Compte Rendu

Projet final

Algorithmes multi-objectifs : MOPSO et Micro-GA

Métaheuristique en Optimisation

8INF852 – Groupe 1

Esmé James - JAME15539504
Wilfried Pouchous - POUW04069501

Sommaire

I - Algorithmes choisis

1. MOPSO

Coello et Pulido proposent dans leur article « *MOPSO : A Proposal for Multiple Objective Particle Swarm Optimization* » une adaptation de l'algorithme génétique PSO pour les problèmes multiobjectifs. L'algorithme utilise le principe de la domination de Pareto pour déterminer la direction des particules et utilise une mémoire externe pour enregistrer les individus non dominés.

Pour comprendre MOPSO, il faut d'abord comprendre le fonctionnement de PSO. L'algorithme PSO fonctionne avec une population (appelée swarm) constituée de particules. Les particules sont déplacées dans l'espace de recherche selon leur meilleure position connue, leur vélocité et les meilleures positions du swarm. Au fur et à mesure du temps et à travers l'exploration des bonnes positions connues, les particules convergent.

Le fonctionnement de MOSPO se déroule de la façon suivante :

1) Initialisation de la population **Po**
2) Stockage des particules non-dominées dans l'archive **Rep**
3) Générer des grilles qui divisent l'espace de recherche (hypercubes) et localiser les particules dans la grille en utilisant leur valeur objective comme coordonnées
4) **Boucle jusqu'à Gmax**
    a) Calculer la vitesse de chaque particule
    b) Mettre à jour la nouvelle position des particules en fonction de leur vitesse
    c) Vérifier que les particules sont toujours dans l'espace de recherche
    d) Évaluer les particules
    e) Garder les nouvelles meilleurs particules non-dominées dans Rep
    f) Mettre à jour la grille
    g) Si Rep est plein, enlever des particules

Les étapes les plus complexes seront décrites de façon plus détaillés dans les parties suivantes.

1.1 Initialisation de la population

Pour l'initialisation, une population Po de taille N est créée en assignant une valeur aléatoire entre les bornes du problème correspondant à la position de chaque particule. Ensuite, la vitesse de chaque particule est initialisée à 0 et la valeur objective de chaque particule est calculée. Enfin, on initialise la mémoire de chaque particule. Ceci se fait en stockant la position et la valeur objective de chaque particule dans la structure "Best" de la particule.

1.2 Génération des hypercubes

La génération des hypercubes consiste en la création d'une grille permettant de diviser l'espace de recherche. Les limites de la grille sont construites à partir des valeurs min et max des valeurs objectives (ou coûts) de la population.

Les bornes de la grille sont créés de la façon suivante : on récupère les valeurs objectives min et max de la population auxquelles on soustrait (pour min) ou ajoute (pour max) la différence entre ces coûts multiplié par un paramètre alpha.

Pour chaque objectif du problème, on crée un vecteur ligne de taille 8 entre les coûts min et max. La borne inférieure de la grille ira de $-\infty$ à ce vecteur ligne et la borne supérieure ira de ce vecteur ligne à $+\infty$.

Il faut ensuite trouver dans quel endroit de la grille se trouve chaque particule. Pour cela, on compare le coût de chaque particule aux valeurs présentes dans le vecteur correspondant à la borne supérieure de la grille. Si le coût est inférieur, on garde l'index de la valeur du vecteur borne supérieure. Ces deux valeurs correspondent au SubIndex. On calcule ensuite l'index de la particule avec le SubIndex.

1.3 Vitesse et position des particules

Le calcul de la vitesse de chaque particule se fait selon la formule suivante :

$$V[i] = W \times V[i] + R1 \times (PBest[i] - Po[i]) + R2 \times (RepH - Po[i])$$

W est le poids inertiel valant 0.4, il permet d'empêcher les particules de se déplacer dans la même direction que précédemment. V correspond à la vitesse précédente de la particule (0 à la première génération). R1 et R2 sont des nombres aléatoires. Po est la

position courante de la particule et PBest correspond à la meilleure position qu'ai eu cette particule. RepH est une valeur choisie dans la mémoire externe, correspondant au "leader", c'est-à-dire à la particule que suivront les autres. Elle est choisie en effectuant une Roulette Wheel Selection sur les cellules de la grille contenant des particules.

