

Université des Sciences et de la Technologie Houari
Boumediene



Département d'informatique

Master 1 SII

Méta-heuristique et Algorithmes Evolutionnaires

Rapport Final :

PSO/GA/BSO

Binôme :

Zakaria Slimi G2

HEBBACHE Imad eddine G1

2019 / 2020

Table of Contents

1.1 Problématique :	4
1.2 Complexité :	4
1.3 Présentation des données :	5
1.4 Solution :	5
2-Les Algorithme Génétique :	6
2.1 Espace de recherche :	6
2.2- La Concept :	6
Croisement :	6
Mutation :	7
PseudoCode :	8
2.3. Expérimentation :	9
Remarque :	11
2.4- Performance :	11
Qualité de la solution :	11
Temps d'exécution :	12
3 Optimisation par essaims particulaires "PSO" :	13
3.1 PSO pour SAT :	13
3.1.1 Codage de solution :	13
3.1.2 Espace de recherche :	13
3.1.3 L'algorithme :	13
3.2 L'application :	15
3.4 Expérimentations:	22
3.4.1 Les Configuration :	22
3.4.2 Les Résultats :	22
3.4.3 Analyse de résultat :	26
3.5 Conclusion :	27
4. Optimisation par essaims d'abeilles "BSO" :	28
4.1 L'application :	28
Les Classes :	28
Instance :	29
La Fonction Principale :	29
Bee :	30

Fonctions:.....	30
Clause	31
Solution	32
BSO	33
4.2. PseudoCode:	41
4.3Résultats:	43
5- Comparaison :	48
6- Conclusion générale.....	49
7. Références:	54

1.1 Problématique :

Le problème de satisfiabilité ou en abrégé « problème SAT » est un problème de décision c.à.d. une question accompagnant la description de l'instance dont la réponse est soit « OUI » soit « NON ».

Instance : $X = \{x_1, x_2, \dots, x_n\}$ un ensemble de littéraux, $C = \{c_1, c_2, \dots, c_n\}$ un ensemble de clauses telle que :

Un littéral : est une variable propositionnelle x (littéral positif) ou la négation d'une variable propositionnelle $\neg x$ (littéral négatif).

Une clause C : est une disjonction de littéraux.

Forme Normal Conjonctif « CNF » : est une conjonction de clauses

Exemple :

$V = \{v_1, v_2, v_3, v_4\}$, $C = \{c_1, c_2, c_3\}$ such that :

$$c_1 = v_1 + v_2 + v_4$$

$$c_2 = v_2 + v_3 + v_4$$

$$c_3 = v_1 + v_2 + v_3$$

QUESTION : Etant donné une Formule de Logique Propositionnelle F , existe-il une instanciation (un ensemble de valeurs booléennes associées aux littéraux) de l'ensemble des littéraux X telle que la conjonction des clauses de C est vraie ? Autrement dit : existe une affectation des littéraux qui rend cette formule F vraie (satisfiable).

1.2 Complexité :

Il s'agit de trouver la solution optimale (La solution que satisfait toutes les clauses)

⇒ Et comme un littéral peut prendre 2 valeur (vrais/faux) donc au pire cas on parcourt toutes l'espace de recherche C'est à dire $O(n) = 2^n$

1.3 Présentation des données :

⇒ La structure utilisée est une matrice d'entier « $N \times 3$ » pour présenter les clauses

1	8	11
-7	10	4
-20	24	15
...		
...		
23	30	-7



1	8	11
-7	10	4
-20	24	15
..

1.4 Solution :

⇒ La structure d'une solution est un tableau de n bit chaque bit représentant une variable

1/0	1/0	1/0	1/0	1/0
-----	-----	-----	-----	-----	-----	-----

2-Les Algorithme Génétique :

Les algorithmes génétiques sont des algorithmes d'optimisation stochastique fondés sur les mécanismes de la sélection naturelle et de la génétique, dont on prend une population initiale et on applique une suite de croisements entre eux, de telles façons à obtenir une nouvelle génération (on peut obtenir quelques mutations).

2.1 Espace de recherche :

Il s'agit de la population qui est constituée d'un nombre n fixe de solutions, initialement ses solutions sont générées aléatoirement, ensuite dans chaque itération et après avoir fait le croisement, le nombre de la population change de telle façon qu'on ne garde que les meilleures solutions.

Du coup, l'espace de recherche devient les meilleures solutions entre les générations.

2.2- La Concept :

Croisement :

Le croisement est le fait de mélanger les chromosomes des parents (2 solution choisit aléatoirement de la population) afin de créer une nouvelle génération différente à partir de la génération précédente. Il existe plusieurs méthodes de croisements, dans notre projet on a fait de telle sorte qu'on choisit un bit inférieur au nombre des variables, les bits qui se trouvent avant le bit choisi dans le père numéro 1 vont être identiques à ceux du 1er fils et le reste des bits du 1er fils vont être les bits qui se trouvent après le bit choisi dans le père numéro 2.

```

public Population crossoverPopulation(Population p) {
    Population P= new Population(p.size());

    for (int i = 0; i < p.size(); i++) {
        Individual parent1 = p.getFittest(i);
        Individual tmp =parent1;
        if (this.crossoverRate > Math.random()) {
            Individual o = new Individual(parent1.getChromosomeLength());

            Individual parent2 = selectParent(p);

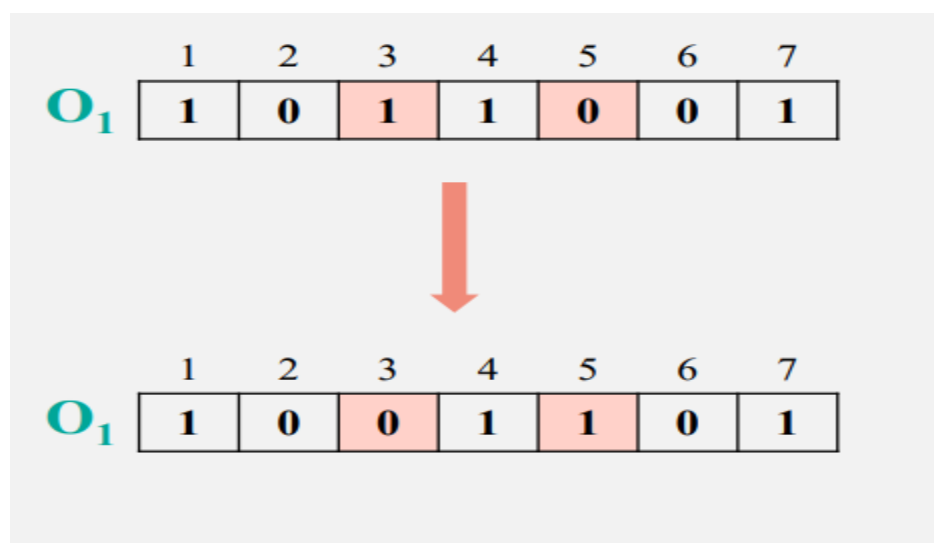
            if (0.5 > Math.random())
            {
                parent1=parent2;
                parent2=tmp;
            }
            for (int j = 0; j < (int) Math.round(Math.random()*parent1.getChromosomeLength());j++) {
                o.setGene(j, parent1.getGene(j));
            }
            for (int j = (int) Math.round(Math.random()*parent1.getChromosomeLength());j <parent1.getChromosomeLength();j++) {
                o.setGene(j, parent1.getGene(j));
            }
            P.setIndividual(i, o);
        } else {
            P.setIndividual(i, parent1);
        }
    }
}

return P;
}

```

Mutation

La mutation se fait par un choix d'un nombre aléatoire puis inverser ce nombre (si c'est un 0, il devient un 1 et vice-versa)



```

public Population mutatePopulation(Population Po) {
    Population P = new Population(this.populationSize);

    for (int i = 0; i < Po.size(); i++) {
        Individual individual = Po.getFittest(i);

        for (int j = 0; j < individual.getChromosomeLength(); j++) {
            if (this.mutationRate > Math.random()) {
                int h = 1;
                if (individual.getGene(j) == 1) {
                    h = 0;
                }
                individual.setGene(j, h);
            }
        }

        P.setIndividual(i, individual);
    }

    return P;
}

```

PseudoCode :

Input: PopulationSize, ProblemSize, Pcrossover, Pmutation

Output: SGbest

Population := InitializePopulation(PopulationSize, ProblemSize)

 EvaluatePopulation(Population)

 SGbest := GetBestSolution(Population)

While (not StopCondition())

 Parents := SelectParents(Population, PopulationSize)

 Children := {}

 For (Parent1, Parent2 ∈ Parents)

 Child1 := Crossover(Parent1, Parent2, Pcrossover)

 Child2 := Crossover(Parent2, Parent1, Pcrossover)

 Children := Mutate(Child1, Pmutation)

 Children := Mutate(Child2, Pmutation)

 End.

