Universitatea Tehnică „Gheorghe Asachi" Iași
Facultatea de Automatică și Calculatoare Iași
Specializarea: Tehnologia Informației

Inteligență Artificială

Algoritm de optimizare multi-obiectiv NSGA II

Profesor coordonator:
Prof. dr. ing Florin Leon

Studenti:
Cobzariu Stefan-Alexandru - 1409A
Duca Tiberiu-Mihai - 1409A

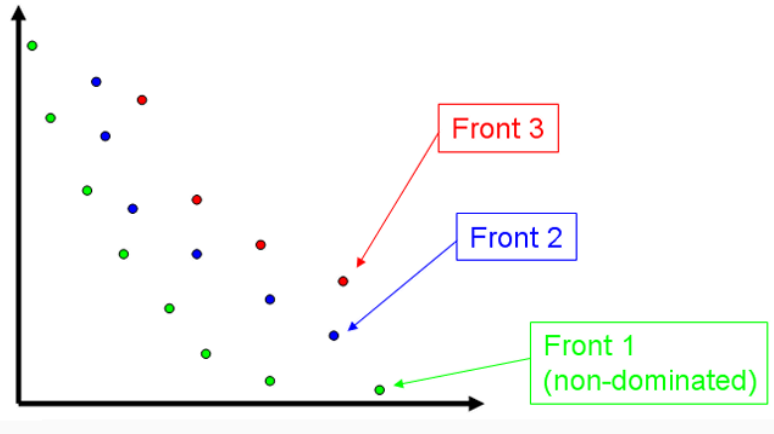
1. Descrierea problemei

Proiectul implementeaza algoritmul NSGA-II (Non-dominated Sorting Genetic Algorithm II), utilizat pentru optimizarea multi-obiectiv. Cele doua obiective contradictorii stabilite in cadrul proiectului reprezinta minimizarea pretului si maximizarea autonomiei in cazul unei masini electrice in vederea achizitionarii celui mai potrivit automobil tinand cont de criteriile mentionate. Proiectul include operatii specifice algoritmilor evolutivi necesare pentru imbunatatirea unei populatii formate dintr-o multime de solutii, unde fiecare solutie reprezinta cate un autoturism caruia ii sunt atribuite cele doua obiective.

2. Aspecte teoretice privind algoritmul

Algoritmul NSGA-II este o metoda de optimizare multi-obiectiv care urmareste gasirea unui set de solutii pentru optimizarea simultana a mai multor functii obiectiv care, de multe ori, sunt contradictorii. Acesta utilizeaza tehnici ale algoritmilor evolutivi la care se adauga calcularea fronturilor Pareto pentru gasirea solutiilor optime si aplicarea selectiei elitiste, unde cele mai bune solutii sunt pastrate in populatie.

Sortare nedominata: Scopul sortarii nedominate este de a identifica un set de solutii Pareto-optime, mai precis solutii care nu sunt dominate de catre alte solutii. Sortarea nedominata grupeaza solutiile in fronturi, in functie de cat de multe solutii domina, incepand cu frontul 1, care contine cele mai bune solutii.

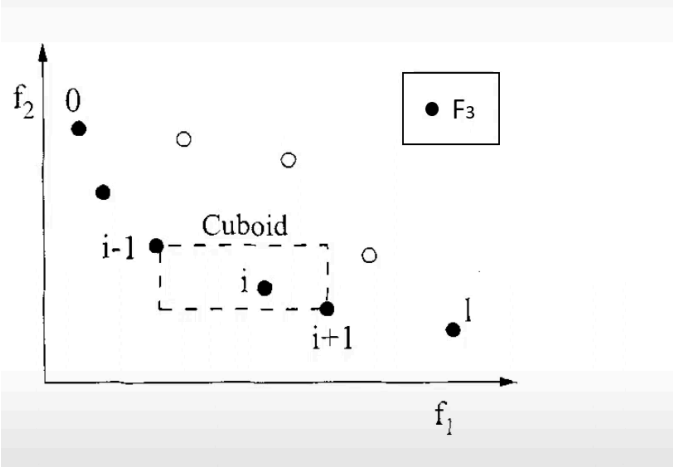


O solutie S1 domina o alta solutie S2 daca si numai daca:

- S1 nu este inferioara lui S2 in raport cu toate obiectivele: pentru orice valoarea a lui i, $S1(i) \geq S2(i)$

- S1 este strict superioara lui S2 in raport cu cel putin un obiectiv: exista un i pentru care S1(i) > S2(i)

Distanta de aglomerare a solutiilor



For o in objectives:

$$sort(front, o)$$
$$distance(front(\min)) = \inf$$
$$distance(front(\max)) = \inf$$
$$distance(i) = distance(i) + \frac{o(i+1) - o(i-1)}{o(max) - o(min)}$$

Distanta de aglomerare a solutiilor din Frontul Pareto reprezinta o masura de creare a diversitatii solutiilor ce consta in aproximarea densitatii solutiilor din vecinatatea unei solutii. Se calculeaza distanta dintre solutii asfel incat individul cu distanta de aglomerare mai mare este considerat mai bun, iar solutia are sanse mai mari sa fie selectata.Valoarea mai mica pentru distanta inseamna aglomerare si exista sanse mai mari ca solutia cu acea valoare sa nu ramana in populatie pe parcursul procesului evolutiv.Prima si ultima solutie din front sunt considerate +infinit pentru a ramane mereu in populatie.

3. Modalitatea de rezolvare

Primul pas in cadrul metodei de rezolvare a algoritmului reprezinta stabilirea numarului maxim de generatii pe care le va parcurge algoritmul, constituind criteriul de oprire.
Este stabilita dimensiunea populatiei ce reprezinta numarul de solutii (masini) din fiecare generatie. Dupa initializarea populatiei are loc procesul de evaluare a fiecarui individ din populatie pe baza functiilor obiectiv. Primul obiectiv, pretul, este minimizat iar cel de-al doilea obiectiv, autonomia masinii, este maximizat prin schimbarea semnului functiei obiectiv**.**

Este aplicata sortarea nedominata asupra populatiei initiale pentru a distribui solutiile in fronturi Pareto pe baza principiului de dominare. Urmatorul pas reprezinta inceperea algoritmului evolutiv prin selectarea parintilor pe baza rang-ului si a distantei de aglomerare, urmata de operatia de incrucisare aritmetica si mutatie. In urma acestui proces va rezulta o noua solutie (descendent). Acest proces se repeta de un numar de ori egal cu dimensiunea populatiei construind astfel o populatie noua de dimensiune 2n**.** Se aplica din nou sortarea nedominata pentru a grupa in fronturi Pareto, solutiile din noua populatie. Se parcurg solutiile din fiecare front nou rezultat si pentru fiecare dintre acestea se calculeaza distanta de aglomerare. In acest moment se selecteaza cele mai optime solutii din fronturi si se copiaza intr-o lista ce va reprezenta populatia urmatoare, cat timp acestea nu depasesc dimensiunea populatiei stabilite initial. Daca, prin adaugarea noilor solutii, se depaseste dimensiunea populatiei, se copiaza doar cele mai bune solutii pe baza distantei de aglomerare, pentru a completa populatia. Acesti pasi sunt reluati pana cand este indeplinita conditia de stop, mai precis parcurgerea numarului de generatii stabilit.

