

Optimizarea mașinii de curse pe baza compromisului
dintre obiective
Algoritmul NSGA-II

Proiect realizat de :
Palade Gabriel
Cunievici Petru-Sebastian

1. Descrierea problemei considerate

Problema abordată în acest proiect constă în **optimizarea sistemului de control al unei mașini autonome** care trebuie să navigheze într-un mediu cu obstacole variabile. Provocarea centrală este gestionarea **compromisului** critic dintre **performanță** (viteză de deplasare) și **siguranță** (evitarea coliziunilor).

Obiectivul: Obiectivul principal este identificarea unei **Frontiere Pareto** de soluții de control, reprezentate prin ponderile genetice W1 (acelerație) și W2 (frânare).

Provocarea: Cea mai semnificativă dificultate tehnică a constat în **calibrarea funcției de fitness** pentru a preveni comportamentele degenerate și supra-învățarea.

În primele iterații, am întâmpinat dificultăți în definirea penalizării pentru coliziune:

Dacă penalizarea era prea mică, algoritmul favoriza mașini care mergeau cu viteză maximă și ignorau coliziunile (acceptând o mică scădere de scor pentru o distanță parcursă foarte bună).

Dacă penalizarea era prea mare, multe mașini erau prea prudente, alegând să rămână pe loc sau să meargă cu viteza minimă pentru a nu risca nimic.

2. Aspecte teoretice privind algoritmul

Pentru rezolvarea acestei probleme s-a utilizat conceptul de optimizare multiobiectiv, unde se ridică ideea maximizării sau minimizării unor obiective contradictorii, idee specifică algoritmului NSGA-II.

Conceptul de bază

Algoritmul NSGA-II este un algoritm genetic multiobiectiv, rapid, elitist și cu sortare nedominantă ce se bazează pe conceptul de front Pareto. Astfel, algoritmul poate fi structurat ca o meta-euristică, utilizându-se un algoritm evolutiv clasic, la care se adaugă calcularea frontului Pareto.

Frontul Pareto

În optimizarea multi-obiectiv, **Frontul Pareto** cuprinde totalitatea soluțiilor care sunt **non-dominante**. O soluție este considerată non-dominată dacă nu există o altă variantă în populație care să fie mai bună la toate criteriile simultan. Astfel, din punct de vedere matematic, o soluție A domină o soluție B dacă sunt îndeplinite două condiții implementate în codul nostru:

- **Condiția de non-inferioritate:** A nu este mai slab decât B la niciun obiectiv.
- **Condiția de superioritate:** A este strict mai bun decât B la cel puțin un obiectiv.

Dacă nicio altă soluție nu îl domină pe A, atunci A aparține **Rank-ului 1** și face parte din Frontul Pareto (în ordine crescătoare de la cel mai bun).

La finalul algoritmului, acesta va conține diverse profiluri de „personalitate” ale mașini (extrema stânga reprezentată de un obiectiv, iar dreapta de cel opus).

Distanța de aglomerare

Spre deosebire de un algoritm genetic (evolutiv) clasic, NSGA-II se remarcă prin elitism (reunirea populației părinților cu a copiilor și alegerea celor mai buni indivizi) și prin sortarea bazată pe distanța de aglomerare.

Distanța de aglomerare este o măsură a densității soluțiilor din jurul unui individ specific într-un front Pareto. În timp ce **Rangul Pareto** decide calitatea soluției (cât de aproape este de optim), **Crowding Distance** decide cât de „unică” este acea soluție.

Rolul său principal este de a asigura diversitatea genetică, permițând algoritmului să exploreze simultan atât comportamente extreme (vitezomanul și prudentul), cât și comportamente intermediare.

Complexitatea algoritmului

Complexitatea de timp a algoritmului NSGA-II este dată de funcția de Fast Non-Dominated Sort, ce face parte din implementarea ideii de sortare a fronturilor Pareto, funcția comparând fiecare individ cu toți ceilalți indivizi din populația combinată de dimensiune $2N$. Există două bucle for care compară $2N \times 2N$ elemente (fiecare individ din populație cu fiecare), comparând M obiective (în cazul de față 2) în funcția Dominates. Astfel, întrucât aceasta funcție este cea mai costisitoare, complexitatea algoritmului este $O(M \times N^2)$.

3. Modalitatea de rezolvare

Implementarea a fost realizată în limbajul C#, utilizând ca și biblioteci principale Linq și System.Forms (pentru manipulare structuri de date respectiv pentru interfața grafică).

