Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Informatique
Département IA et SD

Rapport Mini Projet

Module : Métaheuristique

Résolution de problème des N-reines avec les métaheuristiques

---

Travail présenté par :

- **BENKOUITEN Aymen————-191931046409**
- **KENAI Imad Eddine————-191932017671**

2022/2023

Table des matières

Table des figures

Liste des tableaux

Introduction Générale

Les heuristiques et les métaheuristiques sont des techniques puissantes utilisées dans les problèmes d'optimisation où une solution exacte est difficile ou impraticable à obtenir. Dans ce projet,on va continuer notre travail sur le problème des N-Reines en explorons deux métaheuristiques populaires, l'Algorithme Génétique (GA) et l'Optimisation par Essaim Particulaire (PSO), et les implémentons pour résoudre le problème des N-Reines. Notre objectif est de comparer les performances de ces deux algorithmes et d'évaluer leur efficacité dans la recherche de bonnes solutions au problèmes.

Première partie

Approche espace des solutions.

0.1 Description des concepts :

0.1.1 Solution :

Une solution est une réponse candidate à un problème qui est résolu à l'aide d'un algorithme de métaheuristique. En métaheuristique, une solution est généralement représentée comme un ensemble de valeurs, souvent appelées variables de décision ou paramètres, qui sont choisis d'une manière particulière pour satisfaire les contraintes et les objectifs du problème.

0.1.2 Espace de solutions :

L'espace de solutions, également connu sous le nom d'espace de recherche, désigne l'ensemble de toutes les solutions possibles qui peuvent être générées par l'algorithme. C'est la gamme de valeurs que peuvent prendre les variables de décision. L'espace de solutions peut être discret ou continu, selon la nature du problème.

0.1.3 Taille de l'espace de recherche :

La taille de l'espace de recherche est le nombre de solutions possibles qui peuvent être générées par l'algorithme. Elle représente le nombre total de solutions candidates qui doivent être évaluées par l'algorithme pour trouver la solution optimale. La taille de l'espace de recherche est déterminée par le nombre de variables de décision et la plage de valeurs que chaque variable peut prendre. La taille de l'espace de recherche peut être un facteur crucial pour déterminer l'efficacité de l'algorithme de métaheuristique, car la recherche d'un grand espace de recherche peut être coûteuse en termes de temps de calcul et de temps d'exécution. Dans certains cas, il est possible de réduire la taille de l'espace de recherche en appliquant des connaissances ou des contraintes spécifiques au problème au processus de recherche, ce qui améliore l'efficacité de l'algorithme.

0.1.4 Fonction de fitness :

Une fonction de fitness est une fonction mathématique qui mesure la qualité d'une solution dans l'espace de solutions. La fonction de fitness est définie en fonction des objectifs et des contraintes du problème. Elle attribue une valeur numérique, appelée fitness ou valeur d'objectif, à chaque solution candidate en fonction de la mesure dans laquelle elle satisfait les contraintes et les objectifs du problème. La fonction de fitness est utilisée par l'algorithme de métaheuristique pour évaluer la qualité des solutions générées pendant le processus de recherche et pour guider la recherche vers de meilleures solutions. L'objectif de l'algorithme de métaheuristique est de trouver la solution avec la valeur de fitness la plus élevée.

0.2 Algorithmes Génétiques (GA) :

Les AG sont une classe d'algorithmes métaheuristiques inspirés du processus de sélection naturelle et de génétique. Ils imitent la façon dont les organismes biologiques évoluent et s'adaptent au fil du temps en faisant évoluer une population de solutions potentielles à un problème grâce à un processus de sélection, de croisement et de mutation. [2] La population de solutions est représentée sous la forme d'un ensemble de chromosomes, qui codent les solutions candidates. La

fonction de fitness est utilisée pour évaluer la qualité de chaque chromosome de la population, et les chromosomes performants sont sélectionnés pour être recombinés (c'est-à-dire croisés) afin de créer de nouvelles progénitures potentiellement plus performantes. La mutation introduit des changements aléatoires dans les progénitures pour explorer davantage l'espace des solutions.