1.4 Suppression de particules si Rep est plein

Si la mémoire externe Rep est pleine, on supprime les particules en surplus. Pour choisir la particule à supprimer, on effectue une Roulette Wheel Selection sur les cellules de la grille contenant des particules.

2. Micro-GA

Coello et Pulido proposent dans leur article *« A Micro-Genetic Algorithm for Multiobjective Optimization »* une adaptation de l'algorithme génétique Micro-GA pour les problèmes multiobjectifs. C'est un algorithme qui utilise une très petite population (souvent de l'ordre de 4 ou 5), un processus de réinitialisation et trois formes d'élitisme.

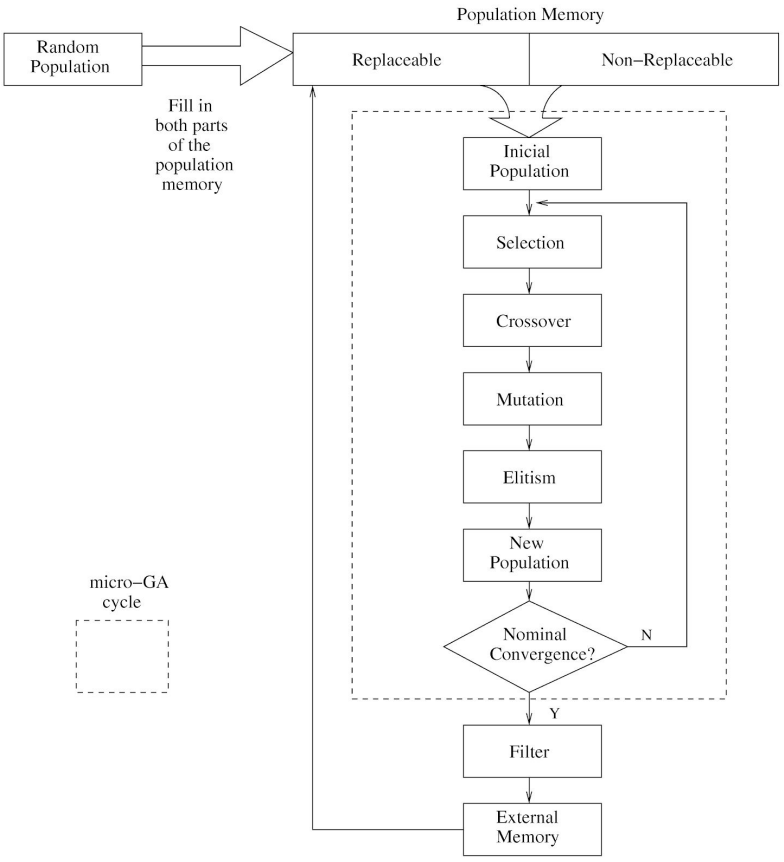
Micro-Ga utilise deux mémoires : la **mémoire population** et la **mémoire externe** (ou l'archive). La mémoire population est divisée en deux parties : une partie **remplaçable** et une partie **non-remplaçable** (qui ne change jamais tout au long du déroulement de l'algorithme).

Le fonctionnement de Micro-GA, illustré par la figure ci-dessous, se déroule de la façon suivante :

1) Génération d'une population aléatoire qui remplit la mémoire population $\boldsymbol{M}$ (formé des parties remplaçable $\boldsymbol{Mr}$ et non-remplaçable $\boldsymbol{Mn}$)
2) **Début de la boucle de Micro-GA (jusqu'à Gmax)**
    a) Création de la population initiale $\boldsymbol{Pi}$ à partir de Mr et Mn
    b) **Boucle jusqu'à la convergence nominale**
        (1) Opérations de sélection, croisement et mutation
        (2) Application de l'élitisme : Un individu non-dominé de la population est choisi pour aller à la génération suivante.
    c) Sélection de deux individus non-dominées de Pi (le premier et le dernier)
    d) Comparaison des deux individus avec la mémoire externe $\boldsymbol{E}$ et ajout s'ils sont toujours non-dominés (si E est rempli, remplacer deux individus de

E qui sont entourés par beaucoup de points par les deux nouveaux individus)

e) Comparaison des deux individus avec deux individus aléatoires de la partie remplaçable (*Mr*) de la mémoire population *M* et remplacement s'ils sont toujours non-dominés
f) Application du 3ème type d'élitisme : on remplace les membres de la population remplaçable par des membres de la mémoire externe E à intervalle régulier (cycle de remplacement)



Les étapes les plus complexes seront décrites de façon plus détaillés dans les parties suivantes.

