# Intelligence Artificielle
# Differential Evolution Algorithm

Rat Joris - Riabi Mohammed - Razafindratsita Florian - Prigent Cathie

March 8, 2015

# Contents

# Chapter 1

# Avant-propos

Ce rapport concerne les travaux realises dans le cadre du cours d'Intelligence Artificielle en seconde annee de cycle Ingenieur a l'ENSISA, sous la direction de M. Idoumghar Lhassane. Il traitera de l'algorithme DEA (Algorithme d'Evolution Differentielle), des differentes benchmarks qui nous ont ete indiquees pour tester notre implementation de l'algorithme et les resultats que nous avons obtenus.

La repartition des taches s'est faite de la facon suivante :

- Code :
  - Solution : Rat Joris - Riabi Mohammed - Razafindratsita Florian - Prigent Cathie
  - Set up Params : Rat Joris - Razafindratsita Florian
  - Problem : Riabi Mohammed
  - My algorythm : Rat Joris - Riabi Mohammed
- Rapport : Prigent Cathie
- Support de presentation : Razafindratsita Florian

# Chapter 2

# Algorithme d'Evolution Differentielle

## 2.1 Presentation generale

Cet algorithme a ete developpe par K. Price et R. Storn en 1995. Il s'applique aux valeurs reelles, et contrairement a d'autres alogrithmes classiques, il ne requiert pas que le probleme soit derivable puisqu'il n'utilise pas le gradient, ce qui permet de l'appliquer a des problemes bruites ou changeants. Son fonctionnement repose sur la sauvegarde des nouveaux individus issus du processus de croisement et de mutation a la seule condition qu'ils soient meilleurs que le parent qui a servi a les generer.

## 2.2 Fonctionnement

Voici une description etape par etape du fonctionnement de l'algorithme :

- Le premier individu I de la population est selectionne
- Deux autres individus distincts A et B sont selectionnes au hasard
- Un troisieme individu C est selectionne aleatoirement : il servira de partenaire pour le croisement
- Un vecteur de difference est calcule a partir de A et B, puis est pondere par F
- Le vecteur de difference est additionne a C
- L'individu de la nouvelle generation, N, est cree si le croisement reussi
- La valeur de la fonction d'objectif de N est evaluee
  - Si elle est meilleure que celle de I, N le remplace

– Sinon, I reste dans la population

## 2.3 Parametres

### 2.3.1 Description

Ces parametres restent constants pendant l'application de l'algorithme :

- CR (Crossover Probability) $\in [0,0;1,0]$ : Chance que deux individus selectionnes donnent un descendant
- F $\in [0,0;2,0]$ : Quantifie les variations differentielles
- NP : Taille de la population

### 2.3.2 Influence

Ces parametres influent sur la convergente de l'algorithme :

- Pas de convergence quand $F < 0,3$ ou $CR = 1$
- $F$ faible tend a augmenter la vitesse de convergence, mais peut faire stagner la resolution si $F < 0,4$

Pour notre simulation, nous avons choisis les valeurs suivantes :

- $F = 0.42$
- $CR = 0,6$

## 2.4 Observations

Apres avoir garde $F = 1.3$ pendant un grand nombre d'essais, nous avons constate que l'algorithme convergeait peu, sauf sur de petits intervalles. Changer pour $F = 0.42$ permet a l'algorithme de converger plus vite, conformement a nos recherches sur les parametres, mais pas necessairement sur de plus grands intervalles. Notre hypothese est que la nature mme de l'algorithme le fait fonctionner mieux sur de petits intervalles.