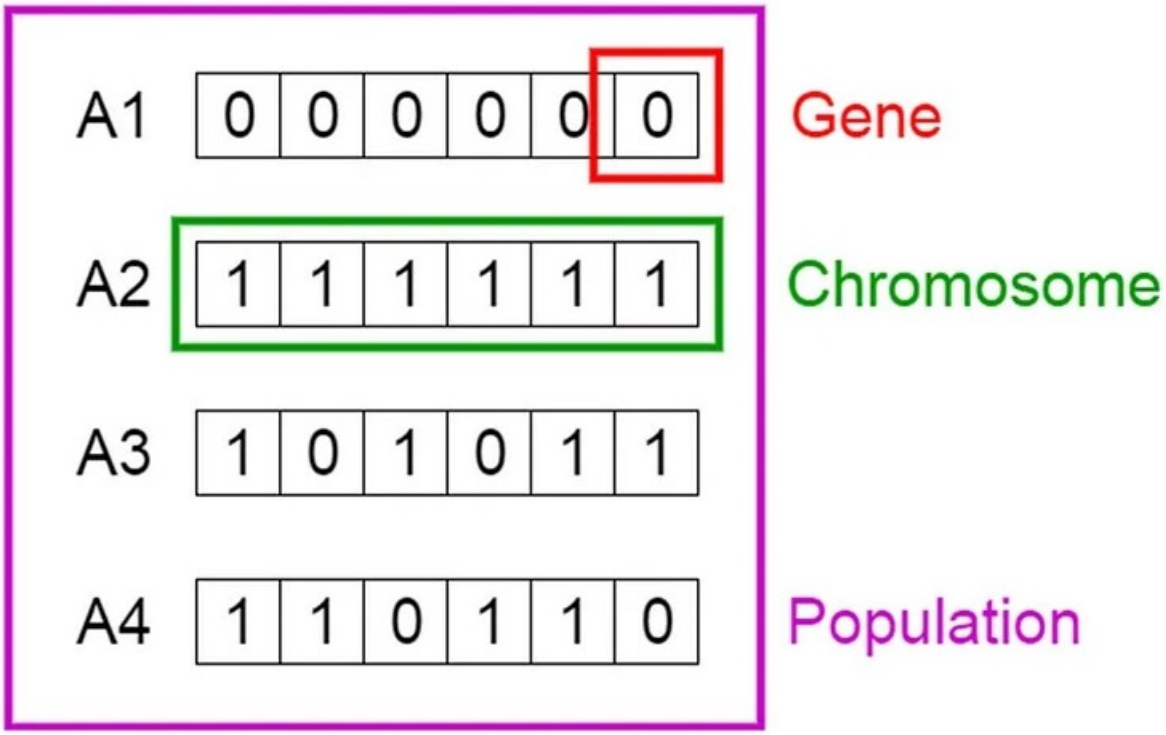


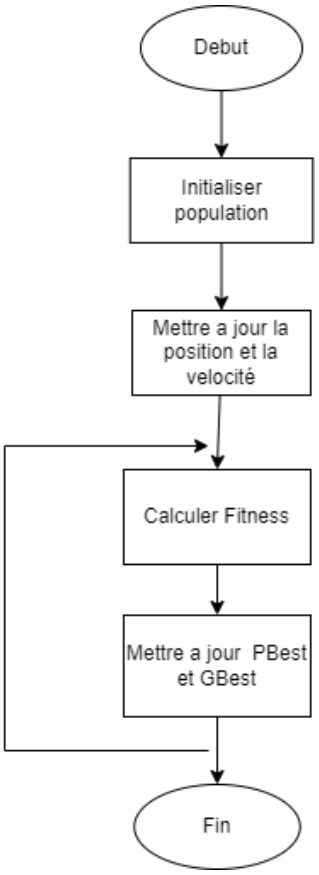
FIGURE 1 – Algorithme génétique

Pseudo code :

Initialiser la population P de taille N
for $i = 1$ to $MaxGen$ **do**
    Évaluer la fitness de chaque individu dans P
    Sélectionner les individus pour la génération suivante
    Appliquer les opérateurs génétiques (croisement, mutation) pour créer de nouveaux individus
    Remplacer l'ancienne population par la nouvelle population
end for
retourner Le meilleur individu

0.3 Optimisation par essaims particulaires (PSO) :

La PSO est une autre classe d'algorithmes métaheuristiques inspirés du comportement collectif des essaims, tels que les vols d'oiseaux et les bancs de poissons. La PSO fonctionne en simulant le mouvement d'un essaim de particules dans un espace de recherche multidimensionnel.[1] Chaque particule représente une solution potentielle au problème en cours d'optimisation, et son mouvement est déterminé par sa position actuelle, sa vitesse et la meilleure position trouvée par l'essaim jusqu'à présent. La fonction de fitness est utilisée pour évaluer la qualité de chaque particule, et les particules performantes influencent le mouvement du reste de l'essaim. La PSO a été appliquée avec succès à un large éventail de problèmes d'optimisation, tels que l'optimisation de fonctions, le regroupement de données et la formation de réseaux neuronaux.



Pseudo code :

Initialiser la population de particules avec des positions et des vélocités aléatoires
Initialiser la meilleure solution globale g
for $i = 1$ to $MaxGen$ **do**
  for all particules dans la population **do**
    Mettre à jour la vélocité de la particule en fonction de sa meilleure solution et de la meilleure solution globale
    Mettre à jour la position de la particule en fonction de sa nouvelle vélocité
    Mettre à jour la meilleure solution personnelle de la particule si nécessaire
  end for
  Mettre à jour la meilleure solution globale en fonction de la population actuelle
end for
retourner La meilleure solution globale

Deuxième partie

Résolution du Problème N-reine a l'aide des algorithmes évolutifs

0.4 Modélisation du problème :

0.4.1 Solution :

Une solution d'un problème de taille N x N est représenté par un vecteur d'entiers de taille N.
Les entiers inclus dans [0 .. N [et chaque case "i" représente la position de la reine à la ligne "i".