 EvaluatePopulation(Children)

 Sbest := GetBestSolution(Children)

 if (Sbest > SGbest) then Sgbest := Sbest

 Population := Replace(Population, Children)

End

Return (Sbest)

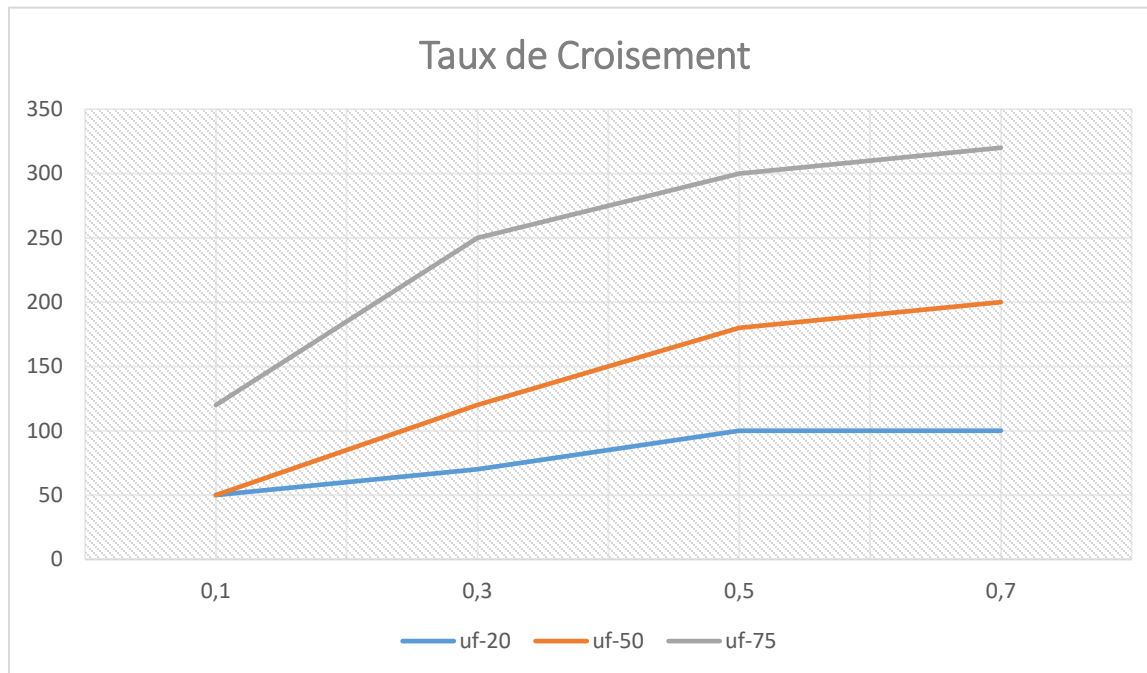
2.3. Expérimentation :

L'algorithme génétique dépend de 4 paramètres :

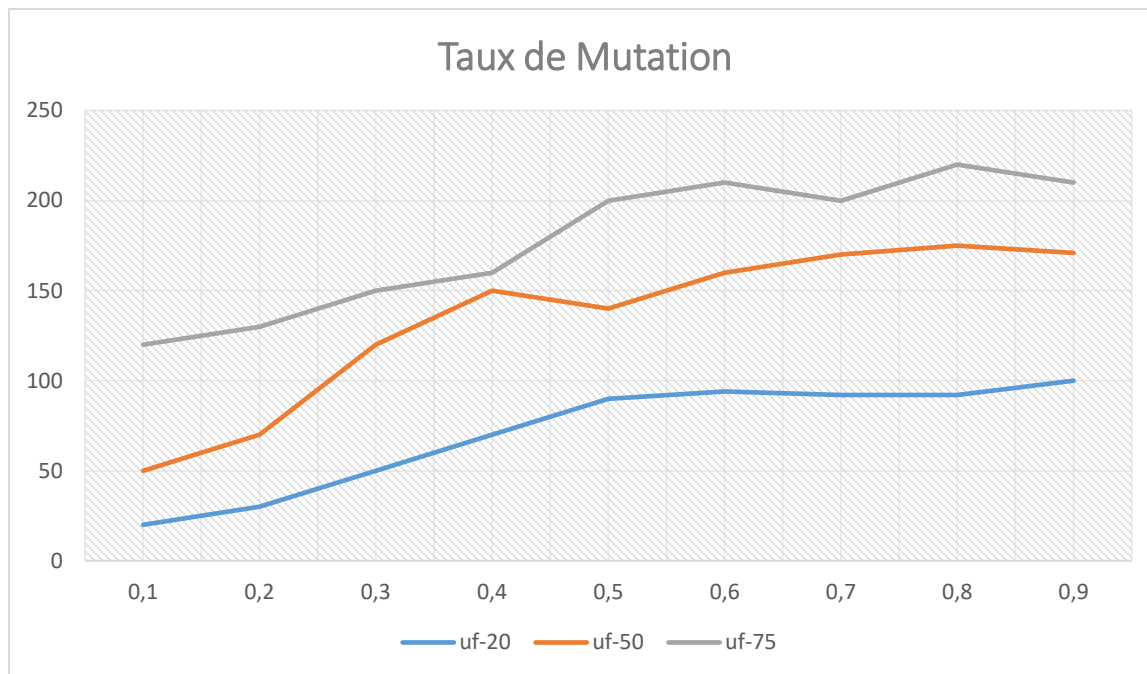
private int populationSize; *//la taille de population*

private double mutationRate; *//Probabilité pour individuelle mutata*

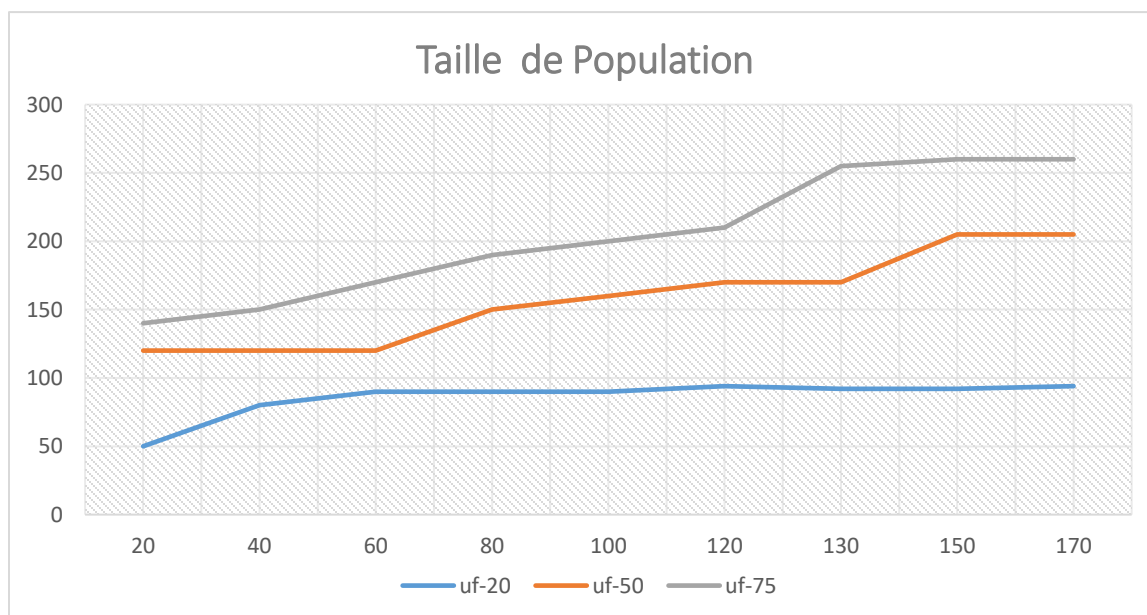
private double crossoverRate; *// Probabilité pour 2 individuelle crossover*



Les meilleures valeurs du taux de croisement obtenues pour les fichiers uf20, uf50, uf75 sont 0.4, 0.7, 0.9 respectivement.



Les meilleures valeurs du taux de mutation pour les fichiers uf20, uf50, uf75 sont 0.3, 0.6, 0.8 respectivement.



Les meilleures valeurs de la taille de population obtenues pour les fichiers uf20, uf50, uf75 sont 50,100, 130 respectivement.

Remarque :

« Les valeurs d'un paramètre se diffèrent d'un type de fichier à un autre (cela est dû à l'augmentation de la complexité qui est liée à l'augmentation des nombres de variables).»

uf20

PopSize	50
CrossRate	0.4
MutRate	0.3
Nblter	1500

uf50

PopSize	100
CrossRate	0.7
MutRate	0.6
Nblter	2000

uf75

PopSize	130
CrossRate	0.9
MutRate	0.8
Nblter	4000

2.4- Performance :

La performance inclut la qualité de la solution et le temps d'exécution.

Qualité de la solution :

En fixant les 4 paramètres cités précédemment aux meilleures valeurs obtenues dans les réglages des paramètres, on parvient la solution optimale pour chacune des instances des fichiers (uf20, uf50, uf75) et cela par rapport au nombre de clauses satisfaites par cette dernière (nombre de clauses satisfaites par la solution est égale au nombre total des clauses des instances des fichiers).

Uf-20	91	91	100%
Uf-50	218	202	93%
Uf-75	325	301	92%

Temps d'exécution :

La solution optimale pour chacune des instances des fichiers (uf20, uf50, uf75) assure un temps d'exécution minimum et très réduit par rapport aux autres solutions (entre 5 et 10 secondes) .

D'après « le pourcentage de la qualité de la solution » et « le temps d'exécution » de chacune des instances des fichiers (uf20, uf50, uf75), on peut conclure que la solution est performante autrement dit L'algorithme génétique a réussi à trouver la solution optimale en un temps réduit i.e. solution qui atteint la meilleure performance pour chaque instance.

3 Optimisation par essais particuliers “PSO” :

- ⇒ Cet algorithme s’inspire à l’origine du monde du vivant, il s’appuie notamment sur un modèle permettant de simuler le déplacement d’un groupe d’oiseaux [1].
- ⇒ Cette méthode d’optimisation se base sur la collaboration des individus entre eux.
- ⇒ Cette idée veut qu’un groupe d’individus peu intelligents puisse posséder une organisation globale complexe.
- ⇒ Ainsi grâce à des règles de déplacements, les particules peuvent converger progressivement vers un minimum local.
- ⇒ Cette méta heuristique fonctionne mieux pour des espaces en variables continues (à l’origine, PSO a d’abord été conçu pour une optimisation continue, Il a ensuite été adapté aux problèmes combinatoires discrets).

