PARTICLE SWARM OPTIMISATION – BRIGHTEST POINT IN A IMAGE WITH A GIVEN HUE

## Descrierea problemei considerate

Ne dorim sa folosim algoritmul Particle Swarm Optimisation pentru aflarea celui mai luminos punct dintr-o imagine data cu o anumita nuanta. Uneori avem nevoie de o aproximatie a sursei luminii dintr-o imagine , pentru editarea acesteia sau generarea unei scene . Astfel ne propunem sa construim o aplicatie care sa poata afla sursa unei lumini. De asemenea folosind un hue custom , putem afla cel mai luminat punct intr-o imagine , de exemplu pe o suprafata .

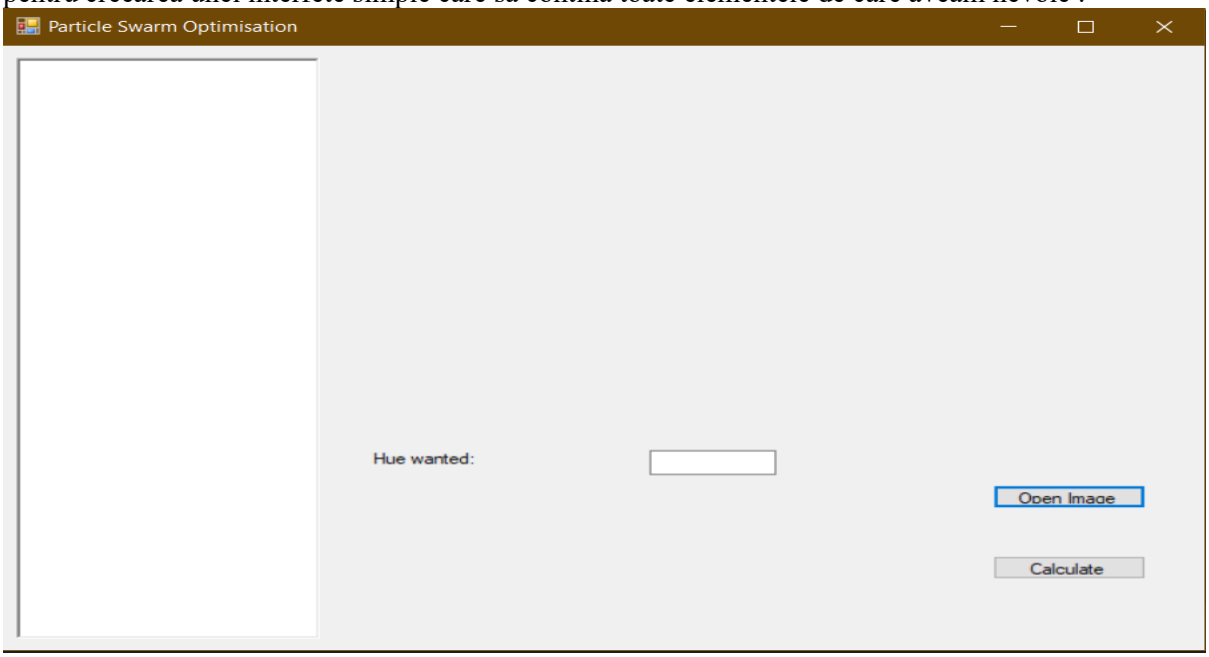
## Aspecte teoretice privind algoritmul

Algoritmul Particle Swarm Optimisation este o metoda care optimizeaza o problema prin iteratii incercand sa imbunatateasca solutia candidat . Problema este rezolvata avand o multime de solutii numite particule si miscand aceste particule prin domeniul problemei folosindu-se de o formula matematica simpla folosind pozitia particulei si velocitatea acesteia. Fiecare particula este influentata de cea mai buna pozitie a ei dar este si ghidata spre cea mai buna pozitie cunoscuta in domeniu , pozitie updatata de celelalte particule.

Parametrii algoritmului pot influenta extrem de mult performanta acestuia. Numarul de particule (cluster) , velocitatea maxima (maxvel) ,numarul de iteratii (iter) , inertia si doi parametri pentru prioritate spre personal best(c1) si prioritate spre global best(c2);

## Modalitatea de rezolvare

Ne-am hotarat sa folosim C# pentru implementarea acestei aplicatii. Am folosit Windows Forms pentru creearea unei interfete simple care sa contina toate elementele de care aveam nevoie .