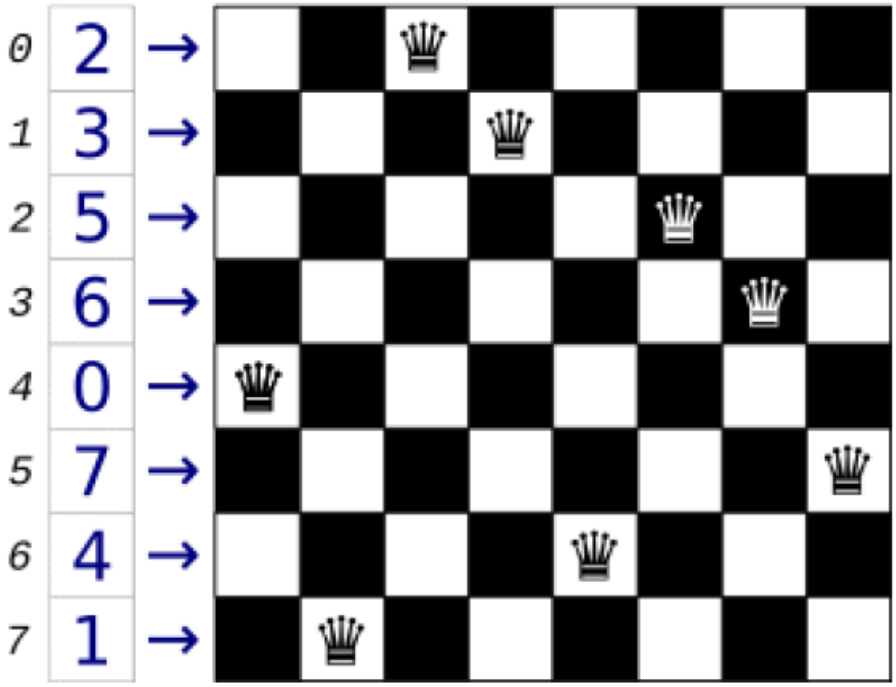


FIGURE 2 – Modélisation de la solution du problème

0.4.2 Espace des solutions :

L'espace des représente un ensemble des solutions d'une taille fixe ,où chaque solution peut être une solution valide au problème.

0.4.3 Fonction d'évaluation (Fitness) :

La fonction d'évaluation consiste a calculer le nombre des conflits pour chaque reine et retourner la somme de tous les conflits.

0.5 Algorithme génétique :

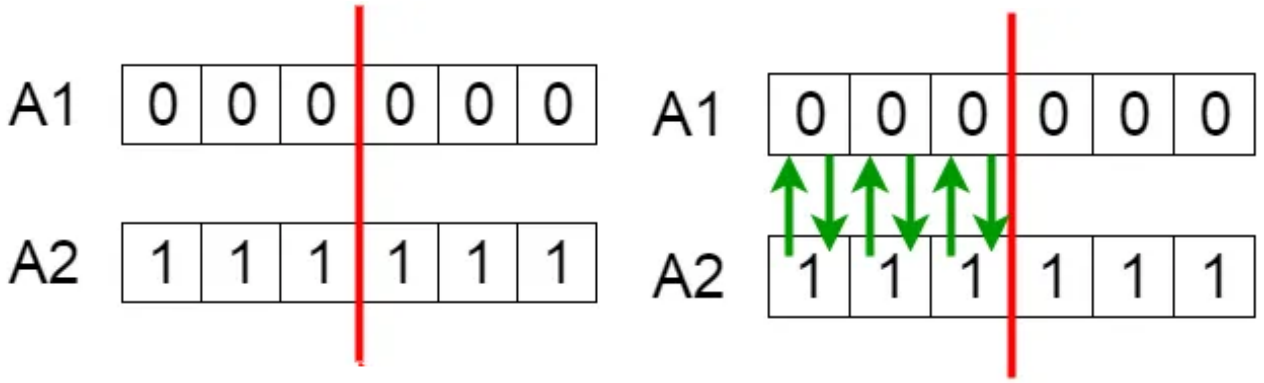
La solution est représentée par un tableau de taille N (N est la taille d'échiquier), où chaque case représente la position d'une reine sur l'échiquier.

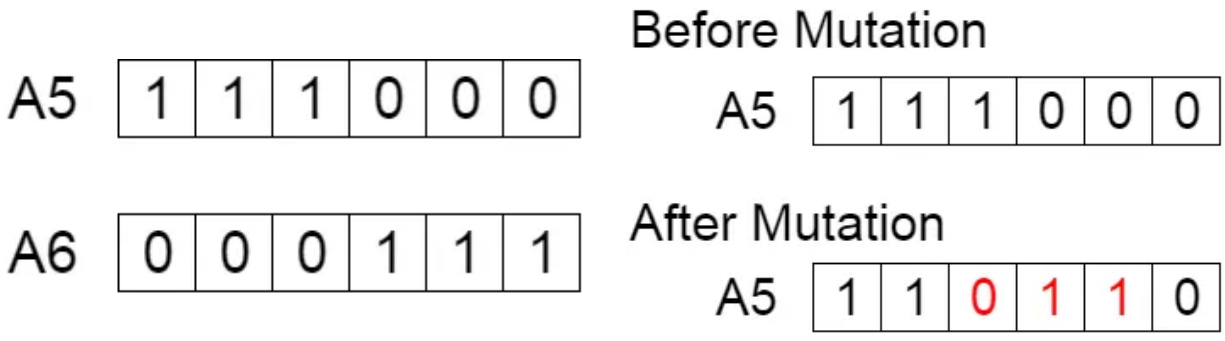
Après avoir initié une population de taille spécifique, on itère sur un nombre maximal de générations (MAXGEN).

