Algoritmul NSGA-II pentru optimizare multi-obiectiv

Proiect IA 2025-2026 ~ Negoiță Petru și Chiuariu Silviu-Vasile

1. Descrierea problemei

Pentru aplicarea algoritmului NSGA-II, am preluat o problemă recurentă în contextul optimizării automobilelor în funcție de variabilele dorite de echipa de proiectare.

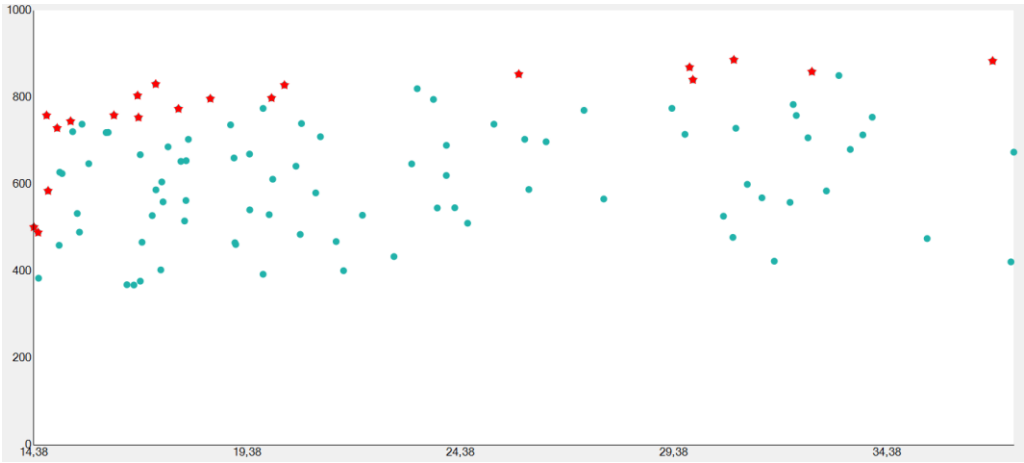
În scenariul curent, se dorește **optimizarea autonomiei împreună cu puterea de accelerație a mașinii până la 100km/h**. Pentru a obține acest lucru, se iau în considerare trei variabile: *puterea motorului, capacitatea rezervorului* și *dimensiunea roților*. Însă, prin stabilirea în mod repetat a mai multor seturi de valori/dimensiuni, se observă compromisuri între cele două componente de optimizat, astfel că nu există o ”cea mai bună opțiune” (de exemplu, pentru a crește autonomia, va fi nevoie de un rezervor mai mare, ceea ce înseamnă o creștere în greutate, deci și o creștere a distanței de accelerație). Astfel, se dorește găsirea celor mai bune ”*n*” modele de mașini ce au raportul autonomie/putere accelerație cel mai optim**.**

Pentru rezolvarea problemei, se aplică algoritmul NSGA-II, ce îndeplinește fix acest lucru.

2. Algoritmul NSGA-II

Aspecte teoretice

NSGA (Non-dominated Sorting Genetic Algorithm II) este un algoritm evolutiv ce este folosit în rezolvarea problemelor de optimizare multi-obiectiv. Acesta se bazează pe stabilirea **Frontului Pareto**, ce reprezintă cele mai bune ”*n*” soluții dintr-o populație (Fig. 1).



(Fig. 1) Frontul Pareto (roșu) – cei mai buni 20 de indivizi dintr-o populație de 100

Implementarea algoritmului este alcătuită din următorii pași (Fig. 2):

1. **Stabilirea variabilelor inițiale:**
   - Mărimea populației (Pt);
   - Membrii populației (Pt) ce sunt aleși în mod aleatoriu;
   - Numărul de generații de copii (Qt) create pe baza populației din iterația curentă;
   - Rata de mutație a genelor ce o pot suferi copiii populației curente;
   - Numărul de rezultate dorite;
   - Parametrii în care se încadrează valorile genelor (în contextul problemei, avem 3 gene: *puterea motorului, capacitatea rezervorului* și *dimensiunea roților*).

1. **Crearea populației de copii (Qt) din populația curentă (Pt) și îmbinarea celor 2 populații într-una singură (Rt)**
   - Se aplică algoritmul evolutiv specific (descris mai jos) pentru a genera o populație de copii Qt de aceeași dimensiune cu populație părinte Pt

   - Printr-un **Turneu**, se stabilesc părinții a doi copii
     I. Pentru stabilirea mamei și a tatălui, se preiau aleatoriu câte 2 membri ai populației Pt
     II. Se compară între ei *fitness-ul* (în cazul nostru, se compară rang-ul și *crowding distance*-ul, termeni ce vor fi explicați pe parcurs)
     III. Cel care îl are cel mai bun devine părinte

