Un algorithme génétique est une métaheuristique évolutionnaire à l'origine proposée par Holland [1]. Depuis lors, il est devenu une méthode puissante pour résoudre de nombreux problèmes d'optimisation combinatoire difficiles, dont la liste peut être trouvée dans de nombreuses références, voir par exemple [2]. Les étapes générales de l'algorithme peuvent être décrites comme suit:

0.1 Pseudocode

Algorithm 1 Algorithme génétique

```
Générer une population initiale
while le critère d'arret n'est pas encore rencontré do
  Choisir des paires pour la reproduction
  Effectuer des croisements pour générer des descendants
  Faire des fluctuations dans les nouveaux descendants (mutation)
  Evaluer l'aptitude de nouveaux descendants
  Générer une nouvelle population
end while
```

Un élément clé de l'AG est la génération d'une population dont les composants sont appelés *chromosomes*, un terme emprunté à la génétique. Un *chromosome* est en fait une représentation codée d'une solution; dont chaque composant est appelé *gène*

0.2 Représentation chromosomique

On représente une solution du Bin packing comme suit :

- On a **n** articles à ranger donc on utilisera n boîtes au pire des cas (un objet par boite).
- On suppose que chaque boîte est composée de **n** cellules, où chaque cellule ne peut contenir qu'un seul article.
- Chaque cellule à un numéro unique dans la solution.
- Si la cellule de l'ordre **i** de la boite **j** est remplie par un objet, alors on aura plus le droit de ranger un objet dans toutes les cellules de l'ordre **i** des autres boites.
- La cellule **zero** contient le nombre de boîtes utilisées dans cette solution.

Pour décrire cette représentation, considérons l'exemple suivant. Supposons que nous ayons quatre objets (n=4) de poids 2 , 2 , 4 et 4 respectivement, affectés à trois boîtes (c0=3) comme indiqué ci-dessous :

Contenu	Cellules	Boites
3	C0	/
2	C1	1
2	C2	
0	C3	
0	C4	
0	C5	2
0	C6	
4	C7	
0	C8	
0	C9	3
0	C10	
0	C11	
4	C12	

Avec c0 : contient le nombre de boîtes utilisées.
Le chromosome correspondant peut être défini comme suit:

3	2	2	0	0	0	0	4	0	0	0	0	4

Cependant, avec cette représentation,les chromosomes ont une longueur égale à $1 + n^2$ (un entier pour chaque cellule), alors pour les grands n, nous aurons besoin d'une grande quantité de mémoire pour les stocker. Pour cela, nous utiliserons une représentation plus compacte qui utilise des chromosomes de longueur $n + 1$ au lieu de $1 + n^2$. Cette dernière consiste à garder trace que des numéros des cellules remplies (ou utilisées).
Dans l'exemple ci-dessus, nous avons utilisé les cellules (c1,c2,c7,c12) et la cellule c0, nous aurions donc la séquence

3	1	2	7	12

qui a une longueur de $n + 1$, avec $n = 4$.

Dans la première cellule, cellule 0, on stocke le nombre de boîtes utilisées qui est de 3 boites. Les numéros suivants indiquent les numéros de cellules dans lesquels les objets sont placés.

Récupérer la solution depuis un chromosome:

1. Le nombre de boîtes utilisées est obtenu à partir de la cellule 0.
2. Pour obtenir dans quelle boîte chaque article a été affecté, on suit les étapes suivantes:

- On pose c le numéro de la cellule et n le nombre d'articles
- On calcule : a= c mod (n) et b=[c/n] où c = bn + a
- Si a=0 (c est divisible par n), alors:
  - Le numéro de l'article est **sa position dans le chromosome** et le numéro de la boîte où il est rangé est **b**.

  Sinon:
  - Le numéro de l'article est **sa position dans le chromosome** et le numéro de la boîte où il est rangé est **1 + b**.

