Cette version de l'algorithme, proposée dans [?] utilise une fonction à dynamique chaotique et la distribution vol de Lévy afin de garantir une convergence rapide. Une phase de mutation est aussi rajoutée à la fin de chaque itération

0.1 Fonction à dynamique chaotique

Soit f:I $\rightarrow$ I une fonction continue.On suppose que la dynamique associée est chaotique.Alors:

1. L'ensemble des points périodiques de f est partout dense dans I
2. f est sensible aux conditions initiales, ceci signifie que s'il y'a un petit changement dans la condition initiale x_0,le changement correspondant de $x_t = f_t(x_0)$ croit avec la croissance de t

Ce sont des fonctions qui permettent d'avoir une suite de nombres aléatoires qui dépend d'une condition initiale.Il existe plusieurs types de fonctions chaotique :fonction logistique,fonction de tchebychev.Après plusieurs tests, il a été remarqué qu'avec les fonctions logistiques nous obtenons une convergence plus rapide de la fonction objective .Le minimum de la fonction objective a été atteint en 5 itérations comme le montre le graphe[?] Du graphe , on peut

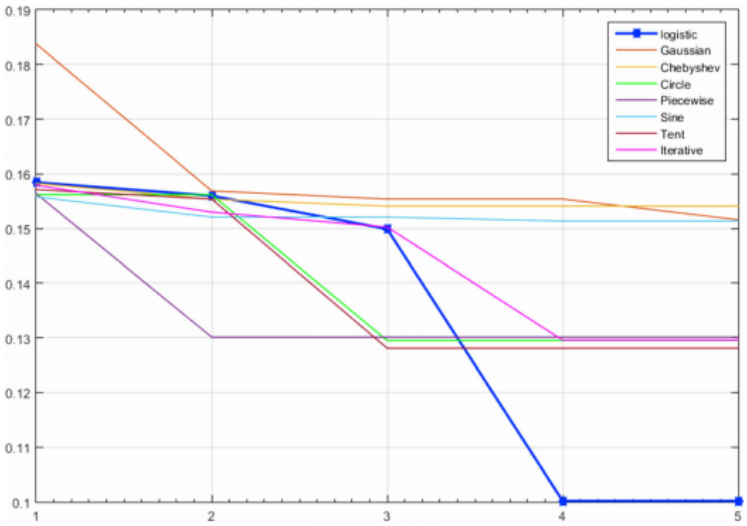

Figure 1: convergence fonctions chaotiques

voir qu'avec la fonction logistique la fonction objective atteint le minimumu à partir de la 5eme itération

0.1.1 Fonction logistique

La fonction logistique est définie par la formule suivante :

$$x_{n+1} = ax_n(1 - x_n), x \in [0, 1], 0 < a \leq 4$$

où a est une constante caractéristique de la fonction logistique que nous avons fixé à la valeur give the value après plusieurs simulations La fonction logistique est utilisée dans l'algorithme pour générer la valeur p

0.2 La distribution vol de Lévy

le vol de levy est un modèle mathématique caractérisé par une moyenne et une variance infinies ce qui rend le mouvement plus lent permettant ainsi une meilleure exploration de l'espace de recherche [**?**]. Dans ILWOA , la variable C est remplacé par un pas aléatoire de la marche aleatoire levy donné par les formules suivantes:

$$Levy \rightsquigarrow \frac{\lambda\Gamma(\lambda)\sin(\pi\lambda/2)}{\pi}\frac{1}{s^{1+\lambda}}$$
$$s = \frac{U}{|V^{\lambda-1}|}, \quad U \rightsquigarrow N(0, \sigma_u^2), V \rightsquigarrow N(0, 1)$$
$$\sigma_u^2 = [\frac{\Gamma(\lambda+1)}{\Gamma((\lambda+1)/2)}\frac{\sin(\pi\lambda/2)}{2^{(\lambda-1)/2}}]^{1/\lambda}$$

0.3 Phase de mutation

Une phase de mutation a lieu à la fin de chaque itération. On vérifie d'abord si la solution atteinte est optimale .dans ce cas la recherche s'arrete sinon on applique la mutation sur cette dernière. Cette phase est composée de 3 opérations exécutées séquentiellement comme le montre la figure ***figure 2***:

- Permutation: deux items de la solution sont choisis aléatoirement pour effectuer une permutation entre eux
- Deplacement: Un sous ensemble d'items choisi aléatoirement est déplacé vers une position qui est egalement choisie aléatoirement