Metoda de rezolvare se bazează pe implementarea algoritmului NSGA-II, o tehnică de optimizare euristică inspirată din selecția naturală, special concepută pentru probleme cu mai multe obiective contradictorii. Structura generală a abordării este următoarea:

1. Reprezentarea și Codificarea
2. Evaluarea Performanței (Funcția de Fitness)
3. Mecanismul NSGA-II: Ierarhizarea și Diversitatea (Sortarea Non-Dominantă și Distanța de Aglomerare)

4. Evoluția și Selecția (Turneul și operatorii genetici clasici)

4. Listarea părților semnificative din codul sursă însoțite de explicații și comentarii

Reprezentarea genetica

```
CSharp
public class Chromosome
{
    public int NoGenes { get; set; } // Numarul de gene ale individului
    public double[] Genes { get; set; } // Valorile genelor (W1, W2
    pentru robot)
    public double[] MinValues { get; set; } // Limitele inferioare ale
    genelor
    public double[] MaxValues { get; set; } // Limitele superioare ale
    genelor

    //NSGA II

    //tablou obiective
    public double[] Objectives { get; set; }

    // rang Pareto, cu 1 cel mai bun
    public int Rank { get; set; }

    // diversitate pt evitare aglomerare/valori similare->alg se duce in
    2 directii difertie
    public double CrowdingDistance { get; set; }

    private static Random _rand = new Random();

    public Chromosome(int noGenes, double[] minValues, double[]
    maxValues)
    {
        NoGenes = noGenes;
        Genes = new double[noGenes];
        MinValues = (double[])minValues.Clone();
        MaxValues = (double[])maxValues.Clone();

        Objectives = new double[2];
    }
}
```



```

        for (int i = 0; i < noGenes; i++)
        {
            // Initializare aleatorie uniforma in domeniul specificat
            Genes[i] = minValues[i] + _rand.NextDouble() * (maxValues[i]
- minValues[i]);
        }
    }

    // Constructor de copiere - Critic pentru elitismul NSGA-II
    (populatia combinata 2N)
    public Chromosome(Chromosome c)
    {
        NoGenes = c.NoGenes;
        Rank = c.Rank;
        CrowdingDistance = c.CrowdingDistance;

        Genes = (double[])c.Genes.Clone();
        MinValues = (double[])c.MinValues.Clone();
        MaxValues = (double[])c.MaxValues.Clone();
        Objectives = (double[])c.Objectives.Clone();
    }
}

```

Această clasă definește structura unei „mașini” și stochează toate datele necesare pentru algoritmul NSGA-II. Fără constructorul de copiere care folosește .Clone(), modificările aduse unui „copil” prin mutație ar schimba direct genele „părintelui” din generația anterioară, distrugând procesul evolutiv.

Definirea Conflictului Pareto (Funcția Compute Fitness)

```

CSharp
public void ComputeFitness(Chromosome c)
{
    double totalDist = 0;
    double minSafety = double.MaxValue;

    // Folosim un set de obstacole pentru evaluare
    for (int i = 0; i < 20; i++)
    {
        double obstacle = 5.0 + _r.NextDouble() * 18.0;

        // NOUA FORMULA: Velocitatea foloseste ambele gene
    }
}

```



```

        // W1 (Genes[0]) impinge robotul inainte proportional cu
distanța
        // W2 (Genes[1]) il franeaza invers proportional cu distanta
        double velocity = (c.Genes[0] * obstacle) - (c.Genes[1] /
obstacle); //cele doua gene sunt invers proportionale

        // Limitare fizica: impiedicam robotul sa mearga cu spatele
sau sa mearga prea repede
        velocity = Math.Max(0.5, Math.Min(10, velocity));

        double gap = obstacle - velocity;

        // Verificam coliziunea (izbirea)
        if (gap < 0)
        {
            // Penalizare drastica pentru coliziune
            totalDist -= 100;
            minSafety = -10;
            break;
        }

        totalDist += velocity; //daca velocitatea e mare inseamna ca
nu a franat mult->inseamna ca a parcurs mai repede traseul
        if (gap < minSafety)
            minSafety = gap; //daca distanta la fiecare iteratie de
obstacol e mare, inseamna ca a franat din timp
    }

    // Setarea obiectivelor pentru NSGA-II
    c.Objectives[0] = -totalDist; // Obiectiv 1: Maximizare
Performanta
    c.Objectives[1] = -minSafety; // Obiectiv 2: Maximizare
Siguranta robotii foarte prudenti (caz extrem front pareto) vor fi genetic
creati sa

        // franeze foarte din timp
inaintea unui obstacol datorita raportului dintre w2 si dits.obstacol
    }

```

Funcția de fitness generează la fiecare apel 20 de iterații diferite (obstacole diferite), cele două gene fiind invers proporționale în formula vitezei. Astfel, viteza crește proporțional cu produsul dintre gena 1 și obstacol (distanța), o valoare mare a acesteia reprezentând un scor bun pentru timpul sau viteza de parcurgere. Gena 2 scade valoarea prin raportul cu obstacolul, un scor bun pentru aceasta fiind reprezentat de un start de frânare înainte de obstacol (daca viteza lasă un spațiu mare până la obstacol, minSafety

primește prima valoare reală și apoi continuă să scadă doar dacă întâlnește un spațiu și mai mic, adică mai periculos). Practic, al doilea obiectiv primește ca scor cel mai mic gap, adică cel mai periculos moment.