Dans l'exemple ci-dessus, pour la cellule 7, nous avons: $7 = 1 \times 4 + 3$, le reste est a= 3 et b = 1, donc b + 1 = 2, donc l'objet 3 est placé dans la boîte 2. Pour la cellule 12, nous avons: $12 = 3 \times 4 + 0$, le reste est a= 0, et b = 3, donc l'objet 4 est placé dans la boîte 3

0.2.1 Initialisation

La génération de la population initiale en :

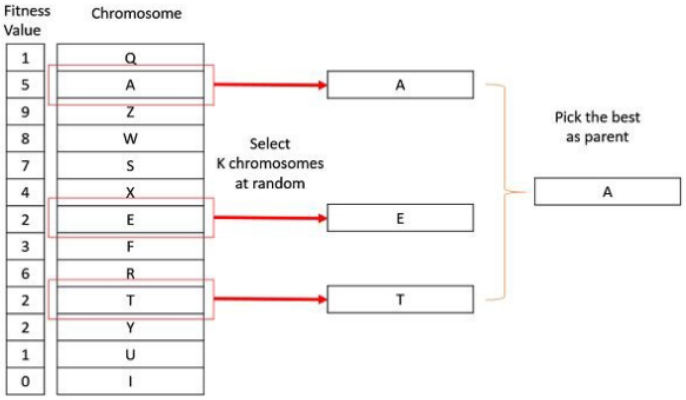
- Changeant l'ordre des objets aléatoirement.
- Génération du chromosome avec **First Fit**

0.2.2 Selection (Choix d'une paire pour la reproduction)

C'est la phase qui consiste à choisir les individus à reproduire et ceux qui participent à la construction d'une nouvelle population. Nous avons opté pour la méthode du Tournament Selection.

0.2.3 Tournament Selection

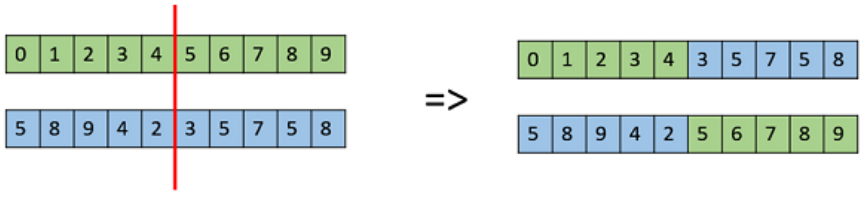
Dans cette approche, un "tournoi, k" est organisé entre k chromosomes (solutions)choisis au hasard dans la population et celui avec la meilleure forme physique (Fitness Value) est sélectionné comme gagnant et sera autorisé à se reproduire.
Supposons que k = 3 :



La pression de sélection(Une *pression de sélection* est une contrainte environnementale qui va **pousser** une espèce à évoluer dans une direction donnée.) peut être facilement ajustée en changeant la taille du tournoi (un k plus élevé augmente la pression de sélection). La sélection du tournoi est indépendante de la fonction Fitness.Parmis ces avantages, on cite la diminution le temps de calcul, et son fonctionnement sur les architectures parallèles.

0.2.4 Croisement

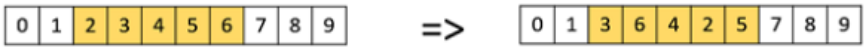
L'opérateur de croisement est analogue à la reproduction et au croisement biologique. Dans ce cas, plus d'un parent est sélectionné et un ou plusieurs descendants sont produits en utilisant le matériel génétique des parents. Le crossover est généralement appliqué dans une AG à forte probabilité *pc*
L'opérateur choisi pour cet algorithme est le 'One Point Crossover' ou 'le croisement à un point'. Dans ce croisement, un point de croisement aléatoire est sélectionné et les queues de ses deux parents sont échangées pour obtenir de nouveaux descendants, ce qui revient à diviser l'ensemble des boîtes en deux fragments et échanger les objets de ces derniers.
Par exemple dans la figure suivante, nous avons le point de croisement =5:



0.2.5 Mutation

La mutation peut être définie comme un petit ajustement aléatoire dans le chromosome, pour obtenir une nouvelle solution. Il est utilisé pour maintenir et

introduire la diversité dans la population génétique et est généralement appliqué avec une faible probabilité *pm*. Si la probabilité est très élevée, l'AG se réduit à une recherche aléatoire.
La mutation est la partie de l'AG qui est liée à «**l'exploration**» de l'espace de recherche. Il a été observé que la mutation est essentielle à la convergence de l'AG alors que le croisement ne l'est pas.
L'opérateur de mutation choisi pour cette algorithme est : "**scramble mutation**" ou "**la mutation de brouillage**", qui, à partir du chromosome entier, choisit aléatoirement un sous-ensemble de gènes dont le cardinale est un paramètre en entrée de la fonction, et ses valeurs sont brouillées ou mélangées au hasard.