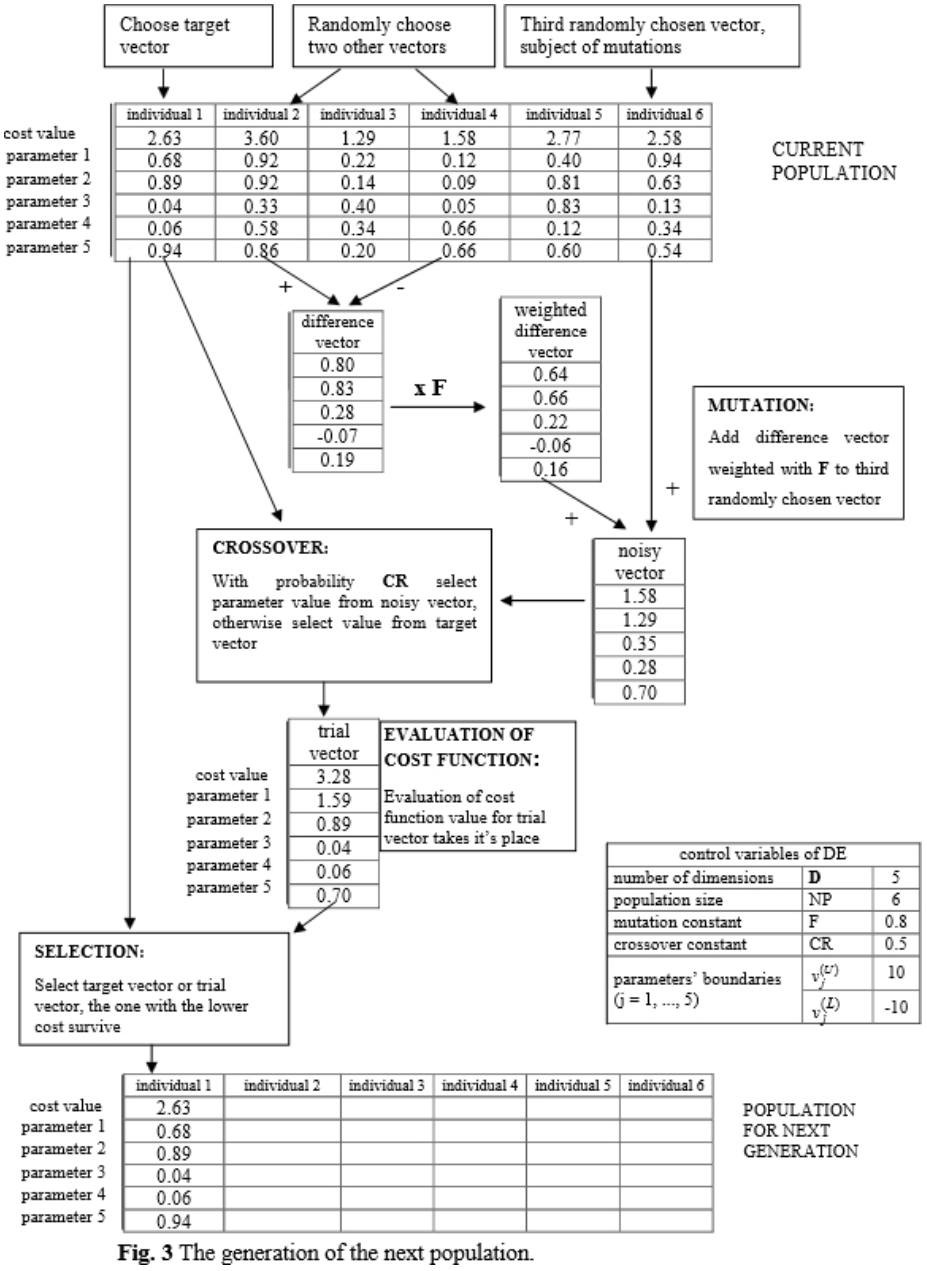


**Fig. 3** The generation of the next population.

Figure 2.1: Diagramme de presentation du fonctionnemet de l'algorithme

# Chapter 3

# Code

## 3.1 Set up Params

Cette classe contient l'ensemble des parametres necessaires a la bonne marche des simulations et de l'algorithme.

## 3.2 Problem

Cette classe contient l'arbre des choix des differents problemes ainsi que le choix des parametres permettant de les resoudre (domaine de definition, par exemple).

## 3.3 Solution

Cette classe contient l'implementation des tableaux de vecteurs qui representent les solutions du probleme et la definition des differents benchmarks necessaires pour evaluer l'efficacite de notre algorithme.