Sortarea Non-Dominantă

```
CSharp
private List<List<Chromosome>> FastNonDominatedSort(List<Chromosome>
population)
{
    // Logica de sortare Pareto pe ranguri
    int n = population.Count; // cati roboti in total in 2N
    var fronts = new List<List<Chromosome>> { new List<Chromosome>()
}; // lista de fronturi (lista de liste)
    int[] dominationCount = new int[n]; // pt fiecare robot cati il
domina
    List<int>[] dominatedSet = new List<int>[n]; // pt fiecare robot
pe cati alti domina acesta

    for (int i = 0; i < n; i++) // compara fiecare robot cu toti
ceilalti
    {
        dominatedSet[i] = new List<int>(); // lista de roboti mai
slabi decat robotul i
        for (int j = 0; j < n; j++)
        {
            if (Dominates(population[i], population[j]))
                dominatedSet[i].Add(j);
            else if (Dominates(population[j], population[i]))
                dominationCount[i]++;
        }

        // dominationCount == 0 ar insemna ca nimeni nu e mai bun
ca el => rank=1, primul front Pareto
        if (dominationCount[i] == 0)
        {
            population[i].Rank = 1;
            fronts[0].Add(population[i]);
        }
    }

    int k = 0;
    while (fronts[k].Count > 0)
    {
        var nextFront = new List<Chromosome>();

        foreach (var dominator in fronts[k]) // luam fiecare individ
din front
```



```

    {
        int pIdx = population.IndexOf(dominator); //fiecare
        individ din front si comparam cu parent (index dominator)
        foreach (int qIdx in dominatedSet[pIdx])
        {
            dominationCount[qIdx]--; //vedem daca dominatorul
            curent e singurul care il bate
            if (dominationCount[qIdx] == 0) //daca da il pun in
            urmatorul front
            {
                population[qIdx].Rank = k + 2; //fronts[0]
                nextFront.Add(population[qIdx]);
            }
        }
        k++;
        fronts.Add(nextFront);
    }
    return fronts;
}

```

Este elementul central al NSGA-II. Permite păstrarea simultană a extremelor în Rank 1, deoarece niciunul nu îl domină pe celălalt (fiecare e mai bun la un alt obiectiv).

Selecția Turneu

CSharp

```

public static Chromosome Tournament(List<Chromosome> population)
{
    Chromosome a = population[_rand.Next(population.Count)];
    Chromosome b = population[_rand.Next(population.Count)];

    if (a.Rank < b.Rank)
        return new Chromosome(a); //apel constructor copiere, in caz
    ca voi folosi mai departe acel copil sa nu modific indivizii din generatia
    anterioara
    if (b.Rank < a.Rank)
        return new Chromosome(b); //apel constructor copiere
}

```



```

        return a.CrowdingDistance > b.CrowdingDistance ? new
Chromosome(a) : new Chromosome(b);
    }

```

Decide cine are dreptul să se reproducă, folosind rangul ca prioritate și distanța de aglomerare ca departajare. Se asigura presiunea de selecție, alegându-se mașinile într-o manieră dispersată, prin funcția de distanță de aglomerare.

Funcția Distanță de Aglomerare (Crowding Distance)