0.2.6 Evaluation

Le choix du chromosome ayant la plus petite valeur de la cellule 0, c'est à dire le plus petit nombre de boîtes après avoir vérifié la validité de la solution (chromosome) i.e si le chromosome constitue réellement une solution acceptable.

0.2.7 Fitness et validation d'un chromosome

Cette fonction nous permet de vérifier si et combien un chromosome donné est "bon" ou "fit" à notre problème.
Pour cela il faut d'abord vérifier la validité de la solution: d'un côté vérifier si tous les articles ont effectivement été rangés dans des boîtes, et de l'autre, vérifier si le total des poids des articles dans chaque boîte est effectivement inférieur ou égal à la capacité d'une boîte.
Après avoire effectué ces vérifications on pourra prendre comme valeur de "*fitness*" du chromosome, le nombre de boîtes utilisées, stocké dans la 1ère cellule du chromosome.

0.2.8 Critère d'arrêt

L'algorithme s'arrête après un nombre d'itérations prédéfini, donné en paramètre.

L'Algorithme AG sera à son tour exécuté plusieurs fois, selon un paramètre en entrée (on a opté pour 10 fois), et à la fin la solution ayant le plus petit nombre de boîtes utilisées sera prise comme solution de l'instance.

0.3 Tests et Résultats

Après avoir finalisé notre code, on a commencé la phase des tests qui a été divisée en deux partie :

1. La partie des tests préliminaires.
2. La partie des tests après le calibrage des paramètres.

On a utilisé pour nos tests comme instances du problème BinPacking, comme pour les méthodes précédentes (Exactes et Heuristiques), les instances du benchmark Scholl. Les paramètres sur lesquels on a joué, sont :

- **nbrGen** : le nombre de populations générées.
- **k** : le paramètre de l'algorithme de sélection "*le k-tournament*"
- **popSize** : la taille d'une population

La probabilité de croisement a été fixée à 0.85, et la probabilité de mutation a été fixée à son tour à 0.1.

0.3.1 Les tests préliminaires

Dans cette partie, on a voulu tester les performances de notre programme , en essayant plusieurs combinaisons des paramètres:

- **nbrGen** : on a essayé 4, 50, 100, 200 et puis 500 qui lui, ne donnait pas de meilleurs résultats que ceux obtenus avec 200 itérations.
- **k** : on a essayé 4 et 25.
- **popSize** : on a essayé avec une population de 4 chromosomes, 8, et 10.

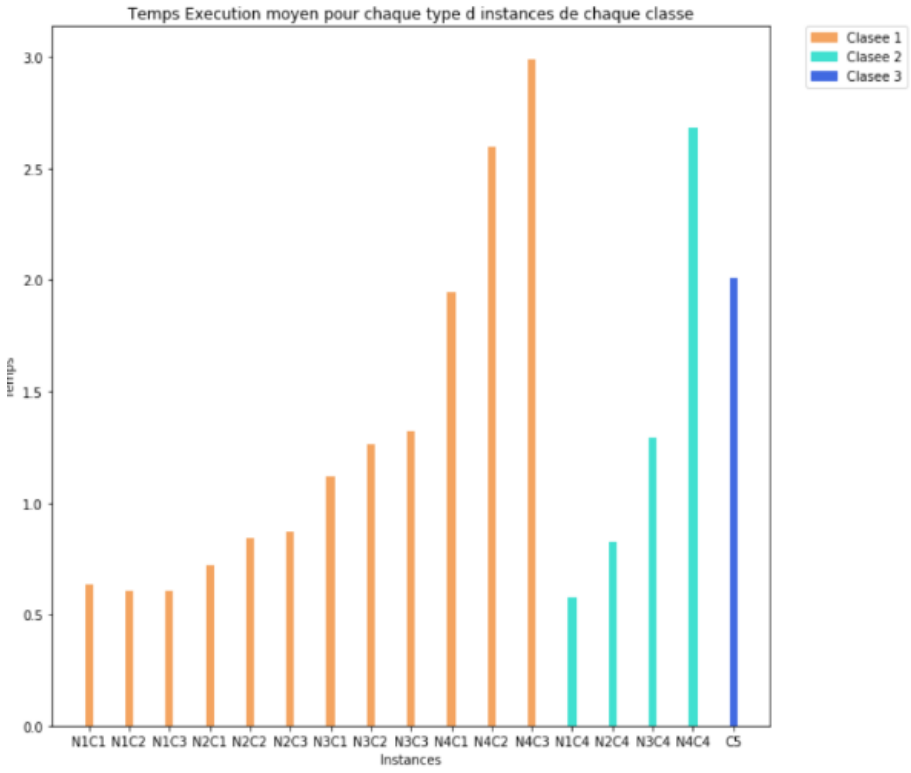
Au final, notre choix s'est porté sur le triplet :(**nbrGen**=200, **k**=25,**popSize**=10) car ils donnaient d'assez bons résultats.