À chaque itération, on sélectionne un pourcentage de la population initiale en fonction du taux de croisement. Cela signifie qu'on choisit les parents pour la prochaine génération et les croisez pour produire des descendants. Le choix de chaque parent se fait d'une manière aléatoire.

Ensuite, on applique une mutation à un pourcentage des fils générés en fonction du taux de mutation. Cela permet d'introduire de la diversité dans la population et d'explorer de nouveaux espaces de solutions et d'éviter de rester coincé dans des optima locaux.

En fin de compte, on combine les enfants avec la population initiale et on trie la nouvelle population en fonction de leur score de fitness. On sélectionne ensuite les meilleurs individus(sélection élitaire) pour la prochaine génération pour garantir que les meilleures solutions de chaque génération sont conservées.
On continue ce processus itératif jusqu'à ce qu'une solution satisfaisante soit trouvée ou que le critère d'arrêt soit atteint.





Pseudo code :

1: Initialiser la population P de taille N
2: **for** $i = 1$ to $MaxGen$ **do**
3: Sélectionner un sous-ensemble de parents de P en fonction du taux de crossover
4: Appliquer le crossover pour créer un ensemble de nouveaux enfants
5: Sélectionner un sous-ensemble d'enfants de la population nouvellement créée en fonction du taux de mutation
6: Appliquer la mutation aux enfants sélectionnés
7: Fusionner les parents et les enfants dans une population combinée
8: Sélectionner les meilleurs individus en fonction de leur fitness pour former la prochaine génération
9: **end for**
10: **retourner** Le meilleur individu

0.6 Optimisation par essaims particulaires (PSO) :

Le PSO maintient un ensemble de particules, chacune représentant une solution potentielle au problème. Chaque particule se déplace dans l'espace de recherche, en ajustant sa position en fonction de sa propre expérience et de celle des autres particules.

Dans notre implémentation, la mise à jour de la position de chaque particule est basée sur trois facteurs : la position personnelle de la particule qui a donné la meilleure solution (PBest), la position globale du meilleur individu de l'essaim (GBest), et une particule choisie au hasard.

Nous utilisons deux paramètres c1 et c2 souvent appelés les "coefficients d'accélération" pour contrôler l'influence de ces facteurs sur la position de chaque particule.

Si un nombre aléatoire généré est inférieur à c1, la particule applique une opération de crossover avec la meilleure solution globale. Cette étape permet à la particule d'explorer l'espace de recherche global en utilisant la meilleure position trouvée par n'importe quelle particule dans l'ensemble de la population comme référence.

Si le nombre aléatoire est compris entre c1 et c2, la particule applique une opération de crossover avec sa meilleure solution personnelle. Cette étape permet à la particule d'exploiter l'espace de recherche en utilisant sa propre meilleure position comme référence.

Si le nombre aléatoire est supérieur à c2, la particule applique une opération de crossover avec une particule choisie au hasard. Cette étape permet à la particule d'explorer l'espace de recherche en utilisant la position d'une autre particule comme référence.

En ajustant les valeurs de c1 et c2, nous pouvons contrôler l'exploration de l'espace de recherche et la convergence vers la meilleure solution. L'algorithme s'arrête lorsque l'un des critères d'arrêt est atteint, comme atteindre un nombre maximal d'itérations ou trouver une solution satisfaisante.

L'utilisation de la croisement permet aux particules de se déplacer vers de meilleures solu-

tions en combinant leur position actuelle avec d'autres positions dans l'essaim. Les valeurs de c1 et c2 déterminent l'équilibre entre l'exploration et l'exploitation, car les particules sont influencées soit davantage par la meilleure position globale ou personnelle, soit par d'autres particules dans l'essaim.

Dans l'ensemble, cette approche du PSO est une variation valide de l'algorithme et peut avoir ses propres avantages et inconvénients par rapport à l'algorithme PSO standard qui utilise des mises à jour de vélocité.