3.1 PSO pour SAT :

3.1.1 Codage de solution :

La solution est un tableau de N bit

3.1.2 Espace de recherche :

L’ensemble de toutes les instanciations potentielles pour l’instance, il s’agit alors de l’ensemble des vecteurs booléens de longueur égale à n (l’ensemble des particules). La taille de l’espace de recherche est égale à 2^n .

3.1.3 L’algorithme :

Debut

Initialiser N particules: positions and velocities;

Évaluer les positions de chaque particule;

Pour chaque particules i faire $P_{besti} = x_i$;

Calculer Gbest on utiliser la règle de transition ;

Pour chaque iteration faire

 Pour chaque particule p faire

 Mise à jour velocity et position;

 Déplacer la particule et évaluer sa fitness;

 Mise à jour Pbest;

 Fin Pour;

 Mise à jour Gbest;

Fin Pour;

Fin

Updating the velocity:

$$v(t+1) = w * v(t) + c1 * r1 * (P_{best} - x(t)) + c2 * r2 * (G_{best} - x(t))$$

where:

- $x(t)$ the current position of the particle.
- P_{best} is the best visited position of the particle.
- G_{best} is the best position found by the swarm.
- $w, c1, c2$ are empirical parameters.
- $r1, r2$ are random numbers.

Updating the position:

$$x(t+1) = x(t) + v(t+1)$$

3.2 L'application :

Les Classe :

ReadConfig

Partie déclaration

```
private int NbLtr;  
private int nbClause;  
private ArrayList<Clause> Clauses;
```

La Fonction Principale :

```
public void GetConfig() {  
    try {  
        File Cnf = new File(this.ConfigPath);  
        Scanner myReader = new Scanner(Cnf);  
        String data=" ";  
  
        /**Skiplines***/  
  
        while ((myReader.hasNextLine()) && (data.charAt(0) != 'p')) {  
            data = myReader.nextLine();  
        }  
  
        String ch[]=this.spliteData(data);  
        this.NbLtr=Integer.parseInt(ch[2]);  
        this.nbClause=Integer.parseInt(ch[3]);  
  
        Clauses=new ArrayList<>(this.nbClause);  
  
        int x=0;  
  
        while (myReader.hasNextLine() & (x<this.nbClause)) {  
            data = myReader.nextLine();  
            Clauses.add(new Clause(data));  
            x++;  
        }  
        myReader.close();  
  
        catch (FileNotFoundException e) {  
  
            System.out.println("An error occurred.");  
            e.printStackTrace();  
  
        }  
    }  
}
```

Clause

Partie déclaration :

```
private int Clause[];
```

Les Fonction :

```
public Clause(String Clause)
{
    this.Clause=this.TransClause(Clause);
}
int[] getClause()
{
    return this.Clause;
}
private int[] TransClause(String Data)
{
    Data=Data.trim();
    String ch[]= Data.split("\\s+");
    int[] c=new int[ch.length-1];
    for(int i=0;i<ch.length-1;i++)
    {
        c[i]=Integer.parseInt(ch[i]);
    }
    return c;
}
int GetFirst()
{
    return this.Clause[0];
}
```

Particle

Partie declaration :

```
int Velocity;
int Solution[];
Particle PBest;
```

Les Fonction :

2 constructeurs :

```
Particle(int Size)
{
    Solution = new int[Size];
    Random R = new Random();
    Velocity = R.nextInt(4);
    for (int i = 0; i < Solution.length ; i++)
    {
        Solution[i] = R.nextInt(2);
    }
    PBest=new Particle(Solution);
}
Particle(int[] Solution)
{
    this.Solution=new int[Solution.length];
    System.arraycopy(Solution, 0, this.Solution, 0, Solution.length);
}
```

Calculer La fonction Fitness

```
int GetFit(ArrayList<Clause> Clauses)
{
    int a,b,Correct,Fit=0;
    int C [];
    for (Clause Clause : Clauses) {
        C = Clause.getClause();
        Correct=0;
        for (int i= 0; i < C.length; i++) {
            a=C[i];
            b=Integer.max(Integer.signum(a),0);
            a=Integer.signum(a)*a;

            if (b == Solution[a-1]) {
                Correct=1;
                break;
            }
        }
        Fit += Correct;
    }
    return Fit;
}
```

Mise à jour vitesse

```

void UpdateVelocity(double W ,double C1 , double C2,int Vmax ,Particle GBest)
{
    int GD = this.Distance(GBest);
    int PD = this.Distance(this.PBest);
    double r1=Math.random();
    double r2=Math.random();
    this.Velocity = (int) Math.round(W * this.Velocity + C1 * r1 * PD + C2 * r2 * GD) ;

    if (this.Velocity > Vmax)
    {
        this.Velocity = 1;
    }
}

```

Mise à jour Position :

```

void UpdatePosition(ArrayList<Clause> C)
{
    Random R=new Random();
    int a;
    for (int i = 0 ; i < this.Solution.length ; i++)
    {
        a=R.nextInt(Solution.length);
        this.Solution [a] = (this.Solution[a]+1)%2;
    }
    if (this.GetFit(C)>PBest.GetFit(C))
    {
        PBest=new Particle(this.GetSolution());
    }
}

```

Hamming Distance

```

int Distance(Particle P)
{
    int a = 0;
    for (int i = 0 ; i < this.Solution.length ; i++)
    {
        if ( P.GetSolution()[i] != this.Solution[i])
        {
            a++;
        }
    }
    return a;
}

```

PSO

Partie déclaration :

```
int NbInst;  
int NbrVar;  
ArrayList<Clause> Clauses;  
  
int MaxIter;  
  
int ParticlesSize;  
ArrayList<Particle> Particles;  
  
double C1,C2,W;  
int Vmax;
```

constructeur

```
PSO (int NbInst , int NbrVar , ArrayList<Clause> Clauses,int ParticlesSize,  
    int MaxIter, double C1, double C2, int Vmax, double W)  
{  
    this.NbInst = NbInst;  
    this.NbrVar = NbrVar;  
    this.Clauses = Clauses;  
    this.ParticlesSize = ParticlesSize;  
    this.C1 = C1;  
    this.C2 = C2;  
    this.Vmax = Vmax;  
    this.W = W;  
    this.MaxIter=MaxIter;  
}
```

Initialiser les particulier :

```
void InitParticles ()  
{  
    Particles= new ArrayList<>();  
    for (int i = 0;i < ParticlesSize ; i++)  
    {  
        this.Particles.add(new Particle(this.NbrVar));  
    }  
}
```

Fonction Principale

```

Particle Run()
{
    this.InitParticles();
    Particle GBest = this.GetBest();
    int i = 0;
    Particle BestSol;
    int BF=0,a,Gen=0;
    boolean Found = false;
    Instant i1 = Instant.now(),i2=Instant.now();
    while ( i< MaxIter && !Found )
    {
        for (Particle P : this.Particles) {
            P.UpdateVelocity(W, C1, C2, Vmax, GBest);
            P.UpdatePosition(Clauses);
        }
        BestSol = this.GetBest();
        a = BestSol.GetFit(Clauses);
        BF = GBest.GetFit(Clauses);
        if (a > BF)
        {
            GBest =(Particle) BestSol.clone();
            i2 = Instant.now();
            BF =a;
            Gen=i;
        }

        if (BF == NbInst)
        {
            Found = true;
        }
        System.out.println("*****Best Solution*****{"+"i+"}: "+
            Arrays.toString(GBest.GetSolution())+" "+GBest.GetFit(Clauses));
        i++;
    }
    System.out.println("*****Best Solution Found*****{"+"Gen+"}: "+
        Arrays.toString(GBest.GetSolution())+" in "+Duration.between(i1, i2).toNanos());
    return GBest;
}

```

Obtenez la meilleure particule:

```
Particle GetBest()
{
    Particle P = Particles.get(0);
    int PF = P.GetFit(Clauses);
    int a;
    for (int i = 1 ; i < this.Particles.size(); i++)
    {
        a=Particles.get(i).GetFit(this.Clauses);
        if(a>PF)
        {
            P = Particles.get(i);
            PF=a;
        }
    }
    return P;
}
```

Main

```
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        String Path="config/uf75-01.cnf";
        ReadConfig Rf=new ReadConfig(Path);
        int Iter = 10000;
        double C1 = Math.random()*2;
        double C2 = Math.random()*2;
        int Vmax = 20;
        double W = Math.random();
        int PS = 100;
        PSO P= new PSO(Rf.GetNbClause(),Rf.GetNbLtr(),Rf.GetClauses(),PS,Iter,C1,C2,Vmax,W);
        Particle Sol = P.Run();
    }
}
```

3.4 Expérimentations:

Comme pour le GA, un ajustement des paramètres est fait dans le but de trouver la meilleure combinaison des hyper paramètres en entrée du PSO.