0.3.2 Les tests après calibrage des paramètres

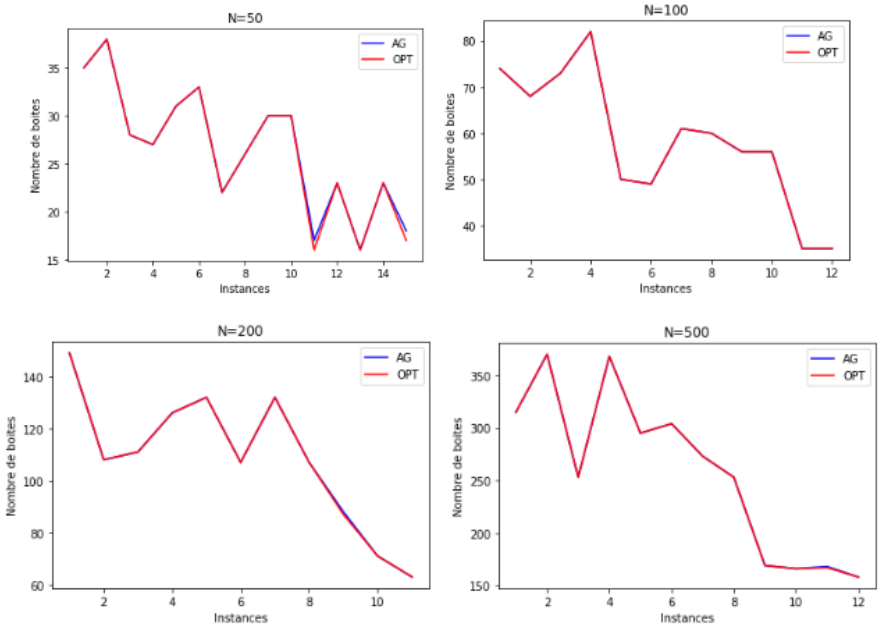
Pour pouvoir confirmer notre choix de paramètres on a vu nécessaire d'utiliser un outil de calibrage, on a utilisé l'outil **IRACE**.
Après avoir exécuté le code avec **IRACE**, on a obtenu cette combinaison des paramètres (**nbrGen**=200,**k**=20,**popSize**=10) en fixant , **probaCroisement**=0.85 et **probaMutation**=0.1, et qui est,en effet, la combinaison qui donne les meilleurs résultats dans le cas de notre programme. Ceci confirme nos résultats pour le choix des paramètres trouvés pendant les tests préliminaires.