   - Prin **Crossover**, se creează doi copii rezultați din combinarea genelor celor doi părinți
     I. Aceasta se întâmplă cu o *rată aleatorie*:
     genă_nouă = *rata* * genă_părinte1 + *rata* * genă_părinte2
     II. Procesul de repetă pentru toate genele, generând în final doi copii

   - Copiii pot suferi o **Mutație** a genelor la un moment aleatoriu
     I. Acest moment aleatoriu poate avea loc dacă după **Crossover**, *o rată generată randomizat* este mai mică decât *rata de mutație*, definită la începutul algoritmului
   - Acești pași se repetă până când populația Qt ajunge de dimensiunea populației Pt
   - Se formează populația Rt

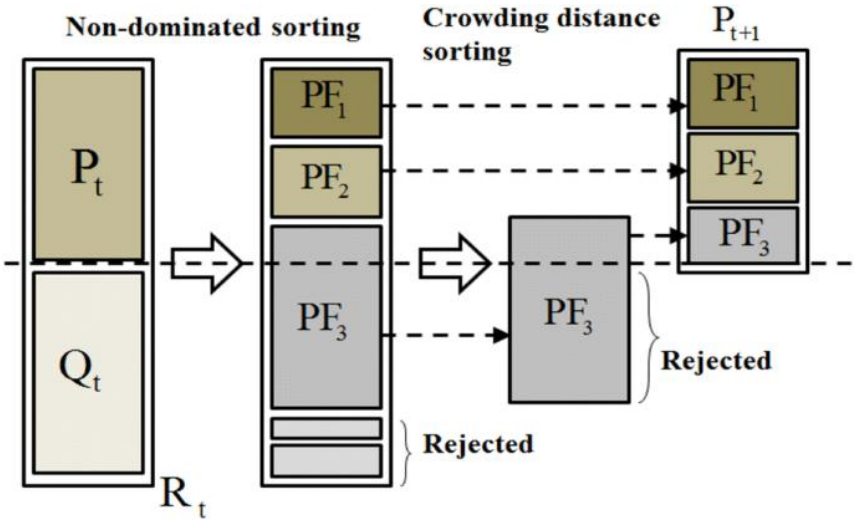
2. **Se aplică algoritmul *Non-dominated sorting***
   - Se parcurg membrii populației Rt iterativ
   - Pentru fiecare membru, se caută în populația Rt **numărul de indivizi care îl domină (*domination_count)* și indivizii pe care îi domină (*dominates[]*).** Un individ A este dominant față de un individ B dacă are valorile cele mai mici ale variabilelor optimizabile (în problema noastră, *puterea de accelerație* și *autonomia*). În funcție de rezultatul comparației, se incrementează una din valorile de interes

- **Se împarte populația în *Fronturi***
  I. Se creează primul front. Acesta este format din populația Rt cu *domination_count* egal cu 0
  II. Pentru compunerea următorului front se iterează prin primul front populația Rt (fără elementele din frontul anterior). Dacă un individ din Rt apare în lista *dominates[]* al unui individ din frontul anterior, i se decrementează *domination_count*-ul. La finalul iterărilor, frontul curent va fi alcătuit din membrii ce au în final *domination_count* egal cu 0
  III. Crearea de fronturi în această manieră se repetă până când vom rămâne fără populație nealocată vreunui front
  IV. Fiecare individ la avea în variabila **rang** valoarea frontului din care face parte

3. Se aplică algoritmul *Crowding distance sorting*

- Deoarece pentru bucla de program următoare avem nevoie de o populație Pt+1 de dimensiune Pt, o vom alcătui din primele cele mai optime fronturi
- Pentru că mai mult ca sigur ultimul front de integrat în Pt+1 nu va încăpea în întregime și pentru a obține o gamă mai largă de rezultate finale pe **Frontul Pareto**, se preiau indivizii din fiecare front în funcție de *distanța maximă* dintre membrii vecini.

4. Se repetă pașii 1-3 până când se atinge numărul maxim de generații dorite



(Fig. 2) Algoritmul NSGA-II – bucla principală

3. Rezolvarea problemei

Pentru a implementa algoritmul NSGA-II în contextul problemei date, am proiectat în ***Visual Studio 2022*** o aplicație cu UI în limbajul C#. Programul este alcătuit deci din **codul de interfață și algoritm.**