1.1 Initialisation des populations

La première étape de l'algorithme Micro-GA est de générer une population aléatoire afin de remplir la mémoire population. Une fois les individus générés, on les répartit entre la partie remplaçable et la partie non-remplaçable en fonction du paramétrage utilisateur concernant la répartition (0.3 de non-remplaçable dans notre étude).

Une fois cela fait, on va rentrer dans une boucle qui va répéter N fois l'algorithme Micro-GA (N étant égal à Gmax). A chaque boucle, l'algorithme ne travaille pas avec l'entièreté des individus mais seulement une petite partie. Il faut donc initialiser une petite population Micro-GA en se servant dans les parties remplaçables et non-remplaçables de la population mémoire (toujours avec le même pourcentage de 0.3).

1.2 Les différents opérateurs

Nous allons appliqué différents opérateurs sur la population Micro-GA et cela plusieurs fois afin d'arriver à une convergence nominale (état où tous les individus sont identiques ou très proches). Parmi ces opérateurs, on retrouve :

- Un opérateur de sélection : Sélection par tournoi binaire
- Un opérateur de croisement : Simulated Binary Crossover
- Un opérateur de mutation : Uniform Mutation

Nous avons opté pour un croisement différent que celui proposé par l'article car nous avions des problèmes à passer les valeurs en binaire.

1.3 Élitisme - partie 1

En plus des opérateurs cités ci-dessus, nous effectuons à chaque fois de l'élitisme sur la population Micro-GA. Cela consiste à prendre aléatoirement un individu parmi ceux non-dominés et à l'ajouter en plus à la population Micro-GA.

1.4 Individus non-dominés et comparaisons

Une fois la convergence nominale atteinte, l'algorithme choisit deux individus non-dominés de la population Micro-GA. Puis, on en crée deux copies : une pour comparer à la mémoire externe E et une pour comparer à la population remplaçable.

Il s'agit d'un même fichier pour ces deux comparaisons : *NotDominated*. Le fichier permet de comparer si des individus sont non-dominés par rapport à une population donnée. S'ils sont dominés on les supprime, donc on retourne uniquement ceux qui sont non-dominés.

En ce qui concerne la comparaison avec la mémoire externe E, on vérifie d'abord si E n'est pas vide (cas de la première boucle). Si E est vide, alors on ajoute les deux individus non-dominés venant de la population Micro-GA. Si E n'est pas vide, alors on compare les individus non-dominés avec chaque individu de E (via le fichier *NotDominated*). S'il reste au moins un des deux individus non-dominés, alors deux cas possibles :

- E n'a pas atteint sa taille maximum, alors on peut les ajouter

- E a atteint sa taille maximum alors on utilise la *CrowdingDistance* pour repérer les individus étant le plus près de ces voisins dans E et les remplacer par les individus non-dominés

En ce qui concerne la comparaison avec la population remplaçable, on sélectionne deux individus aléatoirement de celle-ci et on compare nos deux individus non-dominés aux individus venant d'être sélectionnés. Si l'individu non-dominé l'est toujours, alors on s'en sert pour remplacer l'individu sélectionné dans la population remplaçable (là encore, il s'agit d'élitisme).

1.5 Élitisme - partie 2

La dernière partie de cet algorithme consiste en une autre étape d'élitisme. A un intervalle régulier de la valeur *ReplacementCycle*, nous effectuons :

- Une vérification de la non-dominance de tous les individus de E entre eux et suppression de ceux dominés (parasites).

- Remplacement de la population remplaçable par des membres de E.

