Optimizarea maşinii 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 maşini**, din punct de vedere al vitezei medii și al indicelui de agresivitate al șoferului. Provocarea centrală este gestionarea **compromisului** critic dintre **performanță** (viteză de deplasare) și **consum** (cantitatea de combustibil consumata).

Obiectivul: Obiectivul principal este identificarea unei **Frontiere Pareto** de soluții de control, reprezentate prin ponderile genetice W1 (viteza medie) și W2 (indice de agresivitate).

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

În primele iterații, am întâmpinat dificultăți în definirea penalizării pentru indicele de agresivitate:

Am utilizat funcții exponențiale pentru penalizarea vitezei medii excesive și al agresivitatii în calculul consumului de combustibil.

Parametrii funcțiilor exponențiale oferă ponderea penalizarii vitezei și agresivitatii.

Dacă penalizarea era prea mică, algoritmul favoriza maşini care mergeau cu viteză maximă, respectiv maşini care erau conduse mai agresiv.

Dacă penalizarea era prea mare, multe maşini 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 ridica ideea maximizării sau minimizării unor obiective contradictorii, idee specifica algoritmului NSGA-II.

Conceptul de bază

Algoritmul NSGA-II este un algoritm genetic multiobiectiv, rapid, elitist şi cu sortare nedominantă ce se bazează pe conceptul de front Pareto. Asftel, 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-dominated**. O soluție este considerată non-dominantă 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 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).

Distanţa de agolmerare

Spre deosebire de un algoritm genetic (evolutiv) clasic, NSGA-II se remarcă prin elitism (reunirea populaţiei părinţilor cu a copiilor şi alegerea celor mai buni indivizi) şi prin sortarea bazată pe distanţa 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 algortimului NSGA-II este data de funcţia de Fast Non-Dominated Sort, ce face parte din implementarea ideei de sortare a fronturilor Pareto, funcţia comparând fiecare individ cu toţi ceilalţi indivizi din populaţia combinată de dimensiune 2N. Există două bucle for care compară 2Nx2N elemente (fiecare individ din populaţie cu fiecare), comparând M obiective (în cazul de faţă 2) în funcţia Dominates. Astfel, întrucât aceasta funcţie este cea mai costisitoare, complexitatea algoritmului este O(MxN2).

3. Modalitatea de rezolvare

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

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

1. Reprezentarea și Codificarea

2. Evaluarea Performanţei (Funcţia de Fitness)

3. Mecanismul NSGA-II: Ierarhizarea și Diversitatea (Sortarea Non-Dominantă şi Distanţa de Aglomerare)

4. Evoluţia şi Selecţia (Turneul şi 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 „maşini” 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 (Funcţia Compute Fitness

CSharp

```
public void ComputeFitness(Chromosome c)
{
    double distance = 100; // distanta in kilometri

    c.Objectives[0] = distance / c.Genes[0];

    double optimal_fuel_performance = 15; // 15km / l
    double optimal_average_speed = 80; // 80 km/h
    double warm_up_distance = 5; // 5 km, timpul necesar incalzirii
motorului
    double alfa = 0.00005; // parametrii pentru functiile exponentiale
    double beta = 0.25;

    // Calculate penalty factors
    double speedPenalty = Math.Exp(-alfa * Math.Pow(c.Genes[0] -
optimal_average_speed, 2));
    double aggressivenessPenalty = Math.Exp(-beta * c.Genes[1]);
    double warmUpFactor = 1 - Math.Exp(-distance / warm_up_distance);
```

```
        // Effective fuel efficiency
        double effectiveEta = optimal_fuel_performance * speedPenalty *
    aggressivenessPenalty * warmUpFactor;

        // Safety check to avoid division by zero
        if (effectiveEta < 0.1) effectiveEta = 0.1;

        // Fuel consumed (objective)
        c.Objectives[1] = distance / effectiveEta;

     }
```

Funcția de fitness evaluează un cromozom calculând două obiective care trebuie minimizate: durata drumului și consumul de combustibil pentru o distanță fixă. Durata este obținută ca raport între distanță și viteza medie (prima genă), astfel încât viteze mai mari duc la timp mai mic. Consumul este calculat pornind de la o eficiență ideală a motorului (15 km/l), care este redusă prin factori exponențiali ce modelează efectele reale ale condusului: abaterea de la viteza optimă de 80 km/h (penalizare pentru viteză prea mică sau prea mare), stilul de condus agresiv (a doua genă) și pierderile cauzate de timpul necesar încălzirii motorului la începutul drumului. Acești factori sunt combinați într-o eficiență efectivă, iar consumul final este distanța împărțită la această eficiență, cu o limită minimă impusă pentru a evita probleme numerice.