1. Generare populație inițială

- Se generează aleatoriu populația inițială (pe baza obiectului de tip **Random** *rand*
- Se calculează în funcție de variabilele inițiale *puterea de accelerație* și *autonomia*
- Se generează graficul populației și se afișează toți indivizii acesteia

```
// ============================================================
// generare populatie

for (int i = 0; i < num_populatie; i++)
{
    resultListBox.Items.Clear();

    // generare
    double motor = minMotor + (rand.NextDouble() * (maxMotor - minMotor));
    double rezervor = minRezervor + (rand.NextDouble() * (maxRezervor - minRezervor));
    double roti = minRoti + (rand.NextDouble() * (maxRoti - minRoti));

    // formule
    double acceleratie = (3500 / motor) + (rezervor / 60) + (roti / 10);
    double range = (rezervor * 12) - (motor / 3) - (roti * 2) + rand.Next(-15, 15);

    // salvare in populatie
    populatie[i] = new Individ(motor, rezervor, roti, acceleratie, range);

    // desenare Punct
    DataPoint punct = new DataPoint(acceleratie, range);
    string info = $"SOLUTIA #{i + 1}\n" +
    $"Motor: {populatie[i].putere_motor:F0} CP\n" +
    $"Rezervor: {populatie[i].capacitate_rezervor:F0} L\n" +
    $"Roti: {populatie[i].dimensiune_roti:F0}\n" +
    $"Accel: {populatie[i].acceleratie:F2} s\n" +
    $"Range: {populatie[i].autonomie_range:F0} km";
    punct.ToolTip = info;
    chartPareto.Series[0].Points.Add(punct);

}
```

```
labelStatus.Text = "Gata!";
labelStatus.ForeColor = Color.Green;

string membriPopulatie = "";

for (int i = 0; i < num_populatie; i++)
{
    membriPopulatie = membriPopulatie + $"Motor: {populatie[i].putere_motor:F0} CP; Rezervor: {populatie[i].capacitate_rezervor:F0} L; Roti: {populatie[i].dimensiune_roti:F0};" +
        $"Accel: {populatie[i].acceleratie:F2} s; Range: {populatie[i].autonomie_range:F0} km ;" + "\r\n";
}
Cursor = Cursors.Default;
ShowScrollableMessage("Lista Membri Populatie", membriPopulatie);
```

2. Implementarea algoritmului NSGA-II

Clasa *Individ*

- implementarea obiectului *Individ*, ce alcătuiește populația

```
public class Individ
{
    public double putere_motor;   // CP
    public double capacitate_rezervor; // litri
    public double dimensiune_roti; // inch
    public double acceleratie; // secunde
    public double autonomie_range;    // km
    public int rank; // rangul in populatie
    public double crowding_distance; // distanta de aglomerare
    public int domination_count; // numarul de dominari
    public List<Individ> dominates; // solutii dominate

    2 references
    public Individ()
    {
        putere_motor = 0;
        capacitate_rezervor = 0;
        dimensiune_roti = 0;
        acceleratie = 0;
        autonomie_range = 0;
        rank = 0;
        crowding_distance = 0;
        domination_count = 0;
        dominates = new List<Individ>();
    }
    1 reference
    public Individ(double motor, double rezervor, double roti, double acc, double range)
    {
        putere_motor = motor;
        capacitate_rezervor = rezervor;
        dimensiune_roti = roti;
        acceleratie = acc;
        autonomie_range = range;
        rank = 0;
        crowding_distance = 0;
        domination_count = 0;
        dominates = new List<Individ>();
    }
```

- implementarea modulelor pentru generarea copiilor
- implementarea modulului de generare Fronturi

```
2 references
public static Individ TournamentSelection(Individ[] population)...
1 reference
public static Individ[] Crossover(Individ parent1, Individ parent2)...
2 references
public static void Mutation(Individ child, double mutationRate)...
1 reference
public static List<Individ> buildFront(List<Individ> lastFront, Individ[] population)...
```

Bucla principală

- generare populație copii Qt și formarea populației Rt

```
for (int gen = 0; gen < generatii; gen++)
{
    // Creare generatie noua
    Individ[] populatie_copii = new Individ[num_populatie];
    for (int i = 0; i < num_populatie / 2; i++)
    {
        // Tournament Selection
        // fitness-ul este in functie de rang si crowding
        Individ mother = Individ.TournamentSelection(populatie);
        Individ father = Individ.TournamentSelection(populatie);

        // Crossover
        Individ[] children = Individ.Crossover(mother, father);

        // Mutatie
        Individ.Mutation(children[0], (double)rata_mutatie);
        Individ.Mutation(children[1], (double)rata_mutatie);

        // Adaugare copii in noua generatie
        populatie_copii[2 * i] = children[0];
        populatie_copii[2 * i + 1] = children[1];
    }

    // Combinare populatii parinti + copii
    Individ[] populatie_totala = new Individ[num_populatie * 2];
    Array.Copy(populatie, populatie_totala, num_populatie);
    Array.Copy(populatie_copii, 0, populatie_totala, num_populatie, num_populatie);
```