3.4.1 Les Configuration :

Iter = 10000; *** Nombre d'itération ***

C1 = 1.5;

C2 = 1.25;

Vmax = 20;

W = 10 ;

PS = 200 ; *** Nombre des particules ***

Et en utiliser 3 Benchmarks : uuf75-325 – uuf100-430 – uf20-91

3.4.2 Les Résultats :

Instance	Max Clause	Temps(s)	Taux de Satisfiabilité %
1	213	5	97
2	213	5	97
3	211	11	96
4	212	21	97
5	211	19	96
6	211	18	96
7	211	8	96
8	212	18	97
9	211	10	96
10	212	8	97
Moyen	211.7	12.3	96.5

Résultats détaillés pour Max-SAT BanchMark UUF50-218

Instance	Max Clause	Temps(s)	Taux de Satisfiabilité %
1	404	15	93
2	405	2	94
3	406	34	94
4	405	3	94
5	405	41	94
6	405	16	94
7	409	31	95
8	405	34	94
9	405	1	94
10	404	12	93
Moyen	405.3	18.9	93.9

Résultats détaillés pour Max-SAT BanchMark UUF100-430

Instance	Max Clause	Temps(s)	Taux de Satisfiabilité %
1	91	1	100
2	91	0	100
3	90	0	98
4	91	1	100
5	91	1	100
6	91	0	100
7	91	0	100
8	91	1	100
9	90	0	98
10	91	0	100

Moyen	90,8	0.4	99,6
-------	------	-----	------

Résultats détaillés pour Max-SAT BanchMark UF20-91

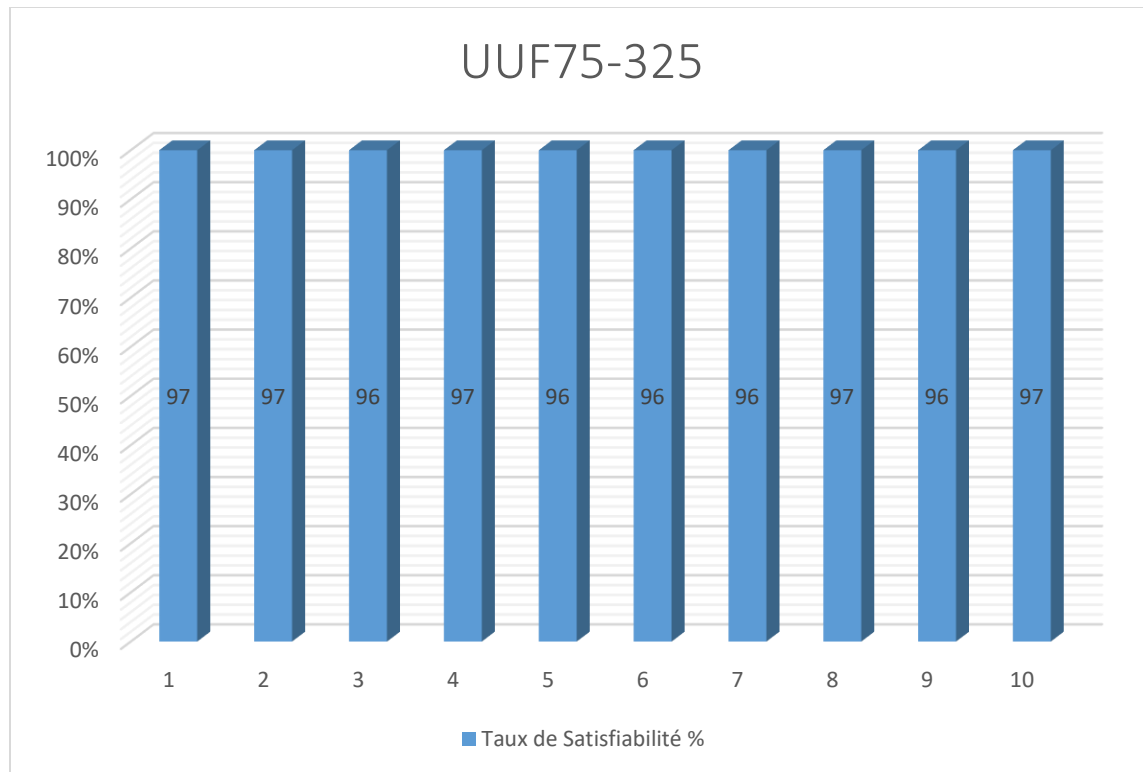


Figure 1: Histogramme montrant les résultats du test de l'algorithme PSO pour le problème SAT sur les fichiers de Benchmarks UUF-75

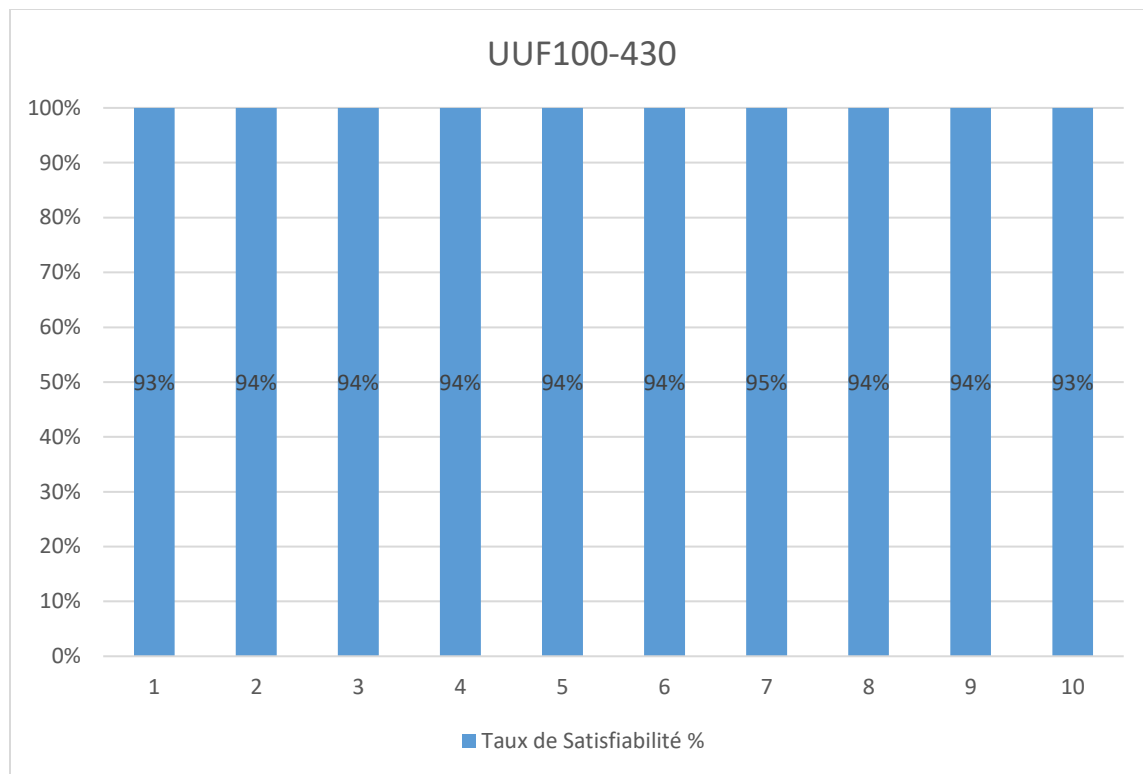


Figure 2: Histogramme montrant les résultats du test de l'algorithme PSO pour le problème SAT sur les fichiers de Benchmarks UUF-100

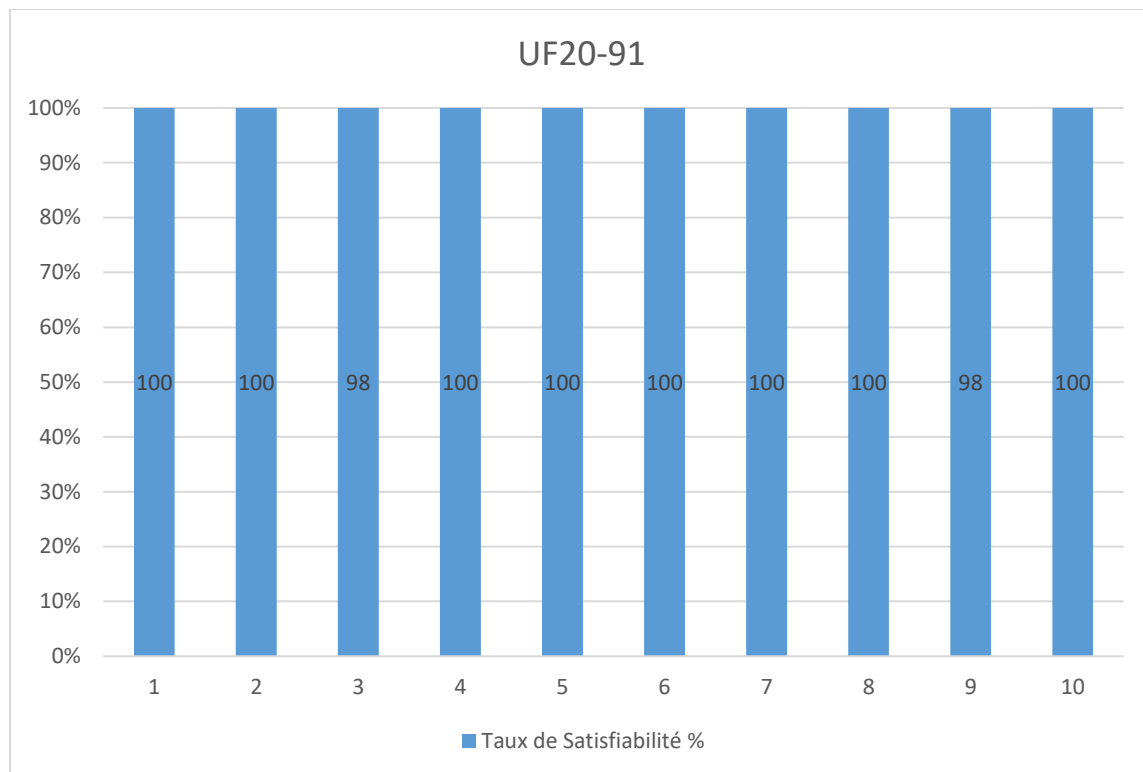


Figure 2: Histogramme montrant les résultats du test de l'algorithme PSO pour le problème SAT sur les fichiers de Benchmarks UF-20

3.4.3 Analyse de résultat : la qualité de la solution et le temps d'exécution

la qualité de la solution :

On parvient la solution optimale pour chacune des instances des fichiers (uf20, uf50, uf75) et cela par rapport au nombre de clauses satisfaites par cette dernière (nombre de clauses satisfaites par la solution est égal au nombre total des clauses des instances des fichiers)

Temps d'exécution :

La solution optimale pour chacune des instances des fichiers (uf20, uf50, uf75) assure un temps d'exécution minimum et réduit (entre 9 et 15 secondes).