II - Résultats

Nous testons les deux algorithmes en affichant le front de pareto sur un graphique mettant en lien les deux objectifs du problème. Pour chaque graphe, il est indiqué (en rouge) le nombre optimal de générations afin de reconnaître l'allure de la courbe attendue.

Pour la vérification des graphiques, nous avons comparé nos résultats à ceux du wiki suivant : https://en.wikipedia.org/wiki/Test_functions_for_optimization.

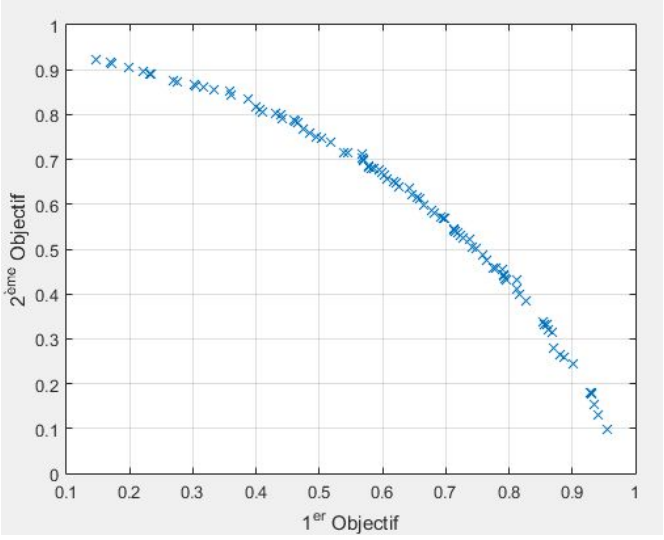
Nous avons fait ces tests avec les 6 fonctions de références suivantes :

- Fonseca et Flemming (FON)
- Poloni (POL)
- Kursawe (KUR)
- Zitzler–Deb–Thiele (ZDT1, ZDT2 et ZDT3)

1. MOPSO

Ci-dessous les paramétrages et les résultats correspondants trouvés avec l'algorithme MOPSO selon les fonctions de références.

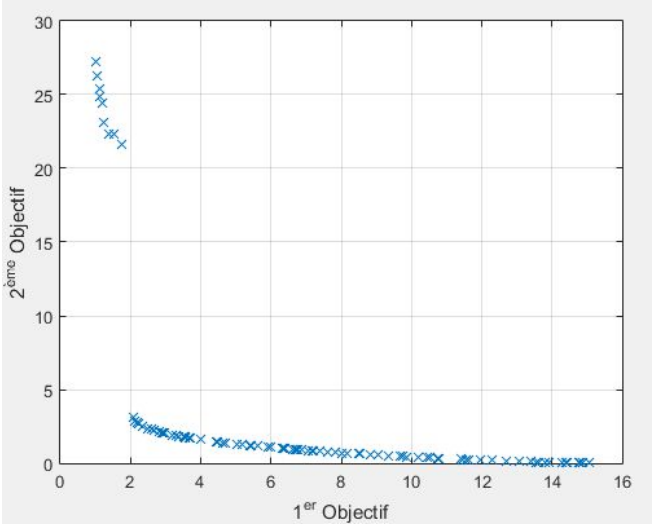
- **Fonction FON :**



Paramétrage :

Taille de population (Po)	*200*
Taille mémoire externe (Rep)	*100*
Gmax	*100*

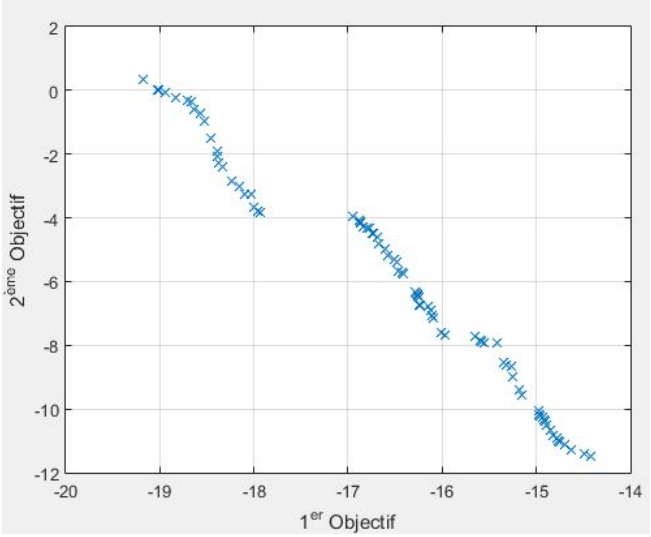
- **Fonction POL :**



Paramétrage :