Pseudo code :

Initialiser la population de particules avec des positions et des vitesses aléatoires
Initialiser la meilleure solution personnelle pour chaque particule
Initialiser la meilleure solution globale
for $i = 1$ to $MaxGen$ **do**
  for all particules dans la population **do**
    Générer un nombre aléatoire entre 0 et 1
    if nombre aléatoire $<$ c1 **then**
      Appliquer le crossover avec la meilleure solution globale
    else if c1 $\leq$ nombre aléatoire $<$ c2 **then**
      Appliquer le crossover avec la meilleure solution personnelle
    else
      Appliquer le crossover avec une particule sélectionnée au hasard
    end if
    Mettre à jour la meilleure solution personnelle de la particule si nécessaire
    Mettre à jour la meilleure solution globale si nécessaire
  end for
end for
retourner La meilleure solution globale

Troisième partie

Expérimentation et analyse des résultats

0.7 Expérimentation pour le paramétrage

Pour cette raison, dans notre expérience, on a essayé de réduire le nombre de combinaisons en choisissant :

Pour le GA

- Tailles de population 50, 100
- Max generations 1000, 10000, 100000
- Taux de croisement 0.1, 0.3, 0.5
- Taux de mutation 0.1, 0.3, 0.5
- Tailles de problème 8, 14
- Nombre de test = 5

Pour le PSO

- tailles de population 50, 100
- Max generations 1000, 10000, 100000
- C1 0.1, 0.3, 0.5
- C2 [c1+0.2 .. 1.0]
- Tailles de problème 8, 14
- Nombre de test = 5

0.7.1 Résultats de quelques tests effectués :

Pour le GA

Taille population	Max Generations	Taux de Croisement	Taux de Mutation	Moyenne Temps d'exécution (ms)	Taux de Succès	Moyenne fitness
50	1 000	0.1	0.1	81.3	20%	2.0
100	1 000	0.1	0.3	168.0	60%	0.8
50	1 000	0.3	0.5	65.7	70%	0.6
50	10 000	0.5	0.1	994.7	70%	0.6
50	**10 000**	**0.3**	**0.3**	**964.8**	**100%**	**0.0**
100	10 000	0.5	0.5	2376.1	100%	0.0
50	100 000	0.1	0.5	9003.9	100%	0.0
50	100 000	0.3	0.3	8394.8	100%	0.0
100	100 000	0.5	0.1	18520.4	100%	0.0
100	**100 000**	**0.3**	**0.5**	**12520.4**	**100%**	**0.0**

FIGURE 3 – Résultats des tests effectués sur le GA

On remarque que lorsque on a une population d'une 50 élément , 10 000 générations ,taux de croisement 0.3 et un taux de mutation 0.3 c'est la meilleure combinaison **Pour le PSO**

Taille population	Max Generations	c1	c2	Moyenne Temps d'exécution (ms)	Taux de Succès	Moyenne fitness
50	1 000	0.1	0.3	47.1	70%	0.6
100	1 000	0.1	0.5	128	90%	0.2
50	**1 000**	**0.1**	**0.9**	**22.2**	**100%**	**0.0**
50	10 000	0.3	0.5	430.7	70%	0.6
50	10 000	0.3	0.9	276.9	100%	0.0
100	10 000	0.5	0.7	829.1	100%	0.0
50	100 000	0.1	0.7	4147.2	100%	0.0
50	100 000	0.3	0.9	3181.5	100%	0.0
100	100 000	0.1	0.9	6276.4	100%	0.0
100	**100 000**	**0.3**	**0.9**	**6197.9**	**100%**	**0.0**

FIGURE 4 – Résultats des tests effectués sur le PSO

On remarque que lorsque on a une population d'une 50 eléments , 1 000 générations , C1= 0.1 et C2= 0.9 c'est la meilleure combinaison.

Mais il ne faut pas oublier que les test ont été effectuées sur des tailles de problèmes qui ne dépassent pas 14 , pour cela si on veut explorer des tailles plus grandes, il faut mieux prendre la meilleure combinaison là où on a MaxGen = 100 000 et une taille de population de 100.

Les combinaisons qui vont être utilisés sont :

. **Pour le GA :**

Taille de population	Max Générations	Taux de Croisement	Taux de Mutation	Taille du problème
100	100 000	0.3	0.5	10

TABLE 1 – La combinaisons utilisée pour le GA

Pour cette combinaison , on assure une exploration suffisante (Max générations et taux de Mutation) d'une la population diversifié (taille population et taux de crossover).

. **Pour le PSO :**

Taille de population	Max Générations	C1	C2	Taille du problème
100	100 000	0.3	0.9	10

TABLE 2 – La combinaisons utilisée pour le PSO

On remarque que cette combinaison donne une importance au meilleur résultat personnel (PBest).

0.8 Expérimentation avec variation des paramètres :

Pour le GA :

Taille population	Max Generations	Taux de Croisement	Taux de Mutation	Moyenne Temps d'exécution (s)	Taux de Succès	Moyenne fitness
50	100 000	0.3	0.5	6.1	100%	0.0
100	100 000	0.3	0.5	10.406	100%	0.0
150	100 000	0.3	0.5	17.7	100%	0.0
200	100 000	0.3	0.5	22.974	100%	0.0
	1 000	0.3	0.5	0.157	40%	1.2
	10 000	0.3	0.5	1.115	80%	0.4
	100 000	0.3	0.5	12.053	100%	0.0
		0.1	0.5	8.144	80%	0.4
		0.2	0.5	9.769	100%	0.0
		0.3	0.5	11.520	100%	0.0
		0.4	0.5	14.734	100%	0.0
		0.5	0.5	16.518	100%	0.0
			0.1	9.966	100%	0.0
			0.2	10.444	100%	0.0
			0.3	10.505	100%	0.0
			0.4	10.566	100%	0.0
			0.5	17.712	100%	0.0