Folosim un textbox pentru a afisa rezultatul , un alt textbox pentru a introduce hue-ul cautat si doua butoane. Unul pentru a gasi imaginea dorita si butonul Calculate care apeleaza efectiv functiile algoritmului PSO.

De asemenea am folosit Bitmap pentru stocarea imaginilor citite . Apoi am folosit o functie de schimbare a datelor in RGB ale pixelilor in HSV pentru a le prelucra.

Am folosit Photoshop pentru a crea imaginile de test.

🕐 **Listarea părților semnificative din codul sursă însoțite de explicații și comentarii**

```
4 references
public  class Problema
{
    int rezolutieX, rezolutieY;

    public double searchedHue;

    1 reference
    public Problema(int rezolutiex, int rezolutiey, double searchedhue)
    {
        rezolutieX = rezolutiex;
        rezolutieY = rezolutiey;
        searchedHue = searchedhue;
    }
    2 references
    public double FunctieObiectiv(double hue, double saturation, double value)
    {
        if (hue > searchedHue + 10 || hue < searchedHue - 10)
            return 0;

        return (0.2*saturation)+(0.8*value);
    }
```

Clasa problema unde se afla functia obiectiv si constrangerile problemei.

In functia obiectiv se da o culoare(hue) tinta si se returneaza cel mai luminos punct apartinand acelei culori. Se da mai multa importanta luminiozitatii decat saturatiei.

Rezolutia imaginii reprezinta domeniul de lucru.

```
4 references
public class Parametri
{
    int clusterSize;
    int parameters;
    double maxVelocity;
    double inertia;
    double c1, c2;
    int iteratii;

    1 reference
    public Parametri(int cluster,int param,double maxvel,int iter)
    {
        c1 = 1;
        c2 = 2;
        inertia =0.4;
        clusterSize = cluster;
        parameters = param;
        maxVelocity = maxvel;
        iteratii = iter;
    }
```

Clasa parametrilor algoritmului Particle Swarm.

ClusterSize reprezinta numarul de particule, parameters – numarul de parametrii de intrare(hue,saturation,value), maxVelocity-viteza maxima.

```
10 references
public class Particle
{
    public int positionX,positionY;
    public double velocityX,velocityY;
    public double cost;
    public Particle best;
}
```

Clasa Particle, contine pozitia pe imagine prin x,y, viteza pe cele 2 dimenisuni, variabila pt functia obiectiv si cea mai buna pozitie a particulei.

```
1 reference
public static Particle PSO(Problema problema,Parametri parametri)
{
    //Initializare
    List<Particle> roi = new List<Particle>(parametri.ClusterSize());


    int xmin, xmax, ymin, ymax;
    xmin = problema.Xmin();
    xmax = problema.Xmax();
    ymin = problema.Ymin();
    ymax = problema.Ymax();

    for (int i = 0; i < parametri.ClusterSize(); ++i)
    {
        Particle P = new Particle();

        P.positionX = rand.Next(xmin, xmax);////
        P.positionY = rand.Next(ymin, ymax);////

        Form1.mask.SetPixel(P.positionX, P.positionY, Color.GreenYellow);

        HSVColor pixel = pixels[(P.positionX * problema.Xmax()) + P.positionY];



        P.cost = problema.FunctieObiectiv(pixel.Hue, pixel.Saturation, pixel.Value);
        P.velocityX = 1;
        P.velocityY = 1;
        P.best = P;

        roi.Add(P);

    }
```

Partea de initializare a algoritmului particle swarm.