- Inversion: On applique une inversion sur un sous ensemble d'items

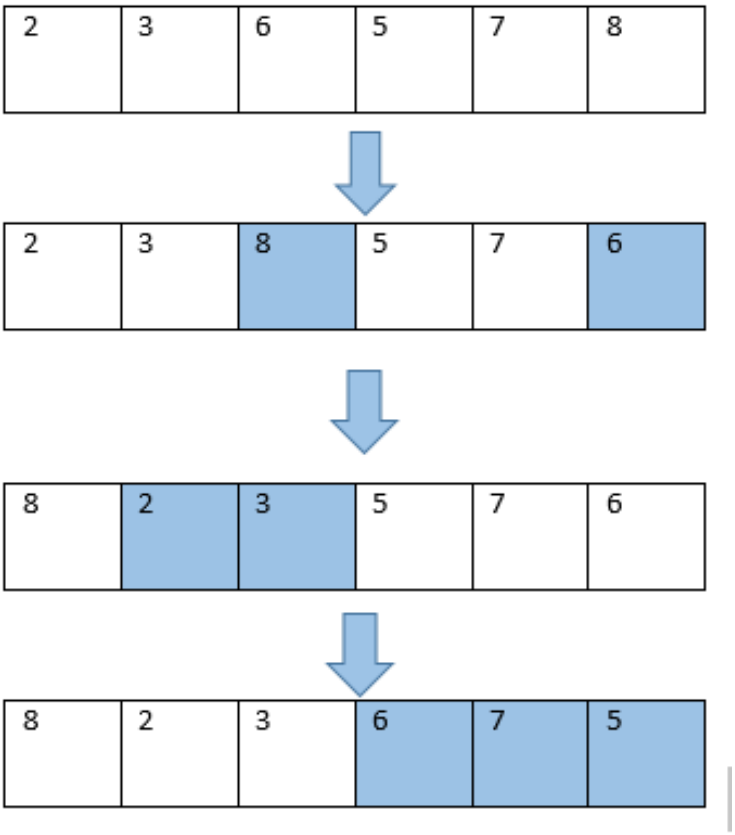

Figure 2: Phase de mutation

0.4 Pseudocode

Algorithm 1 Improved Whale Optimization Algorithm

Initialiser la population de baleines (l'ensemble initial des solutions candidates): Xi (i = 1, 2 ,..., N)
Évaluer les solutions de la population initiale
X* = la meilleure solution actuelle
while $t < max\_iter$ **do**
  for solution ∈ population **do**
    Mettre à jour a, A, C avec le vol de levy, l et p avec la fonction logistique
    if $r < 0,5$ **then**
      if $|A| < 1$ **then**
        Mettre à jour la solution par Eq.(1)
      else
        Sélectionner une solution aléatoire X_r
        Mettre à jour la solution par Eq.(2)
      end if
    else
      Mettre à jour la solution par l'Eq.(3)
    end if
  end for
  Vérifier si une solution dans la population dépasse l'espace de recherche et la modifier
  Appliquer la discretisation par LOV
  Évaluer la nouvelle solution
  Mettre à jour X* s'il existe une meilleure solution Sinon lancer la phase de mutation
  Evaluer la nouvelle solution,màj de la meilleure solution
  t = t + 1
end while

0.5 Test et résultats

Pour montrer les performances de cet algorithme nous avons effectué une série de Test sur des instances du Benchmark «Scholl »composé de trois classes d'une difficulté qui varie d'une classe à une autre. En ce qui concerne les

paramètres de l'algorithme, nous avons utilisé deux méthodes de calibrage : Un calibrage manuel et un calibrage automatique avec IRace; Le tableau dans la ***figure* 1** indique les valeurs obtenues par chacune des deux méthodes.

Table 1: Configuration paramétrique de l'algorithme

	a	**b**	**beta**	**max_iter**	**nb_whales**
Calibrage manuel	8	0.99	1.5	391	13
Calibrage automatique	4	1.5	1	30(Hard),10(Classe 1 et 2)	10

Nous allons dans l'analyse qui suit:

- comparer entre les résultats obtenus par les deux méthodes de calibrage ainsi montrer l'influence du choix des paramètres sur la performance de l'algorithme.
- Analyser les performances de l'algorithme et le comparer à l'algorithme WOA exécuté avec la configuration optimale suivante : $nb\_whales = 28$, $max\_iter = 271$, $b = 7.64$, $a = 20$.

0.5.1 Comparaison entre les résultats obtenus par calibrage automatique et les résultats obtenus par calibrage manuel:

Afin d'effectuer cela, nous avons exécuté l'algorithme avec les paramètres obtenus par calibrage automatique et les paramètres obtenus par calibrage automatique, sur les instances difficiles de la classe 3. Les tableaux dans la ***figure* 3** montrent que le calibrage automatique améliore la qualité de la solution ainsi que le temps d'exécution

0.5.2 Analyse des performances de l'algorithme:

0.5.2.1 Analyse du temps d'exécution: A partir du graphes ***figure* 4** , on peut déduire que :

- Le temps d'exécution de ILWOA varie de 1 à quelques dizaines de secondes pour les grandes instances
- L'algorithme ILWOA est plus lent que WOA , on justifie ceci par le fait qu'en plus des instructions exécutées par le WOA , l'ILWOA comporte une phase de mutation et une initialisation par une heuristique