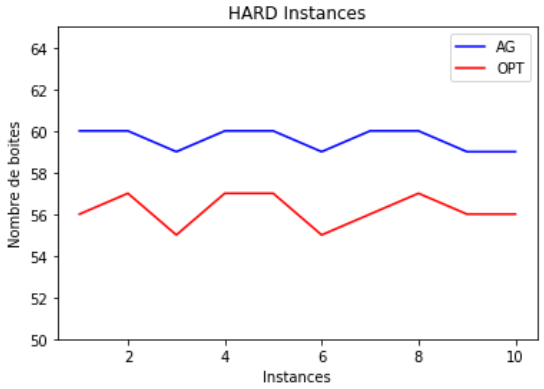
Dans ce qui suit on présentera les résultats d'exécution de notre algorithme génétique en prenant comme paramètres ceux trouvés par l'outils **IRACE** (**nbrGen**=200,**k**=20,**popSize**=10) avec **probaCroisement**=0.85 et **probaMutation**=0.1

Voici les résultats des exécutions, pour chaque classe:
Graphe 01: Graphe représentant le temps d'exécution moyen de chaque type d'instance (la moyenne d'exécution de 5 instances de même type) des 3 classes de Scholl



Graphe 02: Les graphes suivants montrent la différence entre le nombre de boîtes utilisées par AG et la valeur optimale OPT

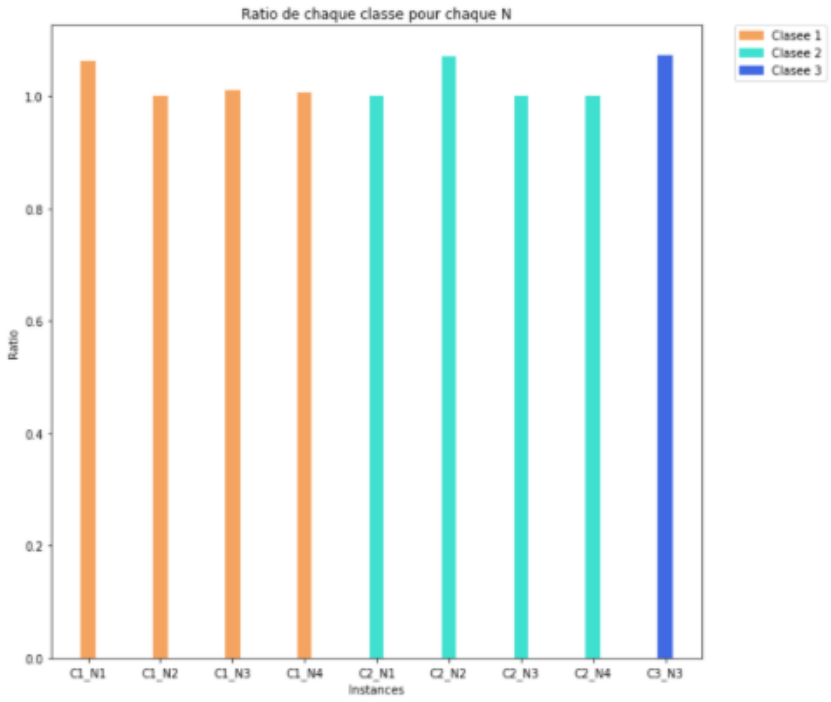




Voici les ratio obtenus par famille d'instances :

Instances	Ratio AG
C1_N1	1.0625
C1_N2	1.0
C1_N3	1.01149
C1_N4	1.00598
C2_N1	1.0
C2_N2	1.07142
C2_N3	1.0
C2_N4	1.0
C3_N3	1.07272

Graphe 03: Et voici le graphe des ratio pour chaque nombre d'objets N des instances



Analyse et Interprétation des résultats :

- Le temps d'exécution correspond à dix exécutions du AG, ceci était nécessaire vu la nature stochastique de l'algorithme, la solution initiale générée aléatoirement (le trie aléatoir de la liste des objets avant d'effectuer FF) et les facteurs liés aux méthodes de croisement et de mutation, influent sur les résultats obtenus.
- En terme de qualité de solution (étudiée en utilisant le ratio), l'AG donne d'assez bons résultats pour les instances du benchmark scholl, surtout dans le cas des classes 1 et 2 , où il arrive souvent à trouver la solution optimale sinon une valeur très proche de cette dernière, mais avec une qualité moindre dans le cas de la 3ème classe. **[Graphe 03]**
- Le Ratio obtenu par l'AG est proche de 1 (Worst case ratio$<$1.1 dans tous les cas). Ceci signifie que l'algorithme génétique donne une bonne qualité de résultats même dans le pire des cas (les instances les plus difficiles).**[Graphe 03]**