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 inseamna 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]
contine robotii de rank1
                            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).

Selecţia 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 selecţie, alegându-se maşinile într-o manieră dispersată, prin funcţia de distanţă de aglomerare.

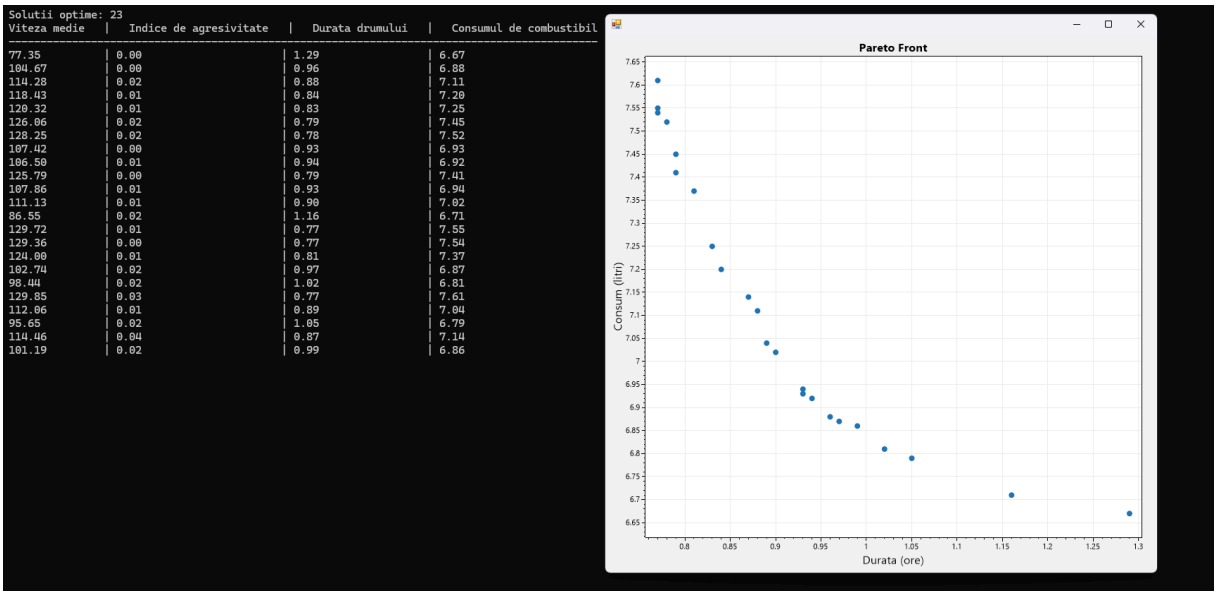
Funcţia Distanţă de Aglomerare (Crowding Distance)

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 menţine o variaţie a genelor indivizilor prin protejarea extremelor, folosindu-se distanţa Manhattan normalizată (normalizată deoarece valorile obiecitvelor 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 cu proprietățile indivizilor de pe frontul Pareto de rang 1, ce contine mașinile care au supraviețuit procesului de selectie naturala și care sunt considerați **optimi**. Acestea sunt maşinile care au ajuns la extrema performanței: nu mai poți găsi o altă maşină care să fie și mai rapidă și să aibă un consum mai scăzut decât ele în același timp. De asemenea, se observă şi maşini cu scoruri bune pentru ambele obiective.

De asemenea, este afișat un grafic cu doua axe in care apar timpul parcurgerii distanței (axa X) și consumul calculat în litri de combustibil. Graficul a fost creat utilizând biblioteca ScottPlot.

6. Concluzii

Implementarea algoritmului NSGA-II a demonstrat că optimizarea multi-obiectiv este superioară celei mono-obiectiv în contextul învăţării automate a maşinilor 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.Bibiliografie

https://sci2s.ugr.es/sites/default/files/files/Teaching/OtherPostGraduateCourses/Metaheuristicas/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/
https://sci2s.ugr.es/sites/default/files/files/Teaching/OtherPostGraduateCourses/Metaheuristicas/Deb_NSGAII.pdf
Cod sursă laborator algoritmi genetici

8.Contribuţii

Palade Gabriel - Contribuţii algoritm multi-obiectiv NSGA-II, interfaţă grafică, funcţii turneu şi elitism, propunere problemă multi-obiectiv pentru optimizare
Cunievici Petru-Sebastian - Contribuţii algoritm multi-obiectiv NSGA-II, funcţie fitness, funcţii genetice, gestionare date finale (front Pareto)