Initializeaza clusterSize numar de particule pe pozitii(pixeli) aleatorii din imagine si calculeaza functia de fitness pt fiecare punct, apoi particulele sunt adaugate in lista roiului.

```
    //Program
Particle optimSocial = roi[0];
foreach (Particle p in roi)
{
        if (p.cost > optimSocial.cost)
        {
            optimSocial = p;
            Console.WriteLine("X: " + optimSocial.positionX + " Y:" + optimSocial.positionY + " values: " + pixels[optimSocial.positionX * problema.Xmax() + optimSocial.positionY].sa
        }
}
for (int i = 0; i < parametri.Iteratii(); ++i)
{
    foreach (Particle p in roi)
    {
        double r1 = rand.NextDouble();
        double r2 = rand.NextDouble();
        p.velocityX = parametri.Inertia() * p.velocityX + parametri.C1() * r1 * (p.best.positionX - p.positionX) + parametri.C2() * r2 * (optimSocial.positionX - p.positionX);
        p.velocityY = parametri.Inertia() * p.velocityY + parametri.C1() * r1 * (p.best.positionY - p.positionY) + parametri.C2() * r2 * (optimSocial.positionY - p.positionY);




        if (p.velocityX > parametri.MaxVelocity())
            p.velocityX = parametri.MaxVelocity();
        if (p.velocityX < -parametri.MaxVelocity())
            p.velocityX = -parametri.MaxVelocity();

        if (p.velocityY > parametri.MaxVelocity())
            p.velocityY = parametri.MaxVelocity();
        if (p.velocityY < -parametri.MaxVelocity())
            p.velocityY = -parametri.MaxVelocity();

        p.positionX = (int)(p.positionX + p.velocityX);
        p.positionY = (int)(p.positionY + p.velocityY);
```

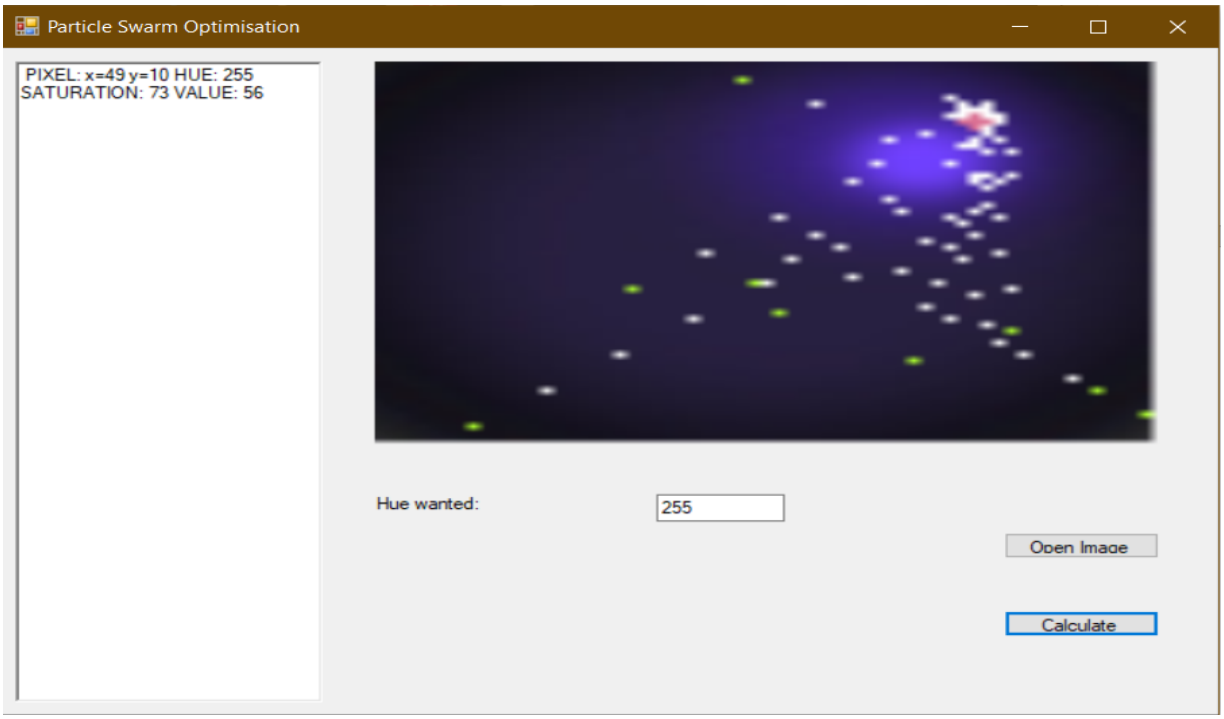
Partea de determinare a urmatoarei pozitii a particulelor in functie de personal-best si social-best si incadrearea sa in domeniu.

```
    Form1.mask.SetPixel(p.positionX, p.positionY, Color.White);

    HSVColor pixel= pixels[(p.positionX * problema.Xmax()) + p.positionY]; ;
    p.cost = problema.FunctieObiectiv(pixel.Hue, pixel.Saturation, pixel.Value);

    if (p.cost > p.best.cost)
    {
        p.best = p;
            if (p.cost > optimSocial.cost)
            {
                optimSocial = p;
                Console.WriteLine("X: " + optimSocial.positionX + " Y:" + optimSocial.positior
            }
        }

}
```

