FST
Faculté des Sciences et Techniques
UNIVERSITÉ HAUTE-ALSACE

Weighted Differential Algorithm

Réalisé par :

Sagna Boubou

Yassine Asmae

Benamara Insaf Salma

Vanulli Antoine

Encadré par :

M. Idoumghar Lhassane

Sommaire :

1. Introduction

Le projet consiste à bien comprendre le fonctionnement de l'algorithme WDEA, de le recoder en c++ tout en respectant le squelette des classes données.

Notre groupe doit utiliser les compétences acquises pour coder un algorithme génétique et les appliquer sur le WDEA. Ainsi que de l'utiliser sur les fonctions Benchmark donnés pour tester son fonctionnement et son efficacité.

Nous tâcherons de bien démontrer le fonctionnement du WDE ainsi que son utilité et ses avantages par rapport aux autres algorithmes du même type ou du même principe.

2. Weighted Differential Evolution Algorithm

Le WDE est un algorithme évolutif pondéré, qui fait partie des algorithmes évolutionnaires. Ce type d'algorithme et utilisés couramment pour résoudre des problèmes complexes d'optimisation numérique, autrement dit, multimodaux, non différentiel, hautement non linéaire et problème a modèle contraint. Les EAs ont une base de population, itératif, mécanismes de recherche stochastiques, recherchant des solutions optimales qui appartiennent au problème connexe.

L'algorithme WDE est un algorithme conçu afin de trouver des solutions plus exactes et moins encombrées. Le WDE est une version pondérée du DE, en effet c'est deux algorithmes sont assez similaires, mais aussi bien différents.

Le WDE est un algorithme basé sur 2 populations, itératif, à recherche évolutionnaire développé pour résoudre des problèmes d'optimisation numérique à valeurs réels.

Bien que la stratégie de génération de modèle d'essai de WDE soit très productive, il y a une possibilité de piégeage au niveau local solution aux problèmes hybrides. Chaque matrice de motifs de WDE évolue en un motif aléatoire et permuté matrice pour fournir l'essaimage à chaque itération. Alors que WDE génère un modèle d'essai en permettant le changement d'un individu pour la première matrice de motifs, il produit un essai modèle en autorisant un nombre d'individus sélectionnés au hasard pour la deuxième matrice de modèle. WDE génère un modèle d'essai en permettant le changement de tous les individus pour la troisième matrice de motifs.

WDE n'a théoriquement aucun paramètre de contrôle. Étant donné que les valeurs des paramètres utilisés dans WDE sont déterminées au hasard, WDE n'a besoin d'aucune opération de réglage des paramètres. Par conséquent, il est facile à utiliser.

Parmi les caractéristiques du WDE on cite :

- Le processus d'essaimage de WDE utilise un nouveau mécanisme aléatoire de contrôle des mutations.

- L’équation du système de WDE est en partie similaire à équation système de l'algorithme d'évolution différentielle mais la stratégie de génération de vecteurs de direction de WDE est différente.
- Les vecteurs de direction générés dans WDE sont composés des vecteurs mixtes de différents vecteurs de motifs.
- WDE utilise une nouvelle méthode de contrôle des limites.
- Les valeurs de tous les paramètres utilisés dans WDE sont déterminées au hasard. Par conséquent, WDE ne gaspille pas temps de réglage initial des paramètres.
- Puisqu'il a une structure non récursive, WDE peut être parallélisé facilement. Par conséquent, c'est plutôt rapide.

3. Pseudo code WDEA :

Input: Objective function:$\mathcal{F}$, Search Limits:(low, up), Size of Population:N,
Dimension of Problem: D, and Maximum Number of Iterations: $MaxCycle$
Output: $gmin$: Global minimum, $gbest$: Global minimizer