## 3.4 MyAlgorithm

Cette classe contient notre implementation de l'algorithme DEA.

```
void MyAlgorithm::runDEA() {

  vector<Solution*>::const_iterator it = _solutions.begin();
  vector<double>::iterator ita ;
  vector<double>::iterator itb;
  vector<double>::iterator itc;
  vector<double>::iterator itx;
  vector<struct  particle>::iterator itv =  _fitness_values.begin();
 int ind_x = 0;
```

```
Solution s1(solution(ind_x));
Solution a(solution(ind_x));
Solution b(solution(ind_x));
Solution c(solution(ind_x));
Solution& x=solution(ind_x);
for (; it != _solutions.end(); ++it)

{
   x = (*(*it)); // j'ai ajouter une reference comme sa si on modifier le x sa sera modifie
  int ind_a, ind_b, ind_c;
  do
  {
    ind_a = ((double)rand() / (double)RAND_MAX) *((double) _setup.population_size() - 1);
  } while (ind_a == ind_x);

  do
  {
    ind_b = ((double)rand() / (double)RAND_MAX) *((double) _setup.population_size() - 1);
  } while (ind_b == ind_x || ind_b == ind_a);

  do
  {
    ind_c = ((double)rand() / (double)RAND_MAX) *((double) _setup.population_size() - 1);
  } while (ind_c == ind_x || ind_c == ind_b || ind_c == ind_a);

  a = solution(ind_a);
  b = solution(ind_b);
  c = solution(ind_c);

  int r = (rand() % x.pbm().dimension()) + 1;
  ita = a.solution().begin();
  itb = b.solution().begin();
  itc = c.solution().begin();
  itx = x.solution().begin();
  for (int i=0; ita < a.solution().end(); ++ita,++itb,++itc,i++)
  {
    double u = rand() / RAND_MAX;
    if (u < CR || (ita - a.solution().begin()) == r - 1)
    {
      s1.position(i,*ita + F * (*itb - *itc));
    }
    else
    {
      s1.position(i,*itx);
    }
  }
```

```
   s1.fitness();
   x = (s1.get_fitness()<x.get_fitness())? s1:x;
   for(;itv != _fitness_values.end(); ++itv)
   {
   if((*itv).index == ind_x)
   {
   (*itv).fitness =x.get_fitness();
   break;
   }
   }
   itv = _fitness_values.begin();
   ind_x++;
 }
}
```

# Chapter 4

# Resultats

Tous les tests des benchmarks ont t raliss avec les conditions suivantes :

- Domaine de dfinition : $[-50; 50]$ (sauf contradications des  la fonction)
- Taille de la population : 30
- Nombre d'excution indpendantes : 30
- Pas d'volution : 5000

Les tudes de moyenne et d'cart-type ont t ralises avec la fitness de chaque meilleure solution des xcutions indpendantes.

## 4.1 Rosenbrock

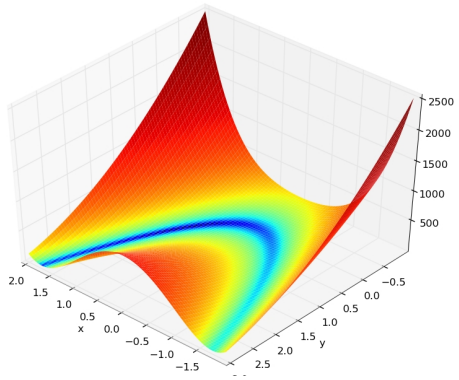


Figure 4.1: Fonction de Rosenbrock en 3 dimensions (x,y,fitness)
Objectif : $f(1, ..., 1) = 0$
Resultats obtenus :

- Ecart-type : $9,31.10^{7}$
- Moyenne : $3,62.10^{7}$

## 4.2 Rastrigin

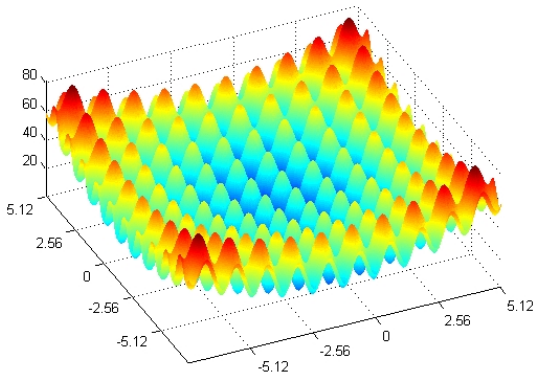


Figure 4.2: Fonction de Rastrigin en 3 dimensions (x,y,fitness)
Domaine de definition : $[-5.12; 5.12]$
Objectif : $f(0, ..., 0) = 0$
Resultats obtenus :