Taille de population (Po)	*200*
Taille mémoire externe (Rep)	*100*
Gmax	*50*

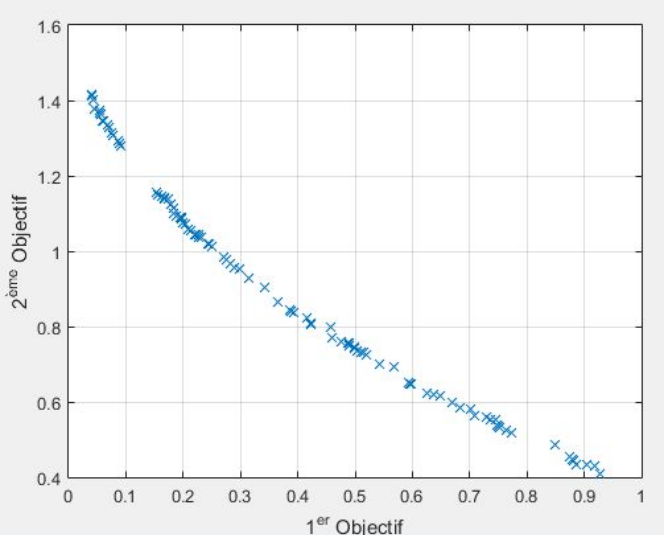
- **Fonction KUR :**



Paramétrage :

Taille de population (Po)	*200*
Taille mémoire externe (Rep)	*100*
Gmax	*200*

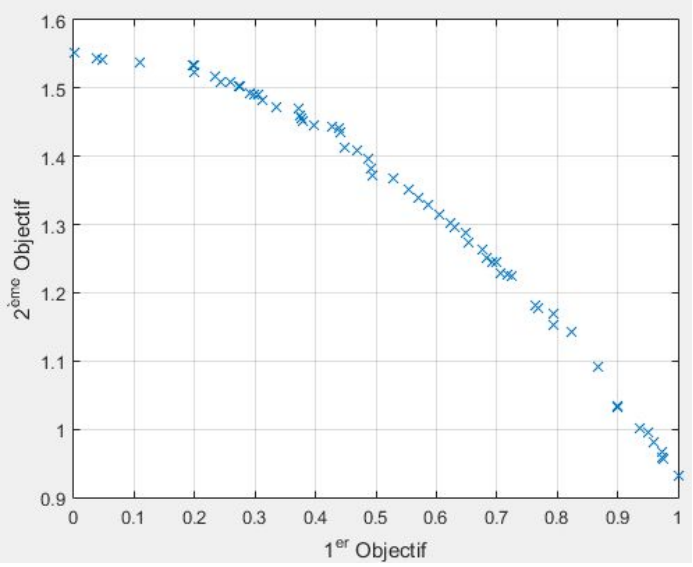
- **Fonction ZDT1 :**



Paramétrage :