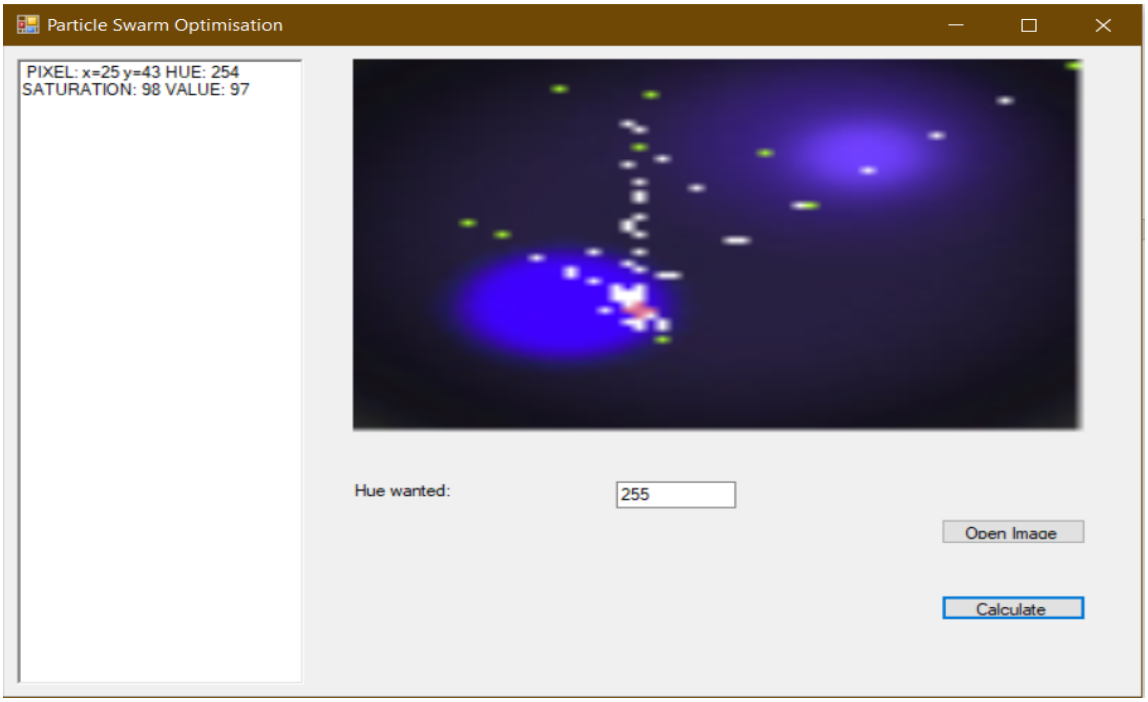
Actualizarea personal-bestului si social-bestului, dupa aceasta portiune de cod urmeaza sfarsitul algoritmului si returnarea rezultatului (optimSocial).

```
//calculam matricea(imaginea cu valorile HSV) ce trebuie trimisa la PSO
for (int i = 0; i < bmp.Height; i++)
{
    for (int j = 0; j < bmp.Width; j++)
    {
        double R = bmp.GetPixel(i, j).R;

        Color original = Color.FromArgb(bmp.GetPixel(i, j).R, bmp.GetPixel(i, j).G, bmp.GetPixel(i, j).B);

        //calculam HSV
        HSVColor hsv = GetHSV(original);
        //richTextBox1.Text +="Pixel: "+i+" "+j+" "+ hsv.hue + " " + hsv.saturation + " " + hsv.value + '\n';
        if (hsv.Saturation > maxS)
            maxS = hsv.Saturation;
        if (hsv.Value > maxV)
            maxV = hsv.Value;
        //updatam matricea
        Particles.Particle_Swarm_Optimisation.pixels[(i*bmp.Width)+j] = hsv;
    }

}
```