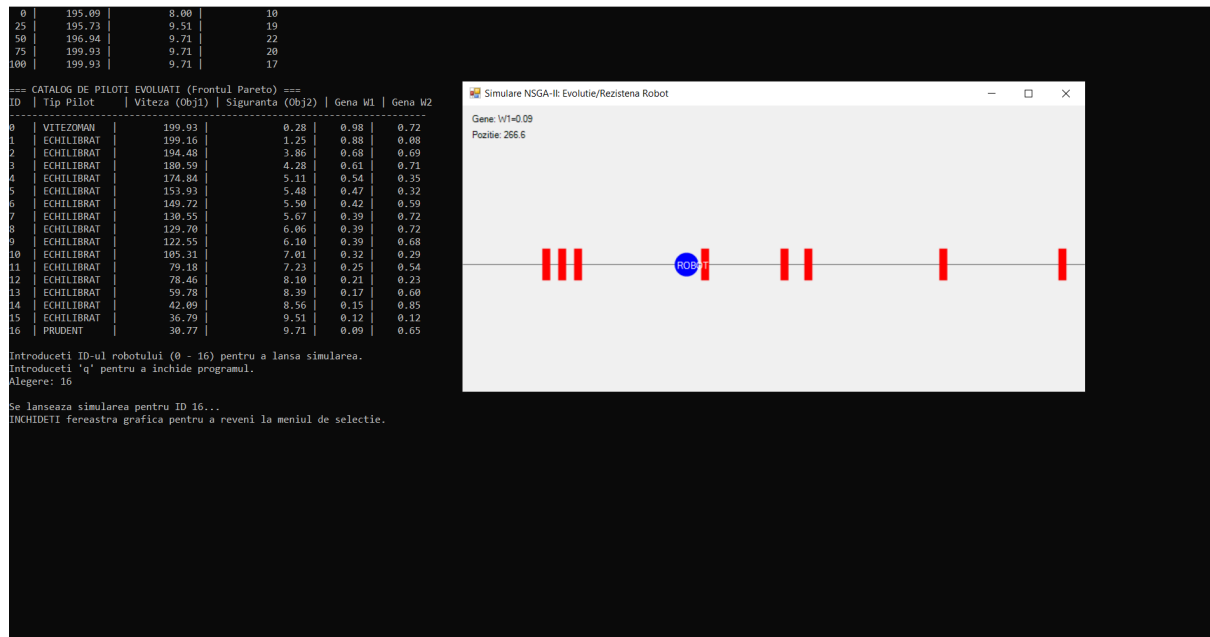
```

CSharp
private void CalculateCrowdingDistance(List<Chromosome> front)
{
    int n = front.Count;
    if (n < 3) { front.ForEach(c => c.CrowdingDistance = 1e10);
return; }
    front.ForEach(c => c.CrowdingDistance = 0);
    for (int m = 0; m < 2; m++)
    {
        var sorted = front.OrderBy(c => c.Objectives[m]).ToList();
        sorted[0].CrowdingDistance = 1e10;
        sorted[n - 1].CrowdingDistance = 1e10;
        double range = sorted[n - 1].Objectives[m] -
sorted[0].Objectives[m];
        if (range > 0)
            for (int i = 1; i < n - 1; i++)
                sorted[i].CrowdingDistance += (sorted[i +
1].Objectives[m] - sorted[i - 1].Objectives[m]) / range;
    }
}

```

Această metodă evaluează cât de "izolat" este un individ față de vecinii săi din același front. Cu cât distanța este mai mare, cu atât individul este mai valoros pentru diversitatea populației. Cu ajutorul ei se menține o variație a genelor indivizilor prin protejarea extremelor, folosindu-se distanța Manhattan normalizată (normalizată deoarece valorile obiectivelor pot fi de amplitudini diferite, cele mai mari având prioritate, caz ce nu este în conformitate cu algoritmul).

5. Rezultatele obținute prin rularea programului în diverse situații, capturi ecran și comentarii asupra rezultatelor obținute



Aplicația afișează un tabel ce rezumă evoluția populației la un interval de 25 de generații. De asemenea, se afișează frontul Pareto de rang 1, ce conține mașinile care au supraviețuit procesului de selecție naturală și care sunt considerați **optimi**. Acestea sunt mașinile care au ajuns la extrema performanței: nu mai poți găsi o altă mașină care să fie și mai rapidă și mai sigură decât ele în același timp. De asemenea, se observă și mașini cu scoruri bune pentru ambele obiective.

Din meniu, se poate alegea vizualizarea pe un traseu fix a oricărei mașini din frontul Pareto optim, cu diferențe vizibile între extreme.

6. Concluzii

Implementarea algoritmului NSGA-II a demonstrat că optimizarea multi-obiectiv este superioară celei mono-obiectiv în contextul învățării automate a mașinilor de curse. Algoritmul nu a oferit o singură soluție "perfectă", ci o frontieră de posibilități (Frontul Pareto), permițând vizualizarea clară a conflictului dintre performanța brută (viteză) și integritatea fizică (siguranță).

7.Bibliografie

https://sci2s.ugr.es/sites/default/files/files/Teaching/OtherPostGraduateCourses/Metaheuristics/Deb_NSGAII.pdf

https://en.wikipedia.org/wiki/Multi-objective_optimization

<https://www.geeksforgeeks.org/deep-learning/non-dominated-sorting-genetic-algorithm-2-ns-ga-ii/>

Cod sursă laborator algoritmi genetici

8.Contribuții

Palade Gabriel - Contribuții algoritm multi-obiectiv NSGA-II, interfață grafică, funcții turneu și elitism, propunere problemă multi-obiectiv pentru optimizare

Cunievici Petru-Sebastian - Contribuții algoritm multi-obiectiv NSGA-II, funcție fitness, funcții genetice, gestionare date finale (front Pareto)