Taille de population (Po)	*200*
Taille mémoire externe (Rep)	*100*
Gmax	*150*

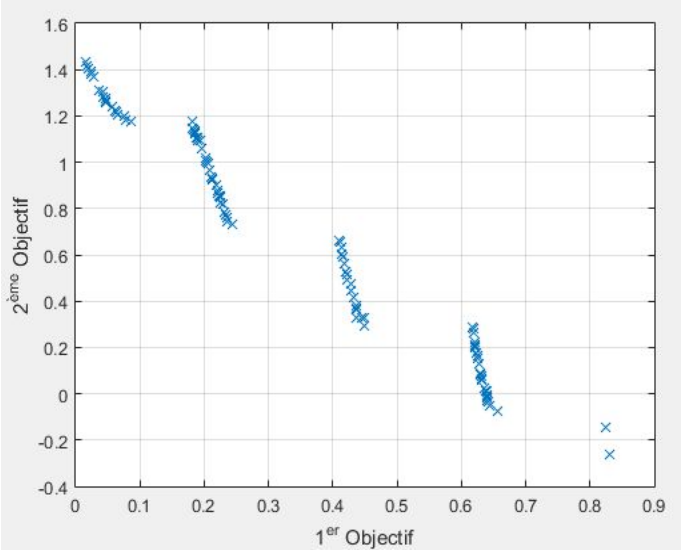
- **Fonction ZDT2 :**



Paramétrage :

Taille de population (Po)	*200*
Taille mémoire externe (Rep)	*100*
Gmax	*100*

- **Fonction ZDT3 :**



Paramétrage :

Taille de population (Po)	*200*
Taille mémoire externe (Rep)	*100*
Gmax	*100*

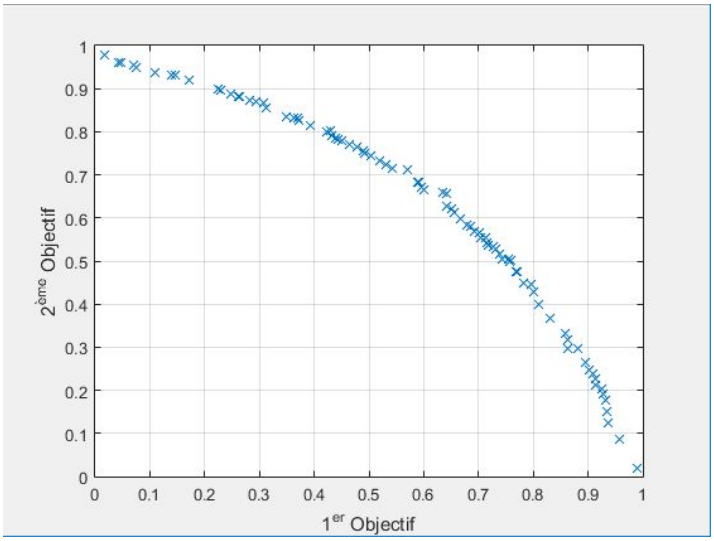
2. Micro-GA

Ci-dessous les paramétrages et les résultats correspondants trouvés avec l'algorithme Micro-GA selon les fonctions de références.

D'autres paramètres que ceux indiqués sont présents à chaque exécution, cependant nous ne les réécrivons pas afin de pas surcharger la visualisation des résultats. Ces valeurs sont celles de l'article pour le test 5 (KUR).

- ➔ Taille population de Micro-GA = 4
- ➔ Pourcentage de population non-remplaçable provenant de la population initiale = 0.3
- ➔ Itérations pour atteindre la convergence nominale = 2
- ➔ Intervalle de cycle de remplacement = 50
- ➔ Probabilité de croisement = 0.7
- ➔ Probabilité de mutation = 0.1

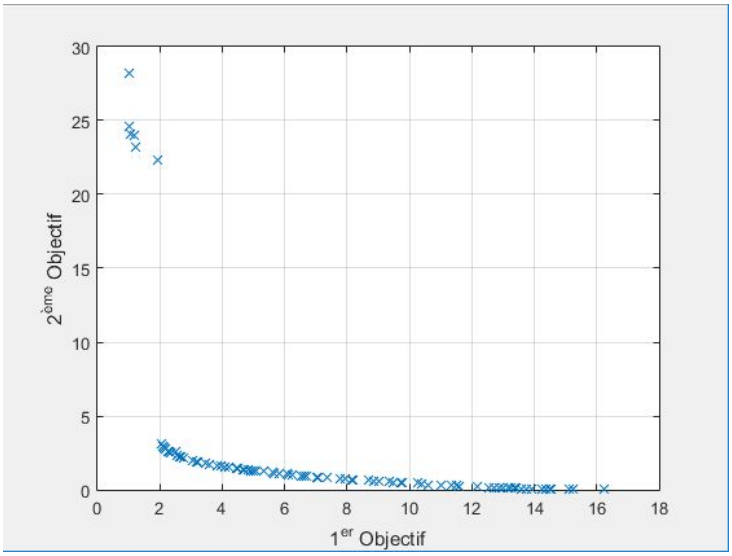
- **Fonction FON :**



Paramétrage :