D'après « le pourcentage de la qualité de la solution » et « le temps d'exécution » de chacune des instances des fichiers (uf20, uuf50, uuf100), on peut conclure que la solution est performante autrement dit PSO a réussi à trouver la solution optimale en un temps réduit.

3.5 Conclusion :

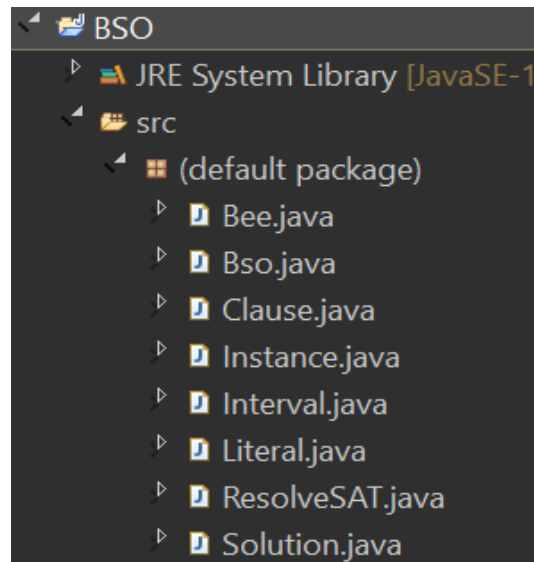
Après observation et analyse des résultats obtenus de l'exécution des algorithmes PSO, sur le problème SAT, on constate clairement que l'utilisation des méta-heuristiques permet d'avoir des solutions Plus optimales. Aussi, on remarque nettement l'efficacité de l'algorithme en terme de temps et Taux de satisfiabilité.

4. Optimisation par essaims d'abeilles “BSO” :

Les recherches et observation des colonies d’abeilles ont abouti au fait que la butineuse indique aux autres abeilles la source de nourriture qu’elle a trouvé grâce à des mouvements effectués en différentes vitesses. Ces mouvements sont dits « Dance ». Les abeilles communiquent donc par la dance ; Une dance très riche en information vu qu’elle permet à la butineuse de communiquer à ses congénères la direction de la source, la distance par rapport à la ruche ainsi que la richesse de cette source. Plus la source est riche, plus la vigueur avec laquelle l’abeille effectue cette danse augmente, de même que la cadence de ses visites. Parallèlement, l’intervalle de temps qui sépare son arrivée au nid du déchargement de sa récolte, diminue. Chaque butineuse, après avoir effectué sa récolte, effectue une dance pour inciter les autres abeilles à se rendre dans la zone contenant la source trouvée

4.1 L’application :

Les Classes :



Instance

Partie déclaration :

```
1 import java.util.ArrayList;
2
3 public class Instance {
4     int Nclause;
5     int Nvar;
6     ArrayList<Clause> instance;
7
8     public Instance(int nc,int nv) {
9         this.instance=new ArrayList<Clause>();
10        this.Nclause=nc;
11        this.Nvar=nv;
12    }
13 }
```

La Fonction Principale :

Évaluation des solutions.

```
37 public boolean ifSolution(boolean tab[]) {
38     boolean solution=true;
39     if(tab.length==this.Nvar) {
40         solution=this.getClause(0).interpretation(tab);
41         for(int i=1; i<Nclause;i++) {
42             solution=solution&this.getClause(i).interpretation(tab);
43         }
44         return solution;
45     }
46     else
47         return false;
48 }
49
50 public double evaluate(Bee bee) {
51     double fitness=0;
52     if(bee.searchLocal.solution.length==this.Nvar) {
53         for(int i=0;i<Nclause;i++){
54             if(this.getClause(i).interpretation(bee.searchLocal.solution))
55                 fitness=fitness+0.0030769230769231;
56         }
57         return fitness*100;
58     }
59     else
60         return fitness*100;
61 }
62 }
```

Bee

Partie déclaration :

```
1
2 public class Bee {
3
4     Solution searchLocal;
5     int iter;
6
7     public Bee(Solution searchLocal, int iter) {
8         super();
9         this.searchLocal = searchLocal;
10        this.iter = iter;
11    }
12
13    public Bee(int iter,int lenght) {
14        this.searchLocal = new Solution(lenght);
15        this.iter = iter;
16    }
17
```

Fonctions:

Beelnit : c'est la fonction qui reflète beelnit et renvoie la solution référence (Sref) initialement «cette méthode est utilisée que par la première abeille dite Beelnit».

```
55 public void BeeInit(int lenght,Instance ins) {
56     //bee init at start point = 0
57     //this.setSearchLocal(this.startPoint(lenght));
58
59     //bee init random
60     boolean tab[]=new boolean[lenght];
61     Interval interval=new Interval(0,lenght);
62     for(int i=0;i<lenght;i++) {
63         if(interval.getRandom()<(interval.getStart()+interval.end)/2)
64             tab[i]=true;
65         else
66             tab[i]=false;
67     }
68     this.searchLocal.setSolution(tab);
69     this.updateDance(ins);
70
71 }
```

exploration : pour exploré la surface de recherche courante

```

78
79• public void exploration() {
80     Interval inter=new Interval(0, searchLocal.getSolution().length);
81     int index;
82     for(int i=0;i<iter;i++) {
83         index=(int)inter.getRandom();
84         searchLocal.getSolution()[index]=!searchLocal.getSolution()[index];
85     }
86 }

```

dance:chaque abeille effectue une danse après la fin de l'exploitation

```

97
98• public double dance(boolean[] t,Instance i) {
99     return i.evaluate(t);
100 }
101 |
102
73
74• public void updateDance(Instance i) {
75     this.searchLocal.setDanceSrenght(this.dance(this.getSearchLocal().getSolution(), i));
76 }
77
78

```

Clause

Partie déclaration

```

2
3 public class Clause {
4     ArrayList<Literal> list;
5• public Clause() {
6         list=new ArrayList<Literal>();
7     }
8• public Clause(Literal l1,Literal l2,Literal l3) {
9         list=new ArrayList<Literal>();
10        list.add(l1);
11        list.add(l2);
12        list.add(l3);
13 }

```

La Fonction Principale :

```
31 public boolean interpretation(boolean tab[]) {  
32     boolean v1,v2,v3;  
33     v1=tab[list.get(0).var-1];  
34     if(list.get(0).isNeg())  
35         v1=!v1;  
36     v2=tab[list.get(1).var-1];  
37     if(list.get(1).isNeg())  
38         v2=!v2;  
39     v3=tab[list.get(2).var-1];  
40     if(list.get(2).isNeg())  
41         v3=!v3;  
42  
43     return v1||v2||v3;  
44 }
```

Solution

Cette classe représente la solution du problème.


```

1 |
2 public class Solution {
3     boolean[] solution;
4     double danceStrenght=0;
5     public Solution(boolean[] solution, double danceStrenght) {
6         this.solution = solution;
7         this.danceStrenght = danceStrenght;
8     }
9     public Solution(boolean[] solution) {
10        this.solution = solution;
11    }
12    public Solution(int lenght) {
13        this.solution = new boolean[lenght];
14    }
15    public boolean[] getSolution() {
16        return solution;
17    }
18    public void setSolution(boolean[] solution) {
19        this.solution = solution;
20    }
21    public double getDanceStrenght() {
22        return danceStrenght;
23    }
24    public void setDanceStrenght(double danceStrenght) {
25        this.danceStrenght = danceStrenght;
26    }
27 }
28 }

```

BSO

Partie déclaration

```

3 public class Bso {
4     int maxIter;
5     int flip;
6     int Nbees;
7     int maxChances;
8     int nbChances;
9     int locIter;
10    int lenght; //the number of variables in the instance
11    ArrayList<Bee> BeesTab;
12    ArrayList<Solution> searchArea;
13    ArrayList<Solution> danceTab;
14    Solution Sref;
15    ArrayList<Solution> tabooTab;
16 }

```

Constructeur

```
34 public Bso(int maxIter, int flip, int nbees, int maxChances, int locIter, int l, Solution sref) {
35     super();
36     this.maxIter = maxIter;
37     this.flip = flip;
38     Nbees = nbees;
39     this.maxChances = maxChances;
40     this.locIter = locIter;
41     this.lenght=l;
42     BeesTab = new ArrayList<Bee>();
43     this.creatBeeSwarm();
44     this.danceTab = new ArrayList<Solution>();
45     this.Sref = sref;
46     this.tabooTab = new ArrayList<Solution>();
47     this.searchArea = new ArrayList<Solution>();
48     nbChances=maxChances;
49 }
```

Init swarm of bees :

```
116
117 public void creatBeeSwarm() {
118     Bee b;
119     for(int i=0;i<this.getNbees();i++) {
120         b=new Bee(this.getLocIter(),this.getLenght());
121         this.BeesTab.add(b);
122     }
123 }
124
```

Fonctions Principale :