- non-dominated sorting

```
// non-dominated sorting
for (int i = 0; i < num_populatie * 2; i++)
{
    for (int j = 0; j < num_populatie * 2; j++)
    {
        if (i != j)
        {
            // verificare dominare
            if ((populatie_totala[i].acceleratie < populatie_totala[j].acceleratie && populatie_totala[i].autonomie_range > populatie_totala[j].autonomie_ran
                (populatie_totala[i].acceleratie <= populatie_totala[j].acceleratie && populatie_totala[i].autonomie_range > populatie_totala[j].autonomie_ran
                (populatie_totala[i].acceleratie < populatie_totala[j].acceleratie && populatie_totala[i].autonomie_range >= populatie_totala[j].autonomie_ran
            {
                populatie_totala[i].dominates.Add(populatie_totala[j]);
            }
            else if ((populatie_totala[j].acceleratie < populatie_totala[i].acceleratie && populatie_totala[j].autonomie_range > populatie_totala[i].autonomi
                    (populatie_totala[j].acceleratie <= populatie_totala[i].acceleratie && populatie_totala[j].autonomie_range > populatie_totala[i].autonomi
                    (populatie_totala[j].acceleratie < populatie_totala[i].acceleratie && populatie_totala[j].autonomie_range >= populatie_totala[i].autonomi
            {
                populatie_totala[i].domination_count++;
            }
        }
    }


}

// creare fronturi
List<List<Individ>> fronturi = new List<List<Individ>>();
```

- creare fronturi (cu hardcodare de 40 pentru a putea începăpea în populații ~200 indivizi

```
// creare fronturi
List<List<Individ>> fronturi = new List<List<Individ>>();

// primul front
List<Individ> F1 = new List<Individ>();
for (int i = 0; i < num_populatie; i++)
{
    if (populatie_totala[i].domination_count == 0)
    {
        populatie_totala[i].rank = 1;
        F1.Add(populatie_totala[i]);
    }
}
fronturi.Add(F1);


// hardcodare max 40 fronturi
for (int i=1;i<40;i++)
{
    fronturi.Add(Individ.buildFront(fronturi[i - 1], populatie_totala));
}
```

- calculare crowding distance și determinarea valorilor pentru Pt+1

```
// calculare crowding distance
foreach (var front in fronturi)
{
    int frontSize = front.Count;
    if (frontSize > 0)
    {
        // initializare distanta
        for (int i = 0; i < frontSize; i++)
        {
            front[i].crowding_distance = 0;
        }
        // sortare dupa acceleratie
        front.Sort((a, b) => a.acceleratie.CompareTo(b.acceleratie));
        front[0].crowding_distance = double.MaxValue;
        front[frontSize - 1].crowding_distance = double.MaxValue;
        double accel_min = front[0].acceleratie;
        double accel_max = front[frontSize - 1].acceleratie;
        for (int i = 1; i < frontSize - 1; i++)
        {
            front[i].crowding_distance += (front[i + 1].acceleratie - front[i - 1].acceleratie) / (accel_max - accel_min);
        }
        // sortare dupa range
        front.Sort((a, b) => a.autonomie_range.CompareTo(b.autonomie_range));
        front[0].crowding_distance = double.MaxValue;
        front[frontSize - 1].crowding_distance = double.MaxValue;
        double range_min = front[0].autonomie_range;
        double range_max = front[frontSize - 1].autonomie_range;
        for (int i = 1; i < frontSize - 1; i++)
        {
            front[i].crowding_distance += (front[i + 1].autonomie_range - front[i - 1].autonomie_range) / (range_max - range_min);
        }
    }
}
populatie = new Individ[num_populatie];
int count = 0;
foreach (var front in fronturi)
{
    if (count + front.Count <= num_populatie)
    {
```

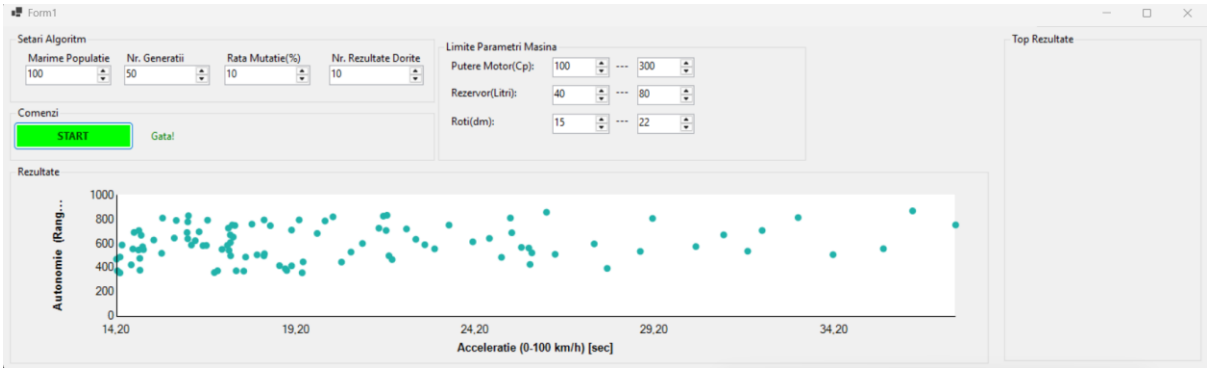
```
        int count = 0;
        foreach (var front in fronturi)
        {
            if (count + front.Count <= num_populatie)
            {
                // adaugare intreg front
                foreach (var indiv in front)
                {
                    populatie[count] = indiv;
                    count++;
                }
            }
            else
            {
                // sortare dupa crowding distance descrescator
                front.Sort((a, b) => b.crowding_distance.CompareTo(a.crowding_distance));
                int remaining = num_populatie - count;
                for (int i = 0; i < remaining; i++)
                {
                    populatie[count] = front[i];
                    count++;
                }
                break; // populatia este completa
            }
        }

    }
```