- Ecart-type : $9,52.10^{-1}$
- Moyenne : $4.10^{-1}$

## 4.3 Ackley

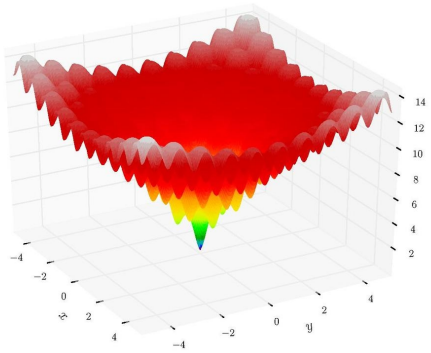

Figure 4.3: Fonction de Ackley en 3 dimensions (x,y,fitness)
Domaine de definition : $[-32; 32]$
Objectif : $f(0, ..., 0) = 0$
Resultats obtenus :

- Ecart-type : $5, 41$
- Moyenne : $9, 92$

## 4.4 Schwefel

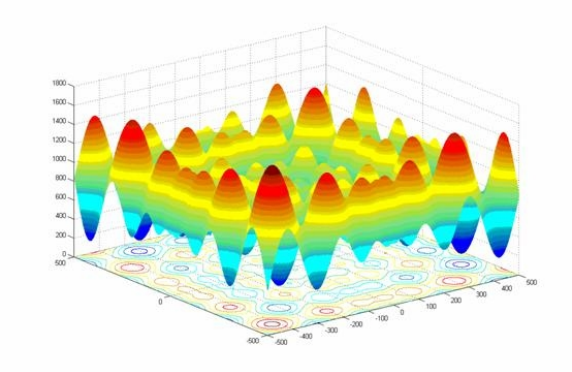

Figure 4.4: Fonction de Schwefel en 3 dimensions (x,y,fitness)
Objectif : $f(420.9687, ..., 420.9687) = 0$
Resultats obtenus :

- Ecart-type : $2, 36.10^1$
- Moyenne : $8, 09.10^2$

## 4.5 Schaffer

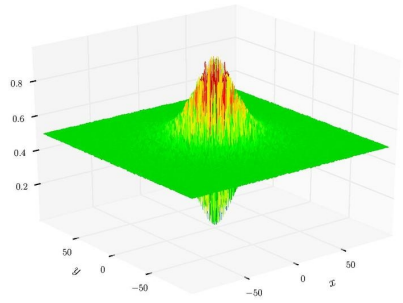

Figure 4.5: Fonction de Schaffer en 3 dimensions (x,y,fitness)
Domaine de definition : $[-100; 100]$
Objectif : $f(0, ..., 0) = 0$
Resultats obtenus :

- Ecart-type : $1,94.10^{-1}$
- Moyenne : $3,30.10^{-1}$

## 4.6 Weierstrass

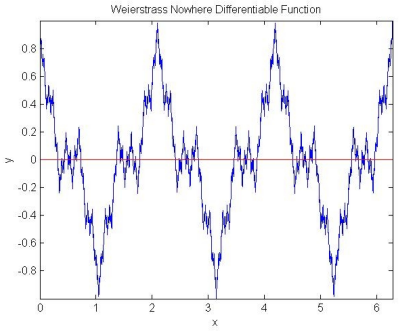


Figure 4.6: Fonction de Weierstrass en 2 dimensions (x,y)
Hypotheses : $a = 0.5$ $(0 < a < 1)$ et $b = 9$ ($b$ impair positif et $ab > 1 + \frac{3}{2}\pi$)
Objectif : $f(0, ..., 0) = 0$
Resultats obtenus :

- Ecart-type : $1,32.10^{-5}$
- Moyenne : $-1,5$

# Chapter 5

# Sources

Les sources qui ont ete utilisees dans l'implementation de cet algorithme et la realisation de ce rapport sont indiquees ci-dessous :

- http://www.tandfonline.com/doi/pdf/10.1623/hysj.49.1.183.54001
- http://sci2s.ugr.es/eamhco/cec2010_functions.pdf
- https://en.wikipedia.org/wiki/Differential_evolution
- http://www.sfu.ca/ ssurjano/optimization.html