Détermination de la zone de recherche

```

152 public void findSearchPoints() {
153     //System.out.print("sref1");
154     //this.print(Sref);
155     Solution cachSref=new Solution(Sref.solution,Sref.danceStrenght);
156     int h=0,p;
157     boolean[] s1=new boolean[75],s2=new boolean[75];
158     s2=this.getSref().getSolution().clone();
159     ArrayList<Solution> list=new ArrayList<Solution>();
160     while(list.size()<this.getNbees() & h<this.flip) {
161         s1=s2.clone();
162         p=0;
163         do {
164             s1[this.flip*p+h]=!s2[this.flip*p+h];
165             p++;
166         }while(flip*p+h<this.lenght);
167         Solution s=new Solution(s1);
168         list.add(s);
169         h++;
170     }
171     this.searchArea=list;
172     Sref=cachSref;
173     //System.out.print("sref2");
174     //this.print(Sref);
175 }

```

Assigner les points de départ trouver aux abeilles.

```

176• public void assignSearchPoint() {
177     int i;
178     Interval inter=new Interval(0, searchArea.size());
179     if(searchArea.size()==this.Nbees) {
180         for(i=0;i<searchArea.size();i++)
181         {
182             BeesTab.get(i).setSearchLocal(searchArea.get(i));
183         }
184     }
185     else {
186         if(searchArea.size()<this.Nbees) {
187             for(i=0;i<searchArea.size();i++)
188             {
189                 BeesTab.get(i).setSearchLocal(searchArea.get(i));
190             }
191             for(int j=i;j<this.Nbees;j++)
192             {
193                 BeesTab.get(j).setSearchLocal(searchArea.get((int)inter.getRandom()));
194             }
195         }
196         else {
197             for(int j=0;j<this.Nbees;j++)
198             {
199                 BeesTab.get(j).setSearchLocal(searchArea.get((int)inter.getRandom()));
200             }
201         }
202     }
203 }

```

Calculer fitness et distance :

```

289• public double fitnessFonc(Bee p,Instance inst) {
290     return inst.evaluate(p);
291 }
292• public double fitnessFonc(boolean sref[],Instance inst) {
293     return inst.evaluate(sref);
294 }
295
296• public int distance(boolean x1[],boolean x2[]) {
297     int distance=0;
298     for(int i=0;i<x1.length;i++) {
299         if(x1[i]!=x2[i])
300             distance++;
301     }
302     return distance;
303 }

```

diversity : calcule la diversité d'un solution

```
305 public int diversity(Solution s) {
306     int i=1,min;
307     min=this.distance(s.getSolution(), this.tabooTab.get(0).getSolution());
308     while(i<this.tabooTab.size())
309     {
310         if(min>this.distance(s.getSolution(), this.tabooTab.get(i).getSolution()))
311             min=this.distance(s.getSolution(), this.tabooTab.get(i).getSolution());
312         i++;
313     }
314     return min;
315 }
```

maxDiversityFromDanceTab : renvoie la solution max diversity de DanceTab

```
327 public Solution maxDiversityFromDanceTab() {
328     int i=1,max;
329     boolean[] bool;
330     double fit;
331     max=this.diversity(this.danceTab.get(0));
332     bool=this.danceTab.get(0).getSolution().clone();
333     fit=this.danceTab.get(0).danceStrenght;
334     while(i<this.danceTab.size())
335     {
336         if(max<this.diversity(this.danceTab.get(i))) {
337             max=this.diversity(this.danceTab.get(i));
338             bool=this.danceTab.get(i).getSolution().clone();
339             fit=this.danceTab.get(i).danceStrenght;
340         }
341         i++;
342     }
343     Solution s=new Solution(bool,fit);
344     return s;
345 }
```

selectRef : cette fait la sélection de la solution référence Sref pour une nouvelle itération.

```
public Solution selectSref() {
    double deltaF;
    Solution best=new Solution(danceTab.get(0).getSolution().clone(),danceTab.get(0).getDanceSrenght());
    if(danceTab.size()==1)
        return best;
    else {
        for(int i=1;i<this.getDanceTab().size();i++) {
            if(this.getDanceTab().get(i).getDanceSrenght()>best.getDanceSrenght())
            {
                best.setSolution(danceTab.get(i).getSolution().clone());
                best.setDanceSrenght(danceTab.get(i).getDanceSrenght());
            }
        }
        deltaF=best.getDanceSrenght()-Sref.getDanceSrenght();
        if(deltaF>0) {
            this.setSref(best);
            if(this.nbChances<this.maxChances)
                nbChances=maxChances;
            return best;
        }
        else
        {
            nbChances--;
            if(nbChances>0) {return this.Sref;}
            else {
                if(this.diversity()>this.diversity(this.maxDiversityFromDanceTab()))
                {
                    nbChances=maxChances;
                    return this.Sref;
                }
                else {
                    this.setSref(this.maxDiversityFromDanceTab());
                    nbChances=maxChances;
                    return this.maxDiversityFromDanceTab();
                }
            }
        }
    }
}
```

ResolveSAT (Main):

Chargement des données.

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    //réservé une instance du problème
    Instance instance=new Instance(325 , 75);
    //PATH to the dataset
    String nomF="C:\\Users\\Dell/Desktop/meta hauristique/uuf75-325/UUF75.325.100/uuf75-0100.cnf";
    //String nomF="C:\\Users\\Dell/Desktop/meta hauristique/uf75-325/ai/hoos/Shortcuts/UF75.325.100/uf75-01.cnf";
    //Dans tout les benchmarks uuf75 faut enlevé l'espace du premier tuple pour permetre l input des données
    try{
        //input the benchmark
        InputStream ips=new FileInputStream(nomF);
        InputStreamReader ipsr=new InputStreamReader(ips);
        BufferedReader br=new BufferedReader(ipsr);
        String ligne;
        String mot1="";
        String mot2="";
        String mot3="";
        Literal lit1=null;
        Literal lit2=null;
        Literal lit3=null;
        int i = 0;
        while ((ligne=br.readLine())!=null){
            if(ligne.startsWith("c")||ligne.startsWith("p")||ligne.startsWith("%")||ligne.startsWith("0")){
            }
            else{
                Clause clause=null;
                while((ligne.charAt(i)!=' ')&&(i<ligne.length())){
                    mot1=mot1 + ligne.charAt(i);
                    i++;
                }
                if(Integer.parseInt(mot1)<0)
                    lit1=new Literal(Integer.parseInt(mot1)*-1, true);
                else
                    lit1=new Literal(Integer.parseInt(mot1), false);
                mot1="";
            }
        }
    }
}
```

L'implementation du BSO