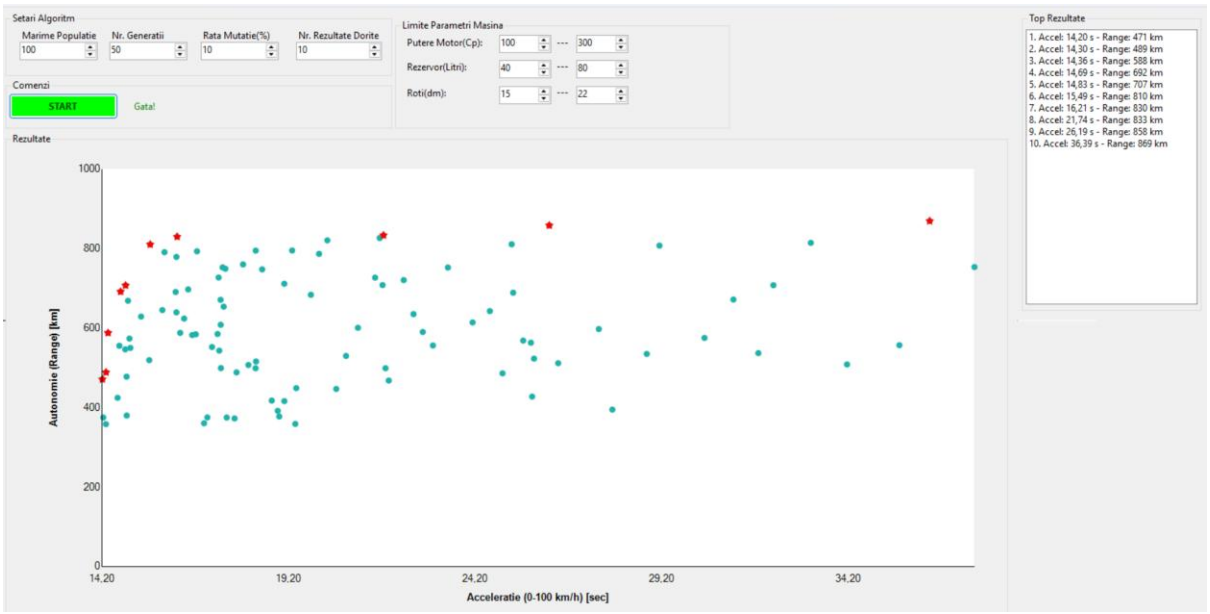
3. Afișarea rezultatelor

```
// afisare rezultate
resultListBox.Visible = true;
for (int i = 0; i < (int)numRezultateDorite.Value; i++)
{

    DataPoint punct = new DataPoint(populatie[i].acceleratie, populatie[i].autonomie_range);
    string info = $"SOLUTIA OPTIMA #{i + 1}\n" +
      $"Motor: {populatie[i].putere_motor:F0} CP\n" +
      $"Rezervor: {populatie[i].capacitate_rezervor:F0} L\n" +
      $"Roti: {populatie[i].dimensiune_roti:F0}\n" +
      $"Accel: {populatie[i].acceleratie:F2} s\n" +
      $"Range: {populatie[i].autonomie_range:F0} km";

    punct.ToolTip = info;
    punct.MarkerStyle = MarkerStyle.Star5;
    punct.MarkerSize = 12;
    punct.Color = Color.Red;

    chartPareto.Series[0].Points.Add(punct);
    resultListBox.Items.Add(resultListBox.Items.Count + 1 + ". Accel: " + populatie[i].acceleratie.ToString("F2") + " s - Range: " + populatie[i].autonomie_range.ToString("F0") + " km");
}
```