1 $i = [1, ..., \alpha, ..., N], i^* \in i,\ \ i0 = [1 : 2N],\ \ j0 = [1 : D]$, $where\ i, i0, j0 \in \mathbb{Z}^+$
2 $\kappa_{(\cdot)} \sim U(0,1), \kappa_{(\cdot)} \neq 0,\ \lambda_{(\cdot)} \sim N(0,1)$
// $\kappa_{(\cdot)}$, $\lambda_{(\cdot)}$ generates $(\cdot)$ sized random numbers at each call.
// INITIALIZATION
3 $P_{(i0,j0)} \sim \mathbf{U}(low_{(j0)}, up_{(j0)})$
4 $fitP_{(i0)} = \mathcal{F}(P_{(i0)})$
5 **for** *iteration=1 to MaxCycle* **do**
6 $j = permute(i0)$; // permute(·) function denotes the vector permutation function
7 $k = j_{(1:N)}$; // see line 9 for $P_{(k)}$
8 $l = j_{(N+1:2N)}$; // see line 15 for $P_{(l)}$
// GENERATION OF SUBPOPULATION; $SubP$ and $TempP$
9 $SubP = P_{(k)}$
10 $fitSubP = fitP_{(k)}$
11 **for** *index=1:N* **do**
12 $w^* = \kappa^3_{(N)} \mid [N, 1] = size(w)$
13 $w^* := \frac{w^*}{\sum w^*}$; // (:=) denotes 'update of variable value' operation
14 $w := w^* \times \Delta \mid \Delta = [\mathbf{1}] \mid [1, D] = size(\Delta)$
15 $TempP_{(index)} = \sum(w \circ P_{(l)})$; // ∘ denotes Hadamart operator
16 **end**
// GENERATION OF BINARY-MAP M
17 $M_{(1:N,1:D)} = 0$
18 **for** *index=1:N* **do**
19 **if** $\kappa_{(1)} < \kappa_{(1)}$ **then** $K = \kappa^3_{(1)}$ **else** $K = 1 - \kappa^3_{(1)}$
20 $J = V(1 : \lceil K \times D \rceil) \mid V = permute(j0)$; // ⌈⌉ denotes ceiling function
21 $M_{(index,J)} := 1$
22 **end**
// GENERATION OF EVOLUTIONARY STEP SIZE
// see lines 2,14 for λ, Δ, respectively.
23 **if** $\kappa_{(1)} < \kappa_{(1)}$ **then** $F = [\lambda^3_{(D)}]' \mid [1, D] = size(F)$ **else**
$F = (\lambda^3_{(N)} \times \Delta) \mid [N, D] = size(F)$ // TRIAL VECTOR GENERATION (MORPHOGENESIS: (Mutation & Random-Crossover)
24 $\Gamma = TempP - SubP_{(m)} \mid m = permute(i) | m \neq [1 : N]$; // see lines 9-16 for $TempP$ and $SubP$, respectively.
25 $T = SubP + F \times M \circ \Gamma$; // ∘ denotes Hadamart operator
// BOUNDARY CONTROL
26 **if** $T_{(i,j0)} < low_{(j0)}$ **then** $T_{(i,j0)} = low_{(j0)} + \kappa^3_{(1)}(up_{(j0)} - low_{(j0)})$
27 **if** $T_{(i,j0)} > up_{(j0)}$ **then** $T_{(i,j0)} = up_{(j0)} + \kappa^3_{(1)}(low_{(j0)} - up_{(j0)})$
// UPDATE
28 $fitT = \mathcal{F}(T)$
// For each $i^* \in i$ run line 29. See line 1 for i^*.
29 **if** $fitT_{(i^*)} < fitSubP_{(i^*)}$ **then** $[SubP_{(i^*)}, fitSubP_{(i^*)}] = [T_{(i^*)}, fitT_{(i^*)}]$
30 $[P_{(l)}, fitP_{(l)}] = [SubP, fitSubP]$; // see line 8 for l
// GET THE SOLUTIONS
31 $[gmin, gbest] = [fitP_{(\alpha)}, P_{(\alpha)}] \mid fitP_{(\alpha)} = \min(fitP) \mid \alpha \in i$; // see line 1 for α
32 **end**

Ce pseudo code présente la structure de l'algorithme ainsi que les quatre partis les plus importantes pour notre code, l'initialisation, la mutation, le croisement et la sélection. En prenant en compte que la fonction objectif est déjà définie et est implémenté au début.

4. La répartition des tâches

Nous avons partagé le travail entre nous à partir des classes données ainsi que les méthodes nécessaires pour coder un algorithme génétique.