```
//-----start BSO processing-----  
//A bee initiates a search from a starting point "Sref"  
Bee beeInit=new Bee(15,instance.getNvar());  
beeInit.BeeInit(instance.getNvar(), instance);  
//put the parameters for BSO  
Bso bso=new Bso(35, 5, 10, 3, 15, instance.getNvar(), beeInit.searchLocal);  
int it=0;  
int i;  
// reiterate the process until maxIter  
while(it<bso.maxIter) {  
    //The Sref is stored in a taboo list  
    bso.getTabooTab().add(bso.getSref());  
    // define search area from Sref  
    bso.findSearchPoints();  
    // Each point find from sref is assigns to a Bee  
    bso.assignSearchPoint();  
    //each bee strat exploration from start point then performs a dance and store it to the dance table  
    for(i=0;i<bso.Nbees;i++) {  
        bso.getBeesTab().get(i).exploration(instance);  
        bso.danceTab.add(bso.getBeesTab().get(i).getSearchLocal());  
    }  
    it++;  
    //the selection of the best value to become the new Sref to reiterate the process  
    bso.selectSref(instance);  
}  
//print results  
bso.print(bso.Sref);  
System.out.println("the fitness of the solution obtained is : "+ bso.fitnessFonc(bso.Sref.solution, instance));  
System.out.println("is it the solution ? "+ instance.ifSolution(bso.Sref.solution));
```


4.2. PseudoCode:

1- Génération d'une solution aléatoire qui va représenter la solution référence et l'ajouter à la liste Taboo

2- Génération de k Abeilles dont chacune va effectuer une recherche locale.

3- Enregistrement des k solutions générées par les abeilles dans la table DANCE ;

4- Evaluation des solutions :

Si une des solutions de la table DANCE satisfait les 325 clauses, alors cette solution est la solution optimale.

Sinon la solution de la table DANCE, qui satisfait le plus grand nombre de clauses, est choisie

 Si elle n'existe pas dans la liste taboo :

 On l'ajoute à cette liste taboo

 MaxChance = 3

 Sinon

 Si la solution choisie n'existe pas dans la liste taboo et MaxChance > 0

 Ajouter cette solution à la liste taboo et décrémenter MaxChance

 Sinon

 Générer une solution aléatoire S, qui soit meilleure en termes de diversité

 MaxChances = 3

 Ajouter S à la liste taboo et la prendre comme référence.

 FSi

 FSi

FSi

Répéter le processus à partir de l'étape 2, jusqu'à atteindre le nombre maximum d'itérations fixé ou l'obtention de la solution optimale.

Etude expérimentale et résultats :

L'algorithme BSO dépend de 3 paramètres :

- ✓ Nombre Abeilles
- ✓ Flip
- ✓ Max chance

Nous avons fait varier les paramètres empiriques, et tester sur l'instance 1, afin de trouver les valeurs permettant de trouver les meilleurs résultats.

Le tableau suivant montre les résultats de quelques tests :

Nbee	Flip	Max Chance	MaxSat
25	3	3	310
25	3	5	311
25	3	10	309
15	5	3	308
15	5	5	311
15	5	10	310
10	10	3	312
10	10	5	311
10	10	10	310

Les Configuration :

maxIter = 35; *** Nombre d'itération ***

Flip = 10;

Nbees = 10;

maxChances = 3;

locIter = 15 ;

On a utilisé la Benchmarks : uuf75-325

4.3 Résultats:

Les Résultats :

Instance	Max Clause	Temps(s)	Taux de Satisfiabilité %
1	406	12,526	93
2	408	10,250	94
3	409	34,456	94,5
4	406	15,356	93
5	408	22,555	94
6	407	12,547	93,5
7	409	31,144	94,5
8	408	34,124	94
9	406	11,623	93
10	405	12,364	92,5
Moyen	407,2	19,695	93,6

Résultats détaillés pour Max-SAT BanchMark UF100-418

Instance	Max Clause	Temps(s)	Taux de Satisfiabilité %
1	211	3,5423	97
2	211	5,2554	97

3	214	11,244	98
4	212	6,2551	97,2
5	212	7,3334	97,2
6	211	5,254	97
7	213	8,2554	97,7
8	214	6,286	98
9	213	9,2214	97,7
10	211	8,35	97
Moyen	212,2	7,0997	97,38

Résultats détaillés pour Max-SAT BanchMark UF50-218

Instance	Max Clause	Temps(s)	Taux de Satisfiabilité %
1	91	2,14	100
2	91	1,20	100
3	91	1,52	100
4	91	1,20	100
5	91	1,40	100
6	91	1,452	100
7	91	1,281	100
8	90	1,354	98
9	91	1,321	100
10	91	1,254	100

Moyen	90,9	1,412	99,6
--------------	------	-------	------