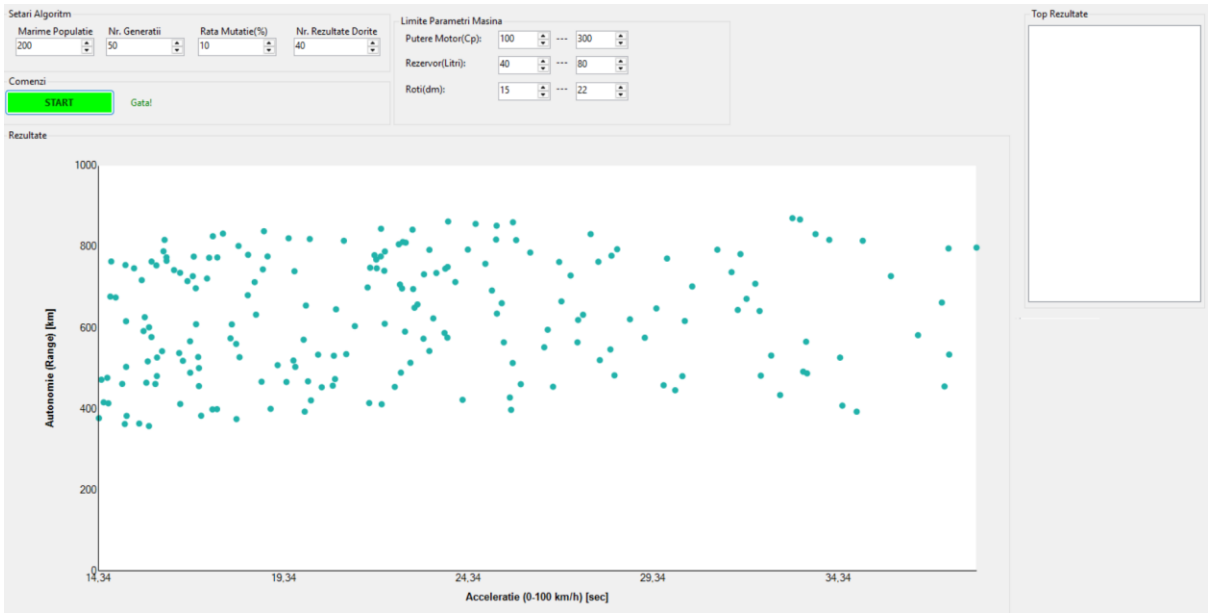
4. Rezultate - cazuri (Frontul Pareto – stelele roșii)

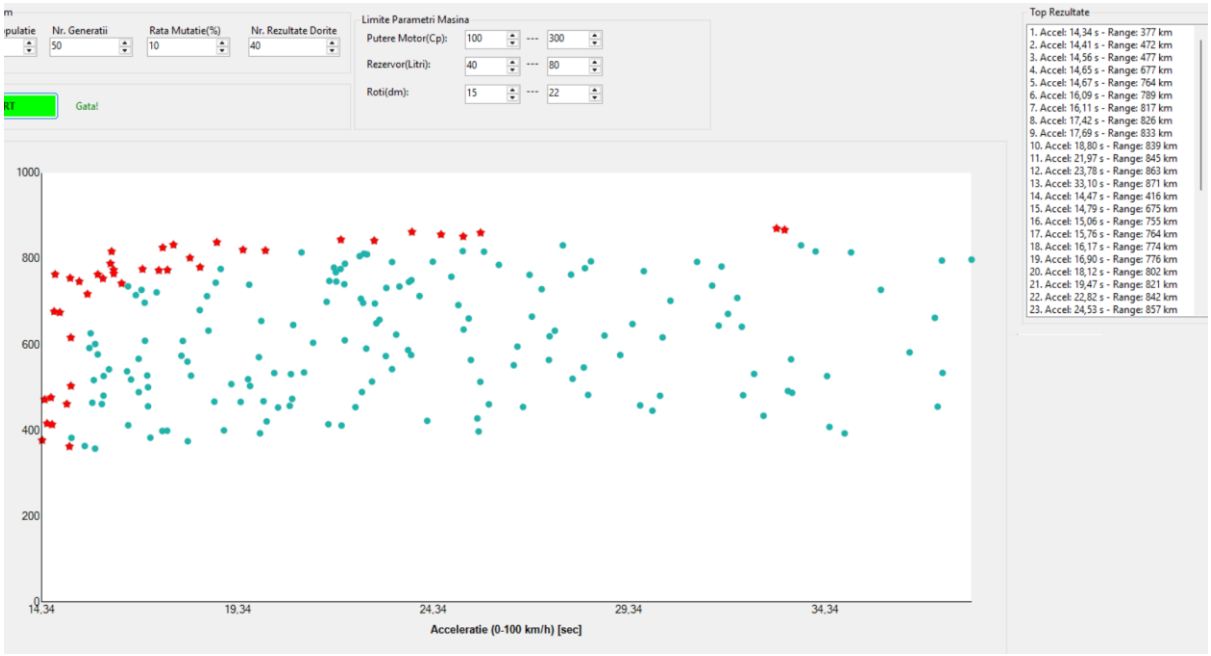
1.