FIGURE 5 – Représentation tabulaire de la variation des paramètres pour l'algorithme GA

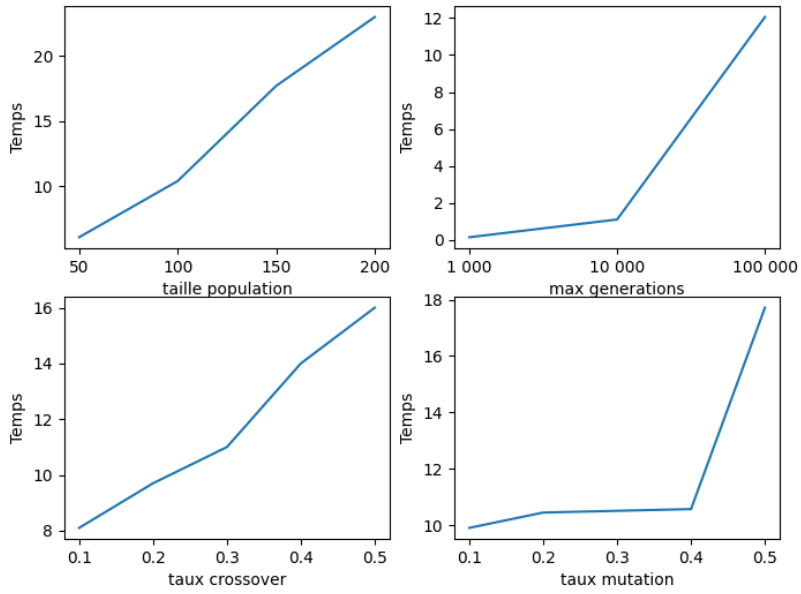


FIGURE 6 – Représentation graphique de la variation des paramètres pour l'algorithme GA

Pour le PSO :

Taille population	Max Generations	C1	C2	Moyenne Temps d'exécution (s)	Taux de Succès	Moyenne fitness
50	100 000	0.3	0.9	1.975	100%	0.0
100	100 000	0.3	0.9	4.242	100%	0.0
150	100 000	0.3	0.9	7.196	100%	0.0
200	100 000	0.3	0.9	8.531	100%	0.0
	1 000	0.3	0.9	0.06	80%	0.4
	10 000	0.3	0.9	0.480	100%	0.0
	100 000	0.3	0.9	4.185	100%	0.0
		0.1	0.9	4.225	100%	0.0
		0.2	0.9	4.487	100%	0.0
		0.3	0.9	3.963	100%	0.0
		0.4	0.9	3.758	100%	0.0
		0.5	0.9	4.145	100%	0.0
			0.4	7.331	100%	0.0
			0.5	6.534	100%	0.0
			0.6	6.553	100%	0.0
			0.7	5.579	100%	0.0
			0.8	4.793	100%	0.0

FIGURE 7 – Représentation tabulaire de la variation des paramètres pour l'algorithme PSO

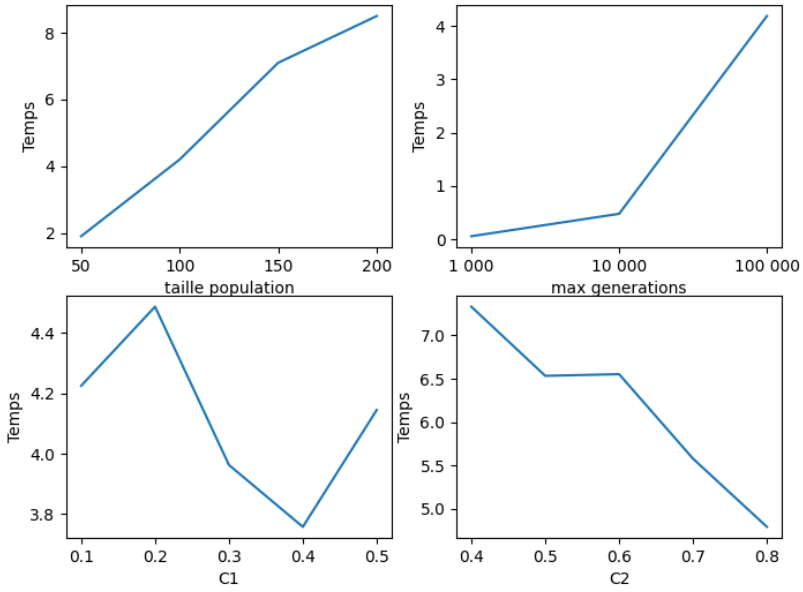


FIGURE 8 – Représentation graphique de la variation des paramètres pour l'algorithme PSO

0.8.1 Discussion des résultats :

Une taille de population plus grande augmente généralement la diversité de la population et les chances de trouver de meilleures solutions, mais augmente également le coût computationnel. Une plus petite taille de population peut conduire à une convergence plus rapide mais avec un risque plus élevé de rester bloqué dans des optimum locaux.