Résultats détaillés pour Max-SAT BanchMark UF20-91

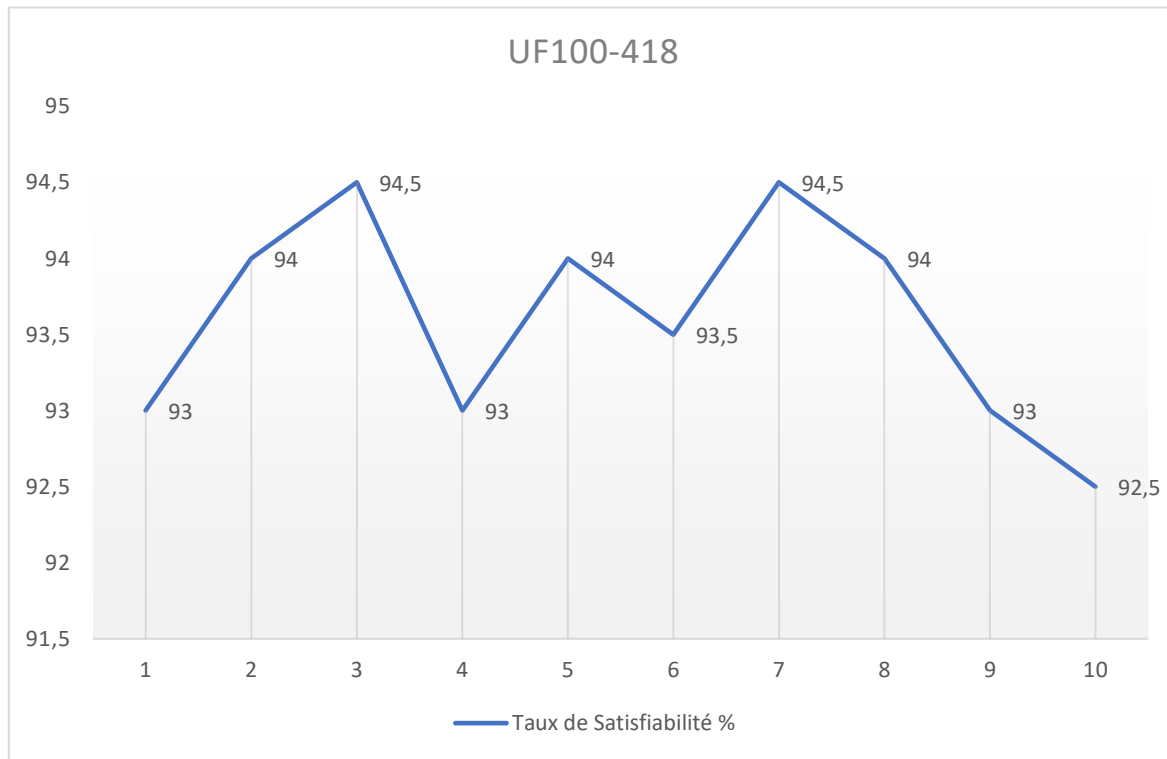


Figure 1: Histogramme montrant les résultats du test de l'algorithme BSO pour le problème SAT sur les fichiers de Benchmarks UF-100

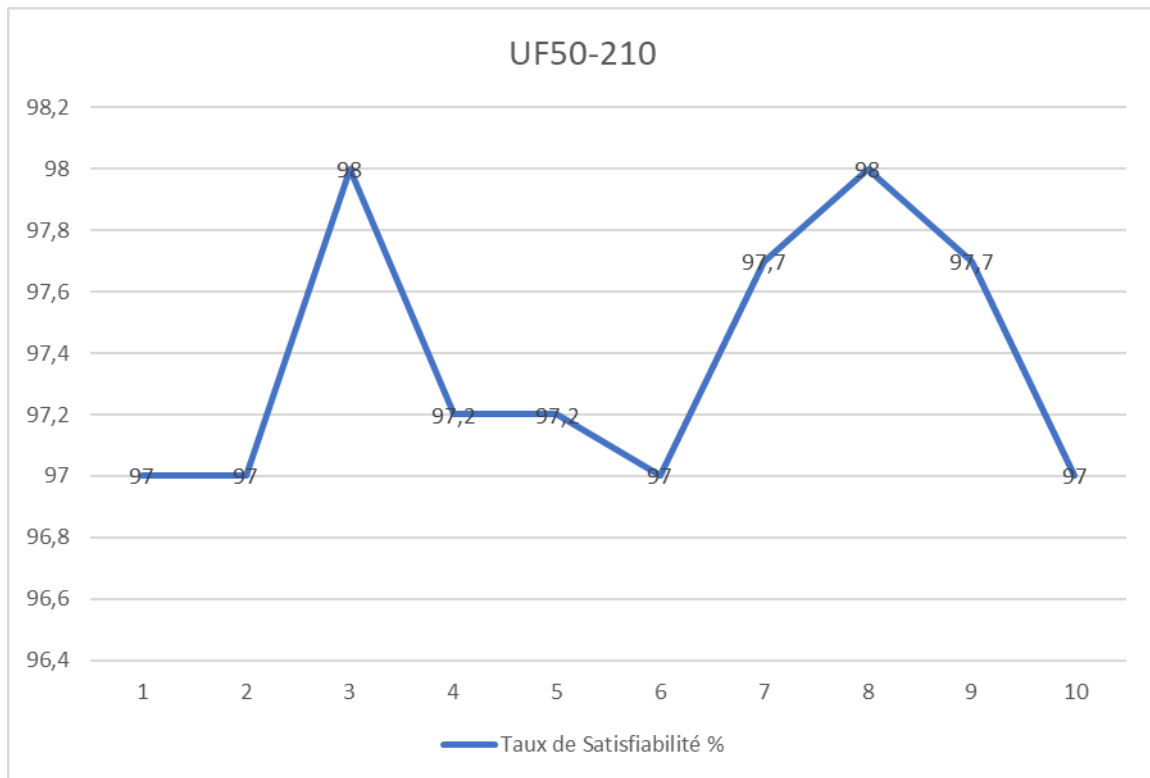


Figure 2: Histogramme montrant les résultats du test de l'algorithme BSO pour le problème SAT sur les fichiers de Benchmarks UF-50

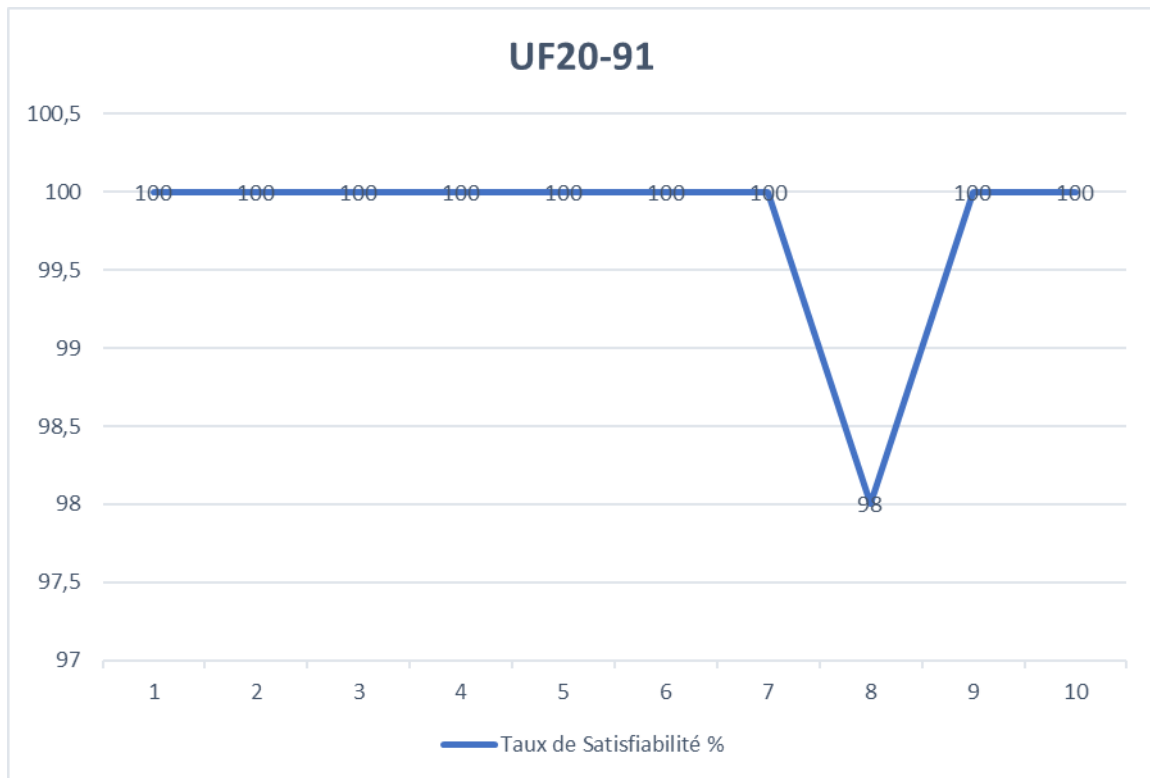


Figure 2: Histogramme montrant les résultats du test de l'algorithme BSO pour le problème SAT sur les fichiers de Benchmarks UF-20

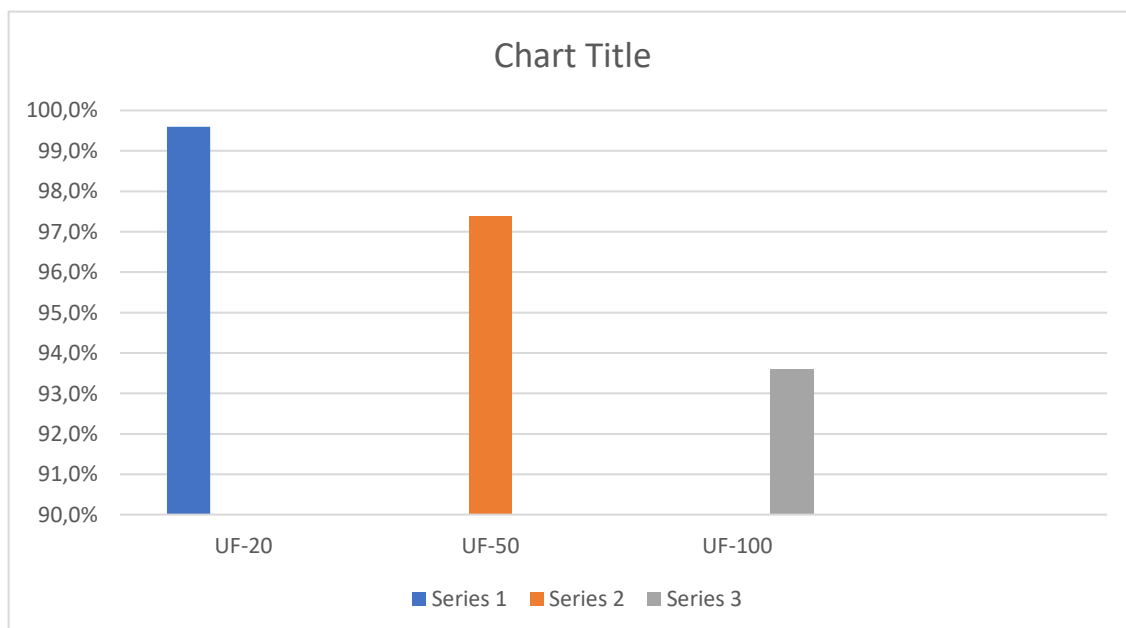
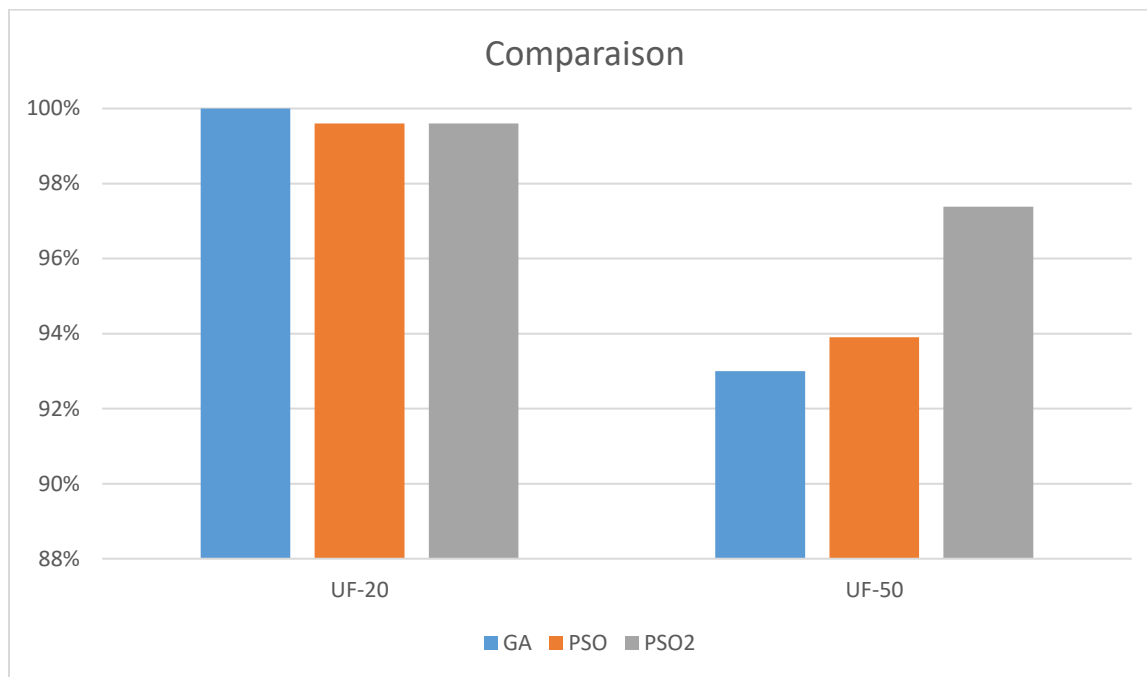


FIGURE 3 – Taux de satisfiabilité selon la nature de l'instance

Analyse : L'exécution de l'algorithme de résolution de SAT basé sur la métaheuristique « BSO » a donné des résultats satisfaisants au bout de 30 à 60 secondes. L'algorithme n'a pas atteint la solution optimale mais il a pu satisfaire jusqu'à 97% des clauses.

5- Comparaison :



Commentaires : On remarque que l'Algorithme Génétique est optimal pour l'instance uf-20, et BSO la plus Optimal pour l'instance UF-50.

⇒ La différence est certes minime mais ceci nous laisse croire que BSO est légèrement plus performant sur le problème max-SAT qu'PSO/GA. Ceci bien sûr dans les limites de nos moyens, et suivant les paramètres fixés.


6- Conclusion générale

Arrivé à la fin de ce projet, et après beaucoup de temps passé à apprendre, modifier et tester les différents algorithmes vus en cours, nous pensons avoir achevé un travail que nous jugeons assez complet, nous avons exploré différents aspects de la résolution de problème, en partant des méthodes basiques aux méthode plus avancées, nous pouvons résumer notre travail aux points suivants :

- De nouvelles méthodes ont fait leur apparition, sacrifiant le désir de trouver une solution exacte (ou optimale) qui peut prendre un temps inconcevable pour être déterminée, au profit de solutions, certes moins optimales mais qui demeurent une alternative raisonnable.
- Malgré le côté aléatoire et probabiliste des nouvelles approches méta-heuristique, leur façon de fonctionner en fait une représentation fidèle de la vie réelle en générale.

Talk about the good sides of BSO and its bad sides, talk about the potential that resides in the metaheuristics and swarm intelligence

Quelque Capture dans L'app:



SAT

LOSAT

Genetic Algorithm (GA)

File

Clauses Number :

Literals Number :

Literal 1

Literal 2

Literal 3

Population Size

Crossover Rate

Mutation Rate

Number Of Iteration

Start Resolution

100

20

80

5000

SAT

LOSAT

Particle Swarm Optimization (P...

File

Clauses Number :

Literals Number :

Litteral 1

Litteral 2

Litteral 3

Start Resolution

Particle Number

500

Constant 1

1, 4

Constant 2

1, 4

Inner Weight

30

Number Of Iteration

10000

Max Volacity

20

SAT

LOSAT

Bee Swarm Optimization (BSO)

File

Clauses Number :

Literals Number :

Littoral 1

Littoral 2

Littoral 3

Flip

5

Maximum Chance

3

Number Of Bees

10

Number Of Local Search

15

Number Of Iteration

5000

Start Resolution

SAT

LOSAT

Genetic Algorithm (GA)

C:\Users\zakaria\Desktop\uf20-03.cnf

File

Clauses Number :

91

Literals Number :

20

Litteral 1	Litteral 2	Litteral 3
-9	3	-15
-12	-4	-15
6	14	-17
10	16	11
-15	20	-7
-1	10	16
13	17	-7
-2	-14	-13
10	6	15

Start Resolution

100

20

80

5000

Message

Founded In : 2209ms

Solution : [1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0]

Fitness: 96.7032967032967 %

OK

Mutation Rate

Number Of Iteration

7. Références:

[1] J. Kennedy et R. Eberhart, « Particle swarm optimization », , *IEEE International Conference on Neural Networks, 1995. Proceedings*, vol. 4, novembre 1995, p. 1942–1948 vol.4

Particle Swarm Optimization PSO Prof. Habiba Drias.

Genetic Algorithmme GA Prof. Habiba Drias.

Bee Swarm Optimization BSO Prof. Habiba Drias.