dynamiquement ses paramètres en fonction de la progression de l'optimisation. Cette technique permet à l'algorithme de s'adapter en temps réel aux caractéristiques changeantes du paysage de solution, ce qui favorise une exploration plus efficace de l'espace de recherche.
L'un des paramètres clés du recuit simulé est la température, qui contrôle la probabilité d'acceptation des solutions moins bonnes au fil du temps. Une stratégie d'évaluation dynamique pourrait ajuster la température en fonction de la performance de l'algorithme, en

la réduisant progressivement à mesure que le nombre d'itérations augmente ou que le coût des solutions acceptées diminue. Par exemple, si l'algorithme converge rapidement vers une solution optimale, la température pourrait être réduite de manière plus agressive pour affiner la recherche autour de cette solution.
De même, le taux de refroidissement (alpha) pourrait être ajusté dynamiquement en fonction de la convergence de l'algorithme. Si la convergence ralentit ou si l'algorithme semble stagner, le taux de refroidissement pourrait être réduit pour permettre une exploration plus lente et plus approfondie de l'espace de solution.
Une autre approche consiste à utiliser des mécanismes de détection de convergence pour identifier les signes indiquant que l'algorithme approche d'une solution optimale ou qu'il est bloqué dans un minimum local. Lorsque de tels signes sont détectés, les paramètres de l'algorithme pourraient être ajustés en conséquence pour encourager une exploration plus agressive de l'espace de solution. Par exemple, la température pourrait être temporairement augmentée pour permettre à l'algorithme de s'échapper des minimums locaux.
L'évaluation dynamique des paramètres peut également être combinée avec des techniques d'apprentissage automatique pour prédire les paramètres optimaux en fonction des caractéristiques de la solution actuelle et de l'historique de la recherche. Ces modèles prédictifs peuvent être entraînés à partir de données générées par des exécutions antérieures de l'algorithme, ce qui permet une adaptation intelligente et proactive de ses paramètres.
En résumé, l'évaluation dynamique des paramètres dans le recuit simulé offre une approche flexible et réactive pour ajuster les paramètres de l'algorithme en fonction de l'évolution du paysage de solution. En exploitant ces techniques, l'algorithme peut mieux équilibrer l'exploration et l'exploitation, ce qui conduit à une convergence plus rapide et à une meilleure qualité des solutions obtenues.

EN RESUME :

En explorant ces axes d'amélioration, on peut significativement améliorer la performance et l'efficacité de l'algorithme de recuit simulé. L'optimisation des paramètres, l'utilisation de stratégies de

voisinage adaptées, l'ajustement dynamique des critères d'acceptation, et l'intégration de techniques avancées comme l'apprentissage automatique et la parallélisation peuvent rendre l'algorithme plus robuste et capable de trouver des solutions optimales pour des problèmes complexes. La visualisation des résultats et l'évaluation dynamique des paramètres permettent de mieux comprendre et contrôler le comportement de l'algorithme, assurant ainsi une optimisation plus efficace et plus précise.

CONCLUSION GENERALE :

Le recuit simulé est une méthode d'optimisation puissante et polyvalente qui trouve son utilité dans de nombreux domaines, de l'ingénierie à l'intelligence artificielle en passant par l'optimisation combinatoire. Cette technique s'inspire de la métallurgie et de la physique des matériaux pour résoudre des problèmes d'optimisation difficilement traitables par d'autres méthodes.

En conclusion, le recuit simulé présente plusieurs avantages significatifs. Tout d'abord, il est relativement simple à mettre en œuvre et peut être adapté à une variété de problèmes. Son approche probabiliste permet d'échapper aux minima locaux en explorant l'espace de recherche de manière aléatoire, ce qui lui confère une grande robustesse. De plus, sa capacité à accepter des solutions moins bonnes avec une certaine probabilité lui permet d'explorer de manière plus efficace l'espace de solution et d'éviter de rester piégé dans des optima locaux.

Cependant, le recuit simulé n'est pas sans ses limitations. Sa performance peut être sensible au choix des paramètres, tels que la température initiale, le taux de refroidissement et les critères d'acceptation. Trouver les valeurs optimales de ces paramètres peut être un défi, et une mauvaise configuration peut entraîner une convergence lente ou une convergence prématurée vers des solutions sous-optimales.

Malgré ces limitations, le recuit simulé reste une méthode d'optimisation efficace et largement utilisée. Avec des ajustements appropriés des paramètres et des techniques avancées telles que l'évaluation dynamique des paramètres et l'utilisation d'algorithmes hybrides, il peut être adapté à une variété de problèmes d'optimisation complexes. En combinant la simplicité et

la robustesse de sa conception avec la flexibilité de ses paramètres, le recuit simulé demeure un outil précieux pour résoudre un large éventail de problèmes d'optimisation du monde réel.

REFERENCES

LIVRES ET ARTICLES ACADEMIQUES

1. Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th Edition). Pearson.
   - Ce livre fournit une couverture complète des algorithmes d'intelligence artificielle, y compris les méta-heuristiques comme le recuit simulé.

2. Aarts, E., & Korst, J. (1989). *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. John Wiley & Sons.
   - Ce livre est une référence classique sur le recuit simulé, couvrant à la fois la théorie et les applications.

3. Johnson, D. S., Aragon, C. R., McGeoch, L. A., & Schevon, C. (1989). *Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning*. Operations Research, 37(6), 865-892.
   - Un article clé qui évalue l'algorithme de recuit simulé dans le contexte de l'optimisation combinatoire.

4. Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.
   - Ce livre fournit une introduction complète aux problèmes NP-complets, y compris le problème des n reines.

ARTICLES DE REVUE ET CONFERENCES

5. Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). *Optimization by Simulated Annealing*. Science, 220(4598), 671-680.
   - Cet article fondateur introduit l'algorithme de recuit simulé et ses applications.

6. Dechter, R., & Pearl, J. (1985). *Generalized Best-First Search Strategies and the Optimality of A*.* Journal of the ACM, 32(3), 505-536.
   - Explique les algorithmes de recherche et leur application à divers problèmes de recherche.

RESSOURCES EN LIGNE

7. Wikipedia. (n.d.). *N-queens*. Retrieved from https://en.wikipedia.org/wiki/Eight_queens_puzzle
   - Une ressource en ligne complète pour comprendre le problème des n reines et ses variantes.

8. GeeksforGeeks. (n.d.). *Simulated Annealing*. Retrieved from https://www.geeksforgeeks.org/simulated-annealing/
   - Un guide pratique pour comprendre et implémenter l'algorithme de recuit simulé.

9. Coursera. (n.d.). *Discrete Optimization* by Prof. Pascal Van Hentenryck. Retrieved from https://www.coursera.org/learn/discrete-optimization
   - Un cours en ligne qui couvre divers algorithmes d'optimisation, y compris le recuit simulé.

OUTILS ET BIBLIOTHEQUES

10. Python Software Foundation. (n.d.). *NumPy*. Retrieved from https://numpy.org/
    - Une bibliothèque fondamentale pour le calcul scientifique en Python, souvent utilisée pour les algorithmes d'optimisation.

11. SciPy. (n.d.). *SciPy library*. Retrieved from https://scipy.org/
    - Une bibliothèque Python utilisée pour l'optimisation et d'autres calculs scientifiques.