Un nombre maximum de générations plus élevé peut permettre à l'algorithme de continuer à chercher une meilleure solution pendant une période plus longue, mais augmente également le risque de surapprentissage et de gaspillage de ressources computationnelles. Un nombre maximum de générations plus petit peut conduire à une convergence plus rapide mais avec un risque de ne pas explorer suffisamment l'espace de recherche.

Un taux de crossover plus élevé peut augmenter l'exploration de l'espace de recherche et les chances de trouver de meilleures solutions, mais peut également conduire à une convergence prématurée et à une perte de diversité. Un taux de crossover plus petit peut conduire à une convergence plus lente mais avec une meilleure préservation de la diversité.

Un taux de mutation plus élevé peut augmenter les chances d'explorer de nouvelles solutions et d'éviter une convergence prématurée, mais peut également conduire à une perte de bonnes solutions et à une diminution de la qualité de la population. Un taux de mutation plus petit peut conduire à une population plus stable mais avec une exploration plus faible de l'espace de recherche.

Par conséquent, les effets de ces valeurs de paramètres sur les performances de l'algorithme peuvent dépendre des caractéristiques du problème, telles que la complexité du paysage de fitness, la taille de l'espace de recherche et la présence de contraintes ou d'objectifs multiples. Il peut être nécessaire d'expérimenter avec différentes valeurs de paramètres et d'analyser les résultats pour déterminer la configuration optimale pour le problème spécifique.

0.9 Expérimentation sur la taille du problème :

Après avoir effectué des tests sur différents paramètres on va choisir une combinaison et cette fois-ci on va faire varier la taille du problème.

Les combinaisons qui vont être utilisés sont :

. **Pour le GA :**

Taille de population	Max Générations	Taux de Croisement	Taux de Mutation
50	1 000	0.3	0.3

TABLE 3 – La combinaisons utilisée pour le GA

. **Pour le PSO :**

Taille de population	Max Générations	C1	C2
50	1 000	0.1	0.9

TABLE 4 – La combinaisons utilisée pour le PSO

0.9.1 Représentation tabulaire des résultats :

Pour le GA :

Taille du problème	Moyenne Temps d'exécution (ms)	Taux de Succès	Moyenne fitness
4	37.5	100%	0.0
6	31.7	100%	1.2
8	51.0	50%	1.0
10	52.0	40%	1.2
12	70.9	20%	1.6
14	90.3	20%	1.6
16	117.0	50%	1.2
18	173.3	20%	1.6
20	177.5	30%	1.6

FIGURE 9 – Représentation tabulaire de la variation des paramètres pour l'algorithme GA

Pour le PSO :

Taille du problème	Moyenne Temps d'exécution (ms)	Taux de Succès	Moyenne fitness
4	13.2	100%	0.0
6	13.8	100%	0.0
8	22.6	100%	0.0
10	25.2	100%	0.0
12	26.8	100%	0.0
14	38.0	0%	2.0
16	54.4	0%	2.4
18	56.8	0%	2.8
20	69.4	0%	5.6

FIGURE 10 – Représentation tabulaire de la variation des paramètres pour l'algorithme PSO

0.9.2 Comparaison entre GA et PSO :

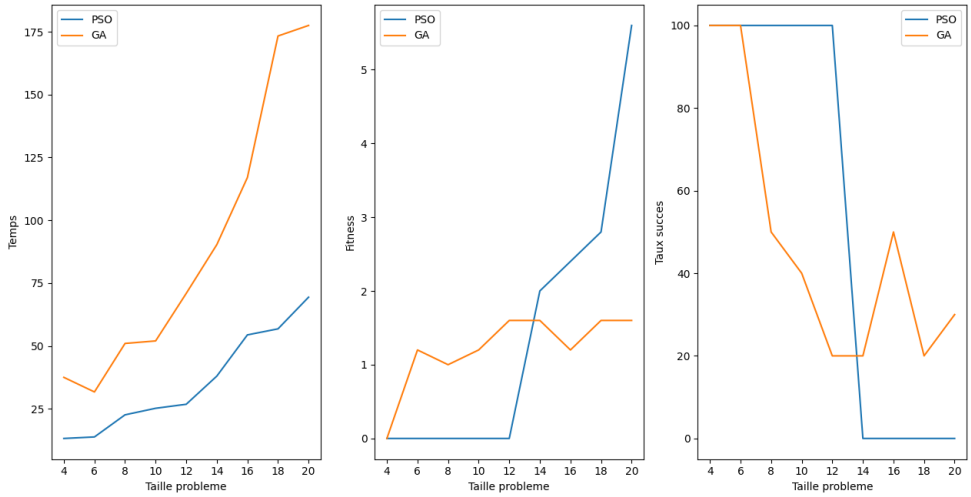


FIGURE 11 – Représentation graphique de la Performances de GA et PSO par rapport a la taille du problème

Analyse :

Le temps requis pour que GA et PSO trouvent la solution optimale ou quasi-optimale dépend de plusieurs facteurs tels que la taille de l'espace de recherche, la complexité de la fonction de fitness, le nombre d'itérations et le matériel utilisé. En général, PSO est plus rapide que GA, car il nécessite moins d'évaluations de la fonction de fitness et à une règle de mise à jour plus simple.