Mémoire population (Pi)	*100*
Mémoire externe (E)	*50*
Gmax	*500*

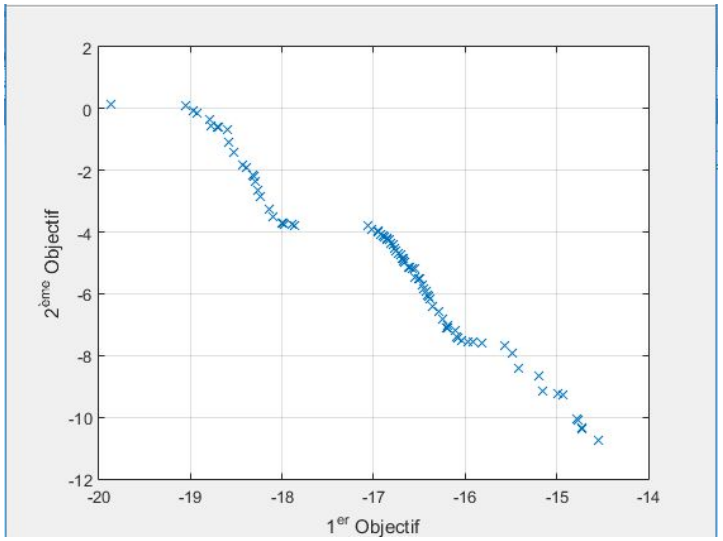
- **Fonction POL :**



Paramétrage :

Mémoire population (Pi)	*100*
Mémoire externe (E)	*50*
Gmax	*300*

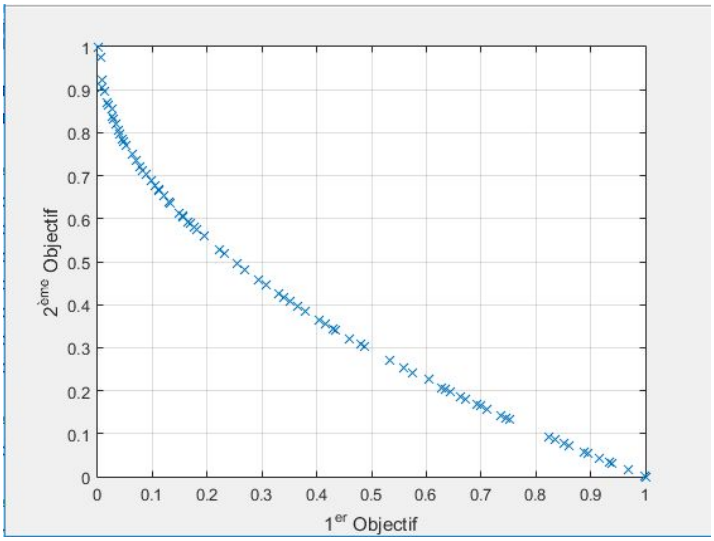
- **Fonction KUR :**



Paramétrage :

Mémoire population (Pi)	*100*
Mémoire externe (E)	*50*
Gmax	*1500*

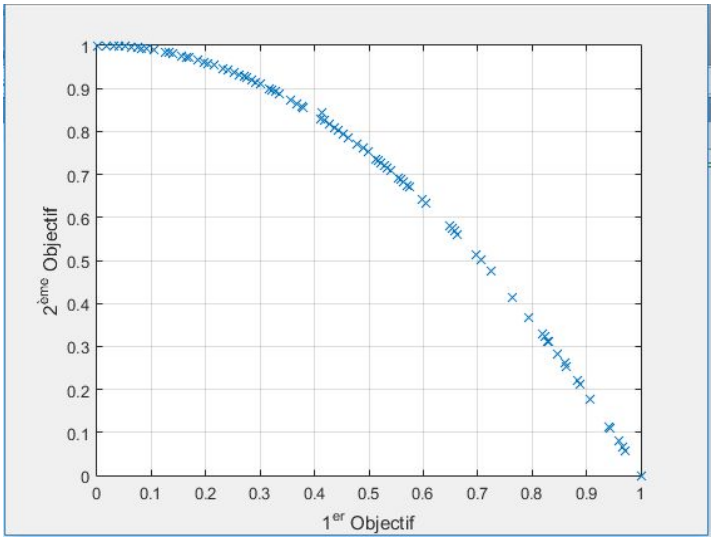
- **Fonction ZDT1 :**



Paramétrage :