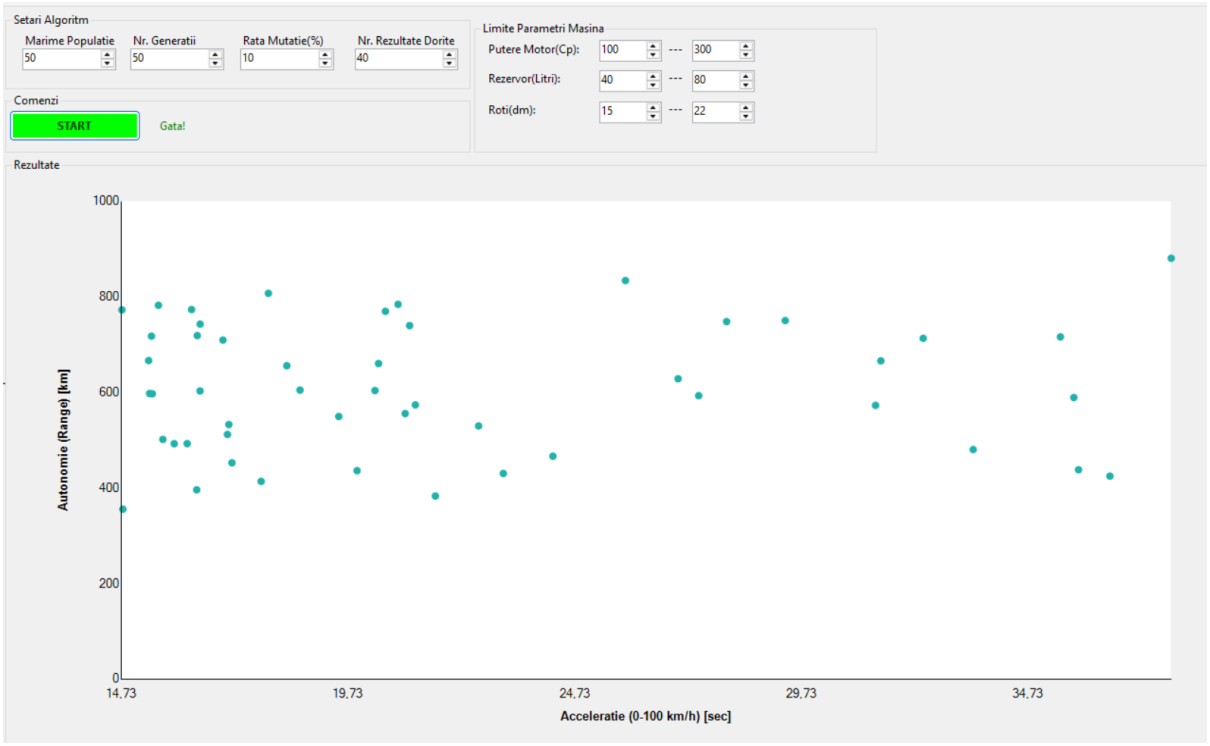


2.


Setari Algoritm
Marime Populatie
200
Nr. Generatii
50
Rata Mutatie(%)
10
Nr. Rezultate Dorite
40
Limite Parametri Masina
Putere Motor(Cp):
100
300
Rezervor(Litri):
40
80
Roti(dm):
15
22
Comenzi
START
Gata!
Top Rezultate
Rezultate
Autonomie (Range) [km]
Acceleratie (0-100 km/h) [sec]


Nr. Generatii
50
Rata Mutatie(%)
10
Nr. Rezultate Dorite
40
Limite Parametri Masina
Putere Motor(Cp):
100
300
Rezervor(Litri):
40
80
Roti(dm):
15
22
Gata!
Top Rezultate
1. Accel: 14,34 s - Range: 377 km
2. Accel: 14,41 s - Range: 472 km
3. Accel: 14,56 s - Range: 477 km
4. Accel: 14,65 s - Range: 677 km
5. Accel: 14,67 s - Range: 764 km
6. Accel: 16,09 s - Range: 789 km
7. Accel: 16,11 s - Range: 817 km
8. Accel: 17,42 s - Range: 826 km
9. Accel: 17,69 s - Range: 833 km
10. Accel: 18,80 s - Range: 839 km
11. Accel: 21,97 s - Range: 845 km
12. Accel: 23,78 s - Range: 863 km
13. Accel: 33,10 s - Range: 871 km
14. Accel: 14,47 s - Range: 416 km
15. Accel: 14,79 s - Range: 675 km
16. Accel: 15,06 s - Range: 755 km
17. Accel: 15,76 s - Range: 764 km
18. Accel: 16,17 s - Range: 774 km
19. Accel: 16,90 s - Range: 776 km
20. Accel: 18,12 s - Range: 802 km
21. Accel: 19,47 s - Range: 821 km
22. Accel: 22,82 s - Range: 842 km
23. Accel: 24,53 s - Range: 857 km
Acceleratie (0-100 km/h) [sec]