En termes de fitness, GA donne généralement des solutions de meilleure qualité.

Pour le taux de réussite : Dans certains cas, GA peut être plus efficace pour trouver l'optimum global, tandis que dans d'autres, PSO peut être plus performant.

Dans l'ensemble, le choix entre GA et PSO dépend du problème spécifique et des ressources informatiques disponibles. Les deux algorithmes ont leurs forces et leurs faiblesses, et le choix optimal ne peut être déterminé que par l'expérimentation et l'analyse des résultats.

0.10 Comparaison avec les heuristiques :

- Dans la 1ere partie du projet on a travailler sur les heuristiques,et en revenant au résultats on peut constater les points suivants :
  - Les heuristiques sont des techniques de résolution de problèmes qui utilisent une approche pratique pour trouver une solution qui peut ne pas être optimale mais qui est acceptable. Les métaheuristiques, en revanche, sont une classe d'heuristiques qui utilisent des stratégies de plus haut niveau pour guider la recherche d'une solution.

- L'algorithme génétique (GA) et l'optimisation par essaim de particules (PSO) sont tous deux des exemples de métaheuristiques couramment utilisés pour résoudre des problèmes d'optimisation. Alors que le GA est basé sur les principes de la sélection naturelle et de l'évolution, le PSO est basé sur le mouvement des particules dans un essaim.
- Les heuristiques conviennent souvent mieux pour résoudre des problèmes d'optimisation simples avec un petit espace de recherche, tandis que les métaheuristiques comme GA et PSO conviennent mieux pour résoudre des problèmes d'optimisation complexes avec un plus grand espace de recherche.
- Dans nos expériences sur le problème des N-Reines, nous avons constaté que les deux GA et PSO étaient capables de trouver de bonnes solutions, le PSO étant généralement meilleur en termes de vitesse de convergence et de qualité de solution. Cependant, le GA était capable de trouver une gamme plus large de solutions et pourrait être plus adapté pour des problèmes avec un plus grand espace de recherche.
- Un inconvénient des métaheuristiques est qu'elles ne garantissent pas toujours de trouver la solution optimale, tandis que les heuristiques donne la solution optimale mais ils sont généralement plus coûteuses en termes de calcul que les métaheuristiques, en particulier pour les problèmes avec un grand espace de recherche.
- En conclusion, les heuristiques et les métaheuristiques comme GA et PSO ont leurs avantages et leurs inconvénients, et le choix de l'un ou l'autre dépendra du problème à résoudre. Pour des problèmes d'optimisation simples avec un petit espace de recherche, les heuristiques peuvent être suffisantes, tandis que pour des problèmes d'optimisation complexes avec un plus grand espace de recherche, les métaheuristiques comme GA et PSO peuvent être plus adaptées.

Quatrième partie

Environnement expérimental.

Caractéristiques de la machine	Système d'exploitation	CPU	RAM	Version compilateur java
BENKOUITEN Aymen	Système d'exploitation 64 bits, processeur x64. Windows 10 Professionnel.	Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz 2.50 GHz	8,00 GO	java-version "1.8.0_291"
KENAI Imad Eddine	Système d'exploitation 64 bits, processeur x64. Windows 10 Professionnel.	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 2.00 GHz	16,00 GO	java-version "1.8.0_291"

Table 5 – Environnement expérimental

Conclusion Générale

Dans ce projet, nous avons exploré l'utilisation de deux métaheuristiques populaires, GA et PSO, pour résoudre le problème des N-Reines. Nous avons comparé leurs performances en termes de qualité de solution et d'efficacité computationnelle, et les avons également comparées à des heuristiques simples.

Nos expériences montrent que les deux GA et PSO ont été capables de trouver de bonnes solutions au problème, avec PSO étant généralement plus performant que PSO en termes de vitesse de convergence et GA plus performant que PSO en termes de qualité de solution. Cependant, des heuristiques simples ont également été en mesure de trouver de bonnes solutions, bien que la convergence ait été plus lente.

Dans l'ensemble, notre projet met en évidence l'efficacité des métaheuristiques telles que GA et PSO pour résoudre des problèmes complexes d'optimisation tels que le problème des N-Reines. Cependant, le choix de l'algorithme dépend des caractéristiques du problème et du compromis entre la qualité de la solution et l'efficacité computationnelle. Dans les cas où l'espace de problème est petit ou que la contrainte de temps n'est pas critique, des heuristiques simples peuvent également être efficaces.

Les travaux futurs pourraient se concentrer sur l'optimisation supplémentaire des paramètres de GA et PSO pour améliorer leurs performances, ainsi que sur l'exploration d'autres algorithmes métaheuristiques tels que BAT et BSO et la comparaison de leurs performances à celles présentées dans ce projet.

Bibliographie

[1] An Overview of Particle Swarm Optimization Variants. MuhammadImranRathiahAuthors. PublishedbyElsevierLtd.

[2] Genes-chrom. https://towardsdatascience.com/.