	Résultats d'exécution sur des instances de la classe 3 avec les paramètres de calibrage manuel									
Instance	HARD0	HARD1	HARD2	HARD3	HARD4	HARD5	HARD6	HARD7	HARD8	HARD9
Solution	64	62	60	61	60	59	60	59	60	60
Temps d'exécution	25.88	25.40	24.63	25.98	24.53	25.50	25.67	25.45	24.01	25.87

	Résultats d'exécution sur des instances de la classe 3 avec les paramètres obtenus par calibrage automatique									
Instance	HARD0	HARD1	HARD2	HARD3	HARD 4	HARD 5	HARD 6	HARD 7	HARD 8	HARD9
Solution	59	60	60	59	60	59	59	58	59	59
Temps d'exécution	5.68	7.48	5.60	4.93	4.54	4.58	4.41	4.34	4.34	4.46

Figure 3: Résultats d'exécution des instances de la classe 3 avec calibrage manuel vs. calibrage automatique.

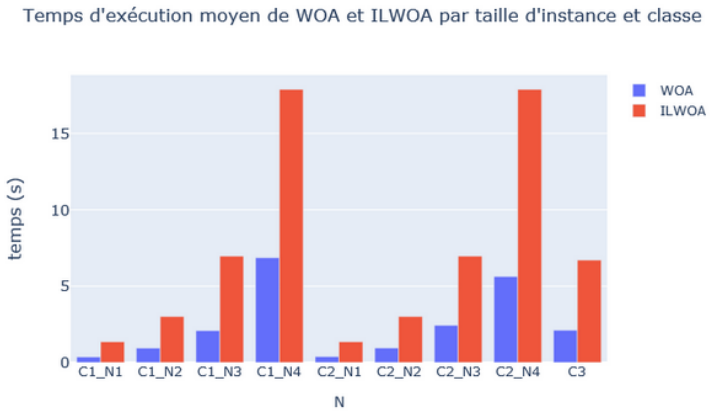

Figure 4: temps d'execution (en s) de ILWOA et WOA

0.5.2.2 Analyse de la qualité de solution: La qualité de solution a été mesurée par le ration dont on a cité la formule précédemment.A partir du graphes ci-dessous , on peut déduire que :

- Le Ratio obtenu par l'ILWOA est proche de 1 pour toutes les instances même pour les instance de la classe 3 ceci signifie qu'il donne de bons résultats quelque soit la difficulté de l'instance
- la solution obtenue avec l'algorithme ILWOA est très proche de la solution optimale voire même égale pour des instances des classes 1 et 2
- ILWOA améliore la qualité de solution par rapport à l'algorithme woa

pour les instances des 3 classes confondues

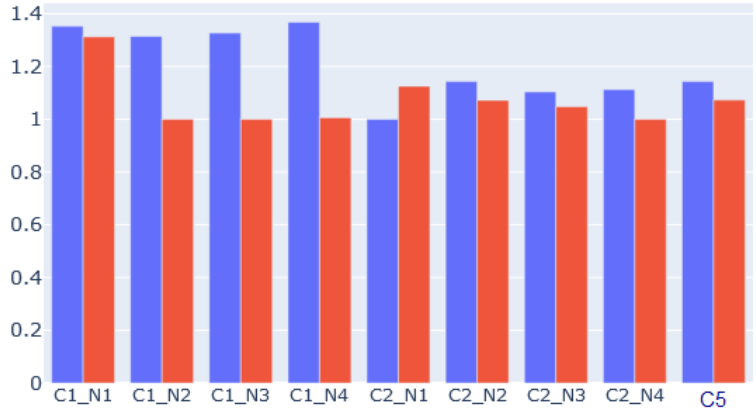

Figure 5: ratio de ILWOA et WOA

0.5.3 Interprétation

- L'Algorithme ILWOA permet d'obtenir une bonne qualité de solution en un temps raisonnable
- Le caractère stochastique de l'algorithme WOA permet de mieux explorer l'espace de recherche, l'utilisation des fonctions logistiques et de la distribution du vol de lévy ont amélioré cela
- En analysant le nombre de search agents et d'itérations ainsi que la solution obtenue , On peut conclure que l'ajout d'une phase de mutation a permis d'améliorer la rapidité de convergence de l'algorithme
- Les améliorations effectuées ont permis d'augmenter la performance de l'algorithme
- La performance de l'algorithme dépend fortement des valeurs de ses paramètres d'où la nécessité du calibrage qui permet de trouver la meilleure configuration pour l'algorithme
- La valeur des paramètres peut dépendre de la nature de l'instance

- Le calibrage automatique est important du fait que ça permet de nous épargner une tâche tant fastidieuse, tout en assurant une meilleure qualité du résultat