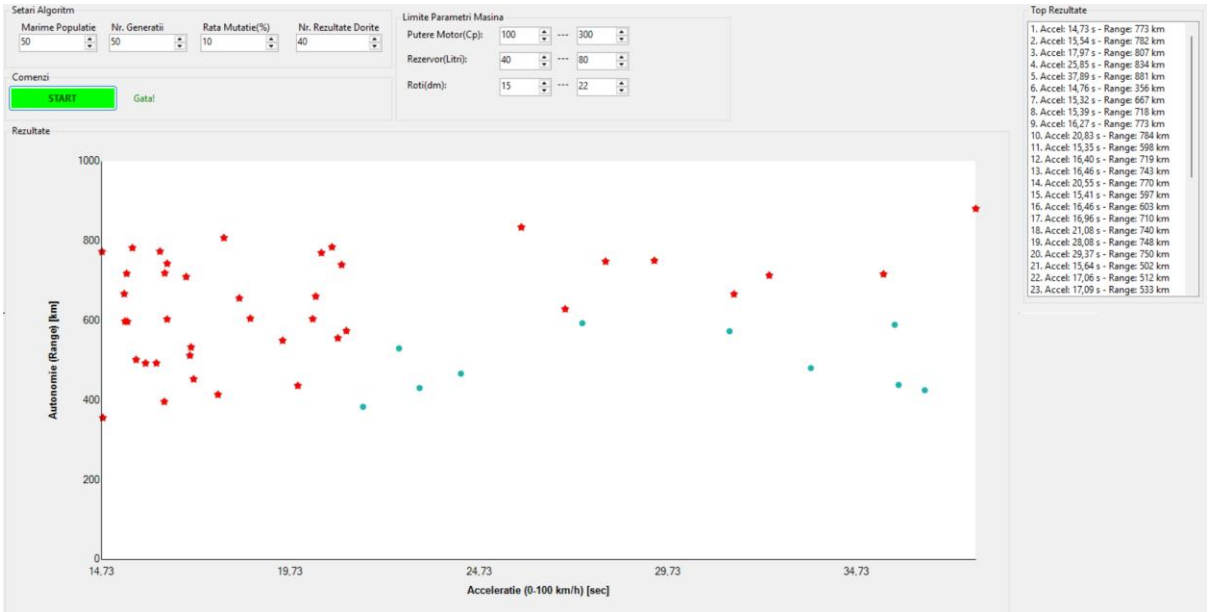
3.


Setari Algoritm
Marime Populatie
50
Nr. Generatii
50
Rata Mutatie(%)
10
Nr. Rezultate Dorite
40
Limite Parametri Masina
Putere Motor(Cp):
100

300
Rezervor(Litri):
40

80
Roti(dm):
15

22
Comenzi
START
Gata!
Rezultate
Autonomie (Range) [km]
1000
800
600
400
200
0
14,73
19,73
24,73
29,73
34,73
Acceleratie (0-100 km/h) [sec]


Setari Algoritm
Marime Populatie
50
Nr. Generatii
50
Rata Mutatie(%)
10
Nr. Rezultate Dorite
40
Limite Parametri Masina
Putere Motor(Cp):
100

300
Rezervor(Litri):
40

80
Roti(dm):
15

22
Comenzi
START
Gata!
Rezultate
Top Rezultate
1. Accel: 14,73 s - Range: 773 km
2. Accel: 15,54 s - Range: 782 km
3. Accel: 17,97 s - Range: 807 km
4. Accel: 25,85 s - Range: 834 km
5. Accel: 37,89 s - Range: 881 km
6. Accel: 14,76 s - Range: 356 km
7. Accel: 15,32 s - Range: 667 km
8. Accel: 15,39 s - Range: 718 km
9. Accel: 16,27 s - Range: 773 km
10. Accel: 20,83 s - Range: 784 km
11. Accel: 15,35 s - Range: 598 km
12. Accel: 16,40 s - Range: 719 km
13. Accel: 16,46 s - Range: 743 km
14. Accel: 20,55 s - Range: 770 km
15. Accel: 15,41 s - Range: 597 km
16. Accel: 16,46 s - Range: 603 km
17. Accel: 16,96 s - Range: 710 km
18. Accel: 21,08 s - Range: 740 km
19. Accel: 28,08 s - Range: 748 km
20. Accel: 29,37 s - Range: 750 km
21. Accel: 15,64 s - Range: 502 km
22. Accel: 17,06 s - Range: 512 km
23. Accel: 17,09 s - Range: 533 km
Autonomie (Range) [km]
1000
800
600
400
200
0
14,73
19,73
24,73
29,73
34,73
Acceleratie (0-100 km/h) [sec]

5. Bibliografie

https://link.springer.com/article/10.1007/s42452-019-1512-2

https://www.geeksforgeeks.org/deep-learning/non-dominated-sorting-genetic-algorithm-2-nsga-ii/

https://www.youtube.com/watch?v=SL-u_7hIqjA&list=PLjpWRUr9AM1Wp5eJBFYrDv8Nxk9tQL950