4. Parti semnificative din codul sursa

- **Calcularea distantei de aglomerare**

```
public static void CalculateCrowdingDistance(List<Car> front)
{
    // distanta de aglomerare inseamna distanta intre solutiile dintr-un front
    // mai mare inseamna mai bun , distanta mai mica inseamna solutiile sunt mai aproape una de alta(mai aglomerate)
    int numObjectives = front[0].objectives.Length; // obtin numarul de obiective pe care le are o solutie
    int f_size = front.Count;
    foreach (var car in front)
    {
        car.crowdingDistance = 0;  // initializez cu zero distanta de aglomerare pentru fiecare solutie
    }
    //calculez distanat pentru fiecare obiectiv
    for (int i = 0; i < numObjectives; i++)
    {
        front.Sort((a, b) => a.objectives[i].CompareTo(b.objectives[i]));
        // prima solutie si ultima solutie au distanta de aglomerare = cu infinit pentru a fi pastrate mereu in populatie
        front[0].crowdingDistance = double.PositiveInfinity;
        front[f_size - 1].crowdingDistance = double.PositiveInfinity;

        // calculez valoarea pentru min si max pentru a putea realiza normalizarea
        //pentru a putea face o comparatie corecta
        double min_val = front[0].objectives[i];
        double max_val = front[f_size - 1].objectives[i];
        // se evita calcularea pentru a nu face impartirea cu 0
        if (max_val == min_val)
        {
            continue;
        }
        // pentru fiecare solutie din front calculez distanta de aglomerare - fiecare obiectiv al solutiei avand o contributie proprie
        // contributiile celor 2 obiective se sumeaza si se obtine CrowdingDistance
        // se calculeaza dupa formula
        for (int j = 1; j < f_size - 1; j++)
        {
            front[j].crowdingDistance += (front[j + 1].objectives[i] - front[j - 1].objectives[i]) / (max_val - min_val);
        }
    }
}
```

- **Sortate nedominata**

```
// algoritmul de sortare nedominata pentru construirea fronturilor Pareto
2 references
public static List<List<Car>> NonDominatedSort(List<Car> population)
{
    var fronts = new List<List<Car>>();
    var dominations = new Dictionary<Car, int>();// numarul de solutii care domina o alta solutie

    // construirea frontului Pareto ce contine solutiile nedominate (frontul 1)
    foreach (var car in population)
    {
        dominations[car] = 0;
        car.dominatedBy = new List<Car>();
        foreach (var solution in population)
        {
            // se compara solutia car cu toate celelalte solutii si se testeaza daca domina sau este dominata
            if (Dominates(car, solution))
            {
                car.dominatedBy.Add(solution);
            }
            else if (Dominates(solution, car))
            {
                dominations[car]++;
            }
        }
        // daca nu exista solutii care sa domine solutia car atunci solutia car
        // este adaugata in primul front Pareto
        if (dominations[car] == 0)
        {
            car.rank = 0;
            if (fronts.Count == 0)
            {
                fronts.Add(new List<Car>());
            }
            fronts[0].Add(car);
        }
    }

    // construirea fronturilor dominate de primul front
    int i = 0;
    while (i < fronts.Count && fronts[i].Count > 0)
    {
        var nextFront = new List<Car>();
        foreach (var car in fronts[i])
        {
            // se parcurg toate solutiile dominate de o solutie car
            foreach (var dominated in car.dominatedBy)
            {
                dominations[dominated]--;
                // daca solutia dominata de car nu mai este dominata de alta solutir
                // atunci se adauga solutia dominata de car in frontul urmator
                if (dominations[dominated] == 0)
                {
                    dominated.rank = i + 1;
                    nextFront.Add(dominated);
                }
            }
        }
        // se adauga frontul urmator in lista de fronturi
        if (nextFront.Count > 0)
        {
            fronts.Add(nextFront);
        }
        i++;
    }
    return fronts;
}
```

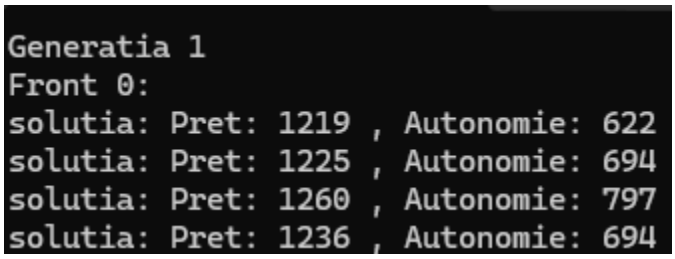
5. Rezultate obtinute prin rularea programului in diverse situatii

Numarul de generatii = 100
Dimensiunea populatiei = 50
Rata de mutatie = 0.17
Crossover = 0.9
Cazul in care parametrii alesi ofera cele mai bune rezultate