Tâches	Participants
Classe Algorithm	Tout le groupe
Classe Solution	Tout le groupe
Classe Problem	BouBou et Antoine
Classse SetUpParams	Boubou
Code des Benchmark	Boubou
Présentation et rapport	Asmae et Insaf

Nous avons détaillé les tâches partagées par les méthodes normalement utilisées et les écrire correctement en fonction de l'algorithme WDE.

Initialisation	Asmae
Mutation	Boubou
Croisement	Insaf
Sélection	Antoine

5. Outils utilisés

Afin de faciliter le travail du groupe, de bien communiquer entre nous et de réaliser un projet correcte et bien fait. Nous avons utilisé des outils conçus pour ces objectifs.

En ce qui concerne la communication entre les membres du groupe, nous avons utilisé GitHub pour partager les codes et tous les fichiers du projet, ainsi que Facebook Messenger pour parler entre nous et bien divisés les tâches aussi bien que de s'entre aider.

Nous avons utilisés Codeblocks pour coder l'algorithme et les fonctions Benchmark en C++, et GanttProject pour diviser les tâches et de suivre un temps de travail défini et bien détaillé.

6. Le déroulement du projet

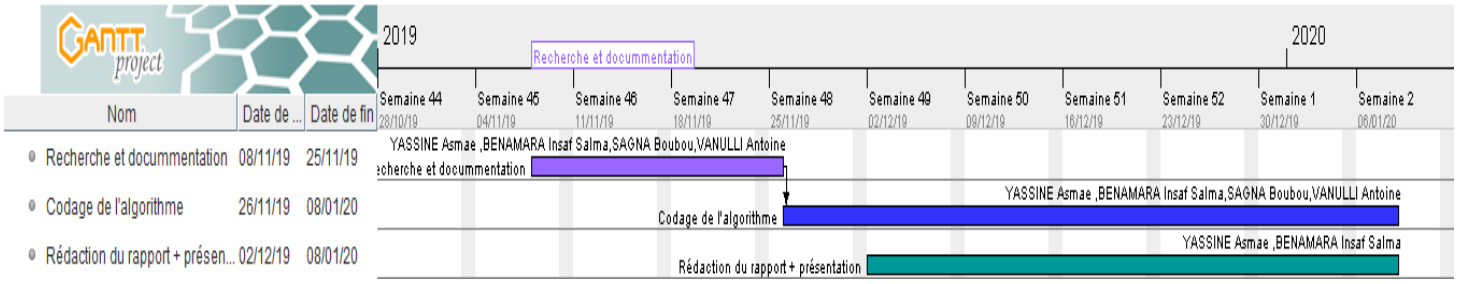
En premier lieu, nous devrions commencer par comprendre le concept du WDE ainsi que son objectif. Ensuite nous devrions trouver le pseudo code de l'algorithme, le diviser et essayer de comprendre le fonctionnement de chaque partie individuellement puis dans l'ensemble de l'algorithme.

Nous devrions aussi chercher les codes des fonctions Benchmark données, BentCigar, Discus, Katsuura, HGBat, HappyCat et Weierstrass, et les coder nous-même en C++.

Après avoir recueillis un nombre de document sur le sujet, nous avons partagé les partis du code entre nous. Chacun d'entre nous, a codé sa partie en c++, puis nous avons implémenté toutes les méthodes dans l'algorithme tout en respectant les classes données préalablement.

Alors que nous travaillons sur e code, nous avons commencé à rédiger le rapport et la présentation en parallèle.

Finalement, nous avons corrigés les erreurs et bugs dans le programme pour acquérir des résultats justes.



7. Conclusion

Ce projet nous a permis de travailler nos compétences en programmation c++, ainsi que de travailler en équipe, tout en respectant nos tâches et en nous aidant mutuellement à aller de l'avant avec le projet pour atteindre les résultats attendus de nous.

Nous avons dû travailler avec des ressources inconnu que nous avons dû découvrir et essayer de comprendre du début à la fin. Cet aspect nous a aidés à entrer dans l'inconnu et à ne pas en avoir peur, au lieu de cela, nous avons dû y arriver, avec des recherches pour mieux le comprendre et prendre notre temps avec le problème et trouver des solutions appropriées.