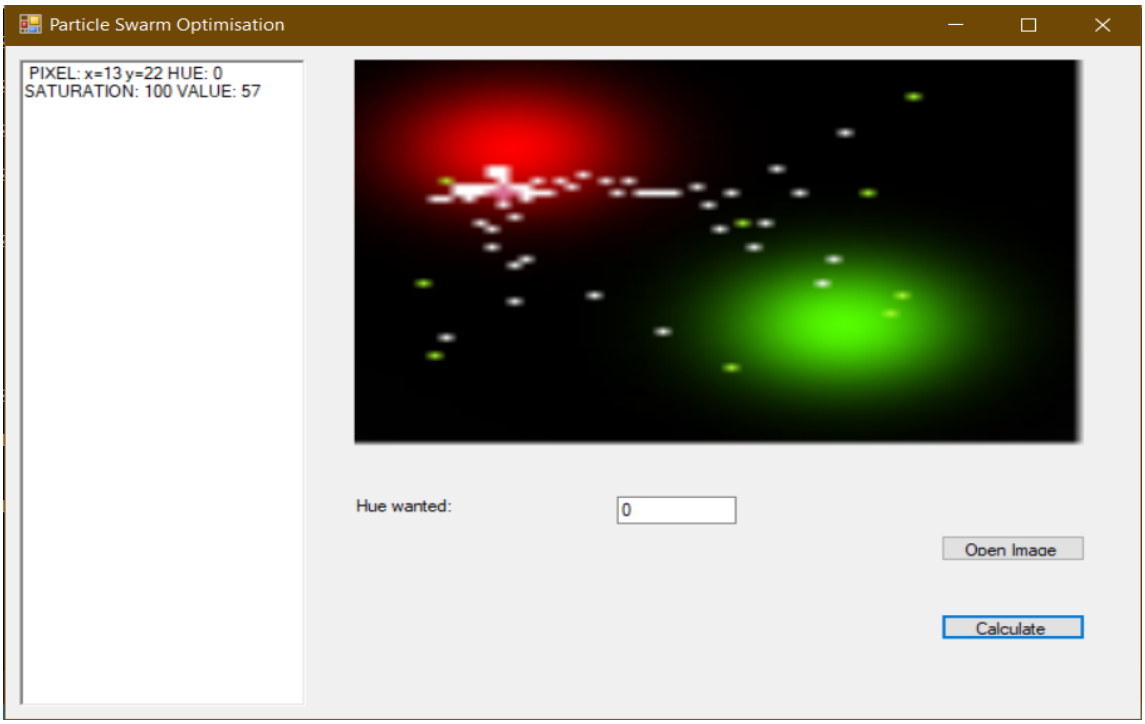
**🕐 Rezultatele obținute prin rularea programului în diverse situații, capturi ecran și comentarii asupra rezultatelor obținute**




O rulare a algoritmului pentru hue de 255(albastru) pe o imagine cu majoritar hue 0 (negru) . Particulele converg intr-o zona apropiata . Pixelii verzi sunt coordonatele initiale a particulelor , pixelii albi sunt pozitiile prin care s-a trecut iar crucea rosie este generata in jurul pixelului rezultat.




O rulare a algoritmului pentru un hue (255) cu puncte cu saturation si value mari (100 100)




O rulare a algoritmului pentru o imagine cu 2 puncte de culori diferite

## 🕐 Concluzii

Algoritmul Particle Swarm Gbest gaseste solutii pentru problemele problemele propuse in timp util (aproape instantaneu) dar nu este 100% precis iar calitatea solutiilor NU se imbunatateste cu fiecare rulare.

## 🕐 Bibliografie

https://en.wikipedia.org/wiki/Particle_swarm_optimization

https://en.wikipedia.org/wiki/HSL_and_HSV

https://stackoverflow.com/questions/359612/how-to-convert-rgb-color-to-hsv

https://edu.tuiasi.ro/pluginfile.php/49569/mod_resource/content/6/IA05_Optimizare2.pdf

## 🕐 Impartirea Taskurilor

Adrian Teohari 1407A – codul algoritmului PSO , reglare parametri , realizare documentatie,

Luca Razvan 1407A – design interfata , citire imagine , generarea matricii de pixeli in HSV, realizare documentatie, creare imagini test