Mémoire population (Pi)	*100*
Mémoire externe (E)	*50*
Gmax	*400*

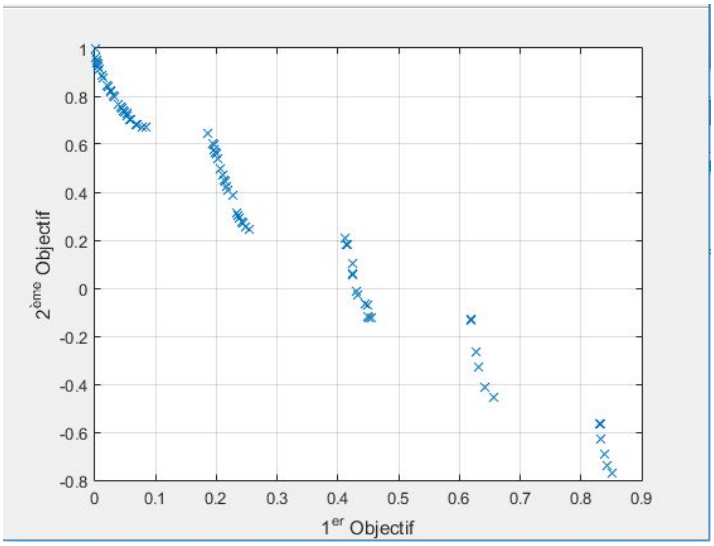
- **Fonction ZDT2 :**



Paramétrage :

Mémoire population (Pi)	*100*
Mémoire externe (E)	*50*
Gmax	*400*

- **Fonction ZDT3 :**



Paramétrage :

Mémoire population (Pi)	*100*
Mémoire externe (E)	*50*
Gmax	*800*

III - Comparaisons

Pour comparer ces deux algorithmes (MOPSO et Micro-GA), nous ne pouvons pas comptabiliser le nombre de générations optimales pour obtenir un graphe de la forme attendu car ces deux algorithmes reposent sur des principes différents. En effet, MOPSO est un algorithme évolutionnaire utilisant une grande population et donc n'ayant pas besoin de beaucoup de générations tandis que c'est le cas inverse pour Micro-GA (petite population et beaucoup de générations).

Ainsi, nous avons décidé de nous baser sur le temps en secondes à former un graphe ressemblant à la forme attendue pour comparer les 2 algorithmes. Ci-dessous un tableau récapitulatif des temps nécessaires pour chaque algorithme par rapport aux six fonctions de références.

Fonctions de référence	**MOPSO** (en secondes)	**Micro-GA** (en secondes)
FON	*10.56*	*10.80*
POL	*6.18*	*6.55*
KUR	*18.34*	*31.26*
ZDT1	*14.29*	*9.09*
ZDT2	*5.68*	*8.99*
ZDT3	*7.48*	*16.94*

D'après ce tableau, on remarque MOPSO est généralement plus rapide que Micro-GA.

De plus, les deux articles présentant chaque algorithme comparent les résultats à d'autres algorithmes multi-objectifs comme NSGA-II et PAES. Les algorithmes MOPSO et Micro-GA retournent tous deux des résultats dans un temps plus court que NSGA-II ou PAES.

IV - Conclusion

Les deux algorithmes étudiés sont donc plus efficaces que des algorithmes multi-objectifs comme NSGA-II ou PAES. On constate que MOPSO est légèrement plus rapide que Micro-GA, cependant ils restent tous deux des algorithmes très compétitifs. Enfin, en termes de complexité, Micro-GA nous a paru plus simple car il se base sur le principe des algorithmes génétiques.