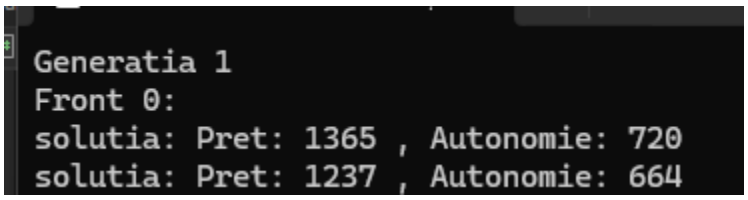
```
Generatia 1
Front 0:
solutia: Pret: 5205 , Autonomie: 790
solutia: Pret: 3879 , Autonomie: 786
solutia: Pret: 4866 , Autonomie: 789
solutia: Pret: 1715 , Autonomie: 775
solutia: Pret: 1551 , Autonomie: 659
solutia: Pret: 1610 , Autonomie: 662
```

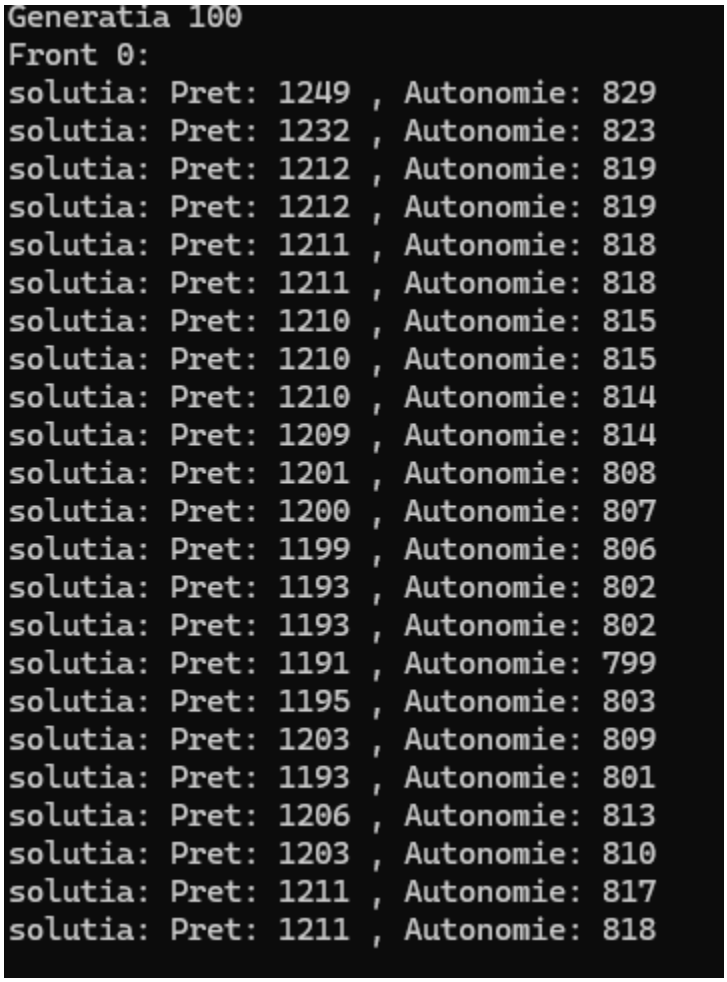
```
Generatia 100
Front 0:
solutia: Pret: 1685 , Autonomie: 855
solutia: Pret: 1655 , Autonomie: 837
solutia: Pret: 1609 , Autonomie: 830
solutia: Pret: 1592 , Autonomie: 825
solutia: Pret: 1672 , Autonomie: 848
solutia: Pret: 1664 , Autonomie: 846
solutia: Pret: 1598 , Autonomie: 825
solutia: Pret: 1551 , Autonomie: 762
solutia: Pret: 1551 , Autonomie: 750
solutia: Pret: 1572 , Autonomie: 795
solutia: Pret: 1569 , Autonomie: 787
solutia: Pret: 1566 , Autonomie: 782
solutia: Pret: 1564 , Autonomie: 778
solutia: Pret: 1543 , Autonomie: 735
solutia: Pret: 1549 , Autonomie: 746
solutia: Pret: 1540 , Autonomie: 725
solutia: Pret: 1568 , Autonomie: 785
solutia: Pret: 1537 , Autonomie: 718
solutia: Pret: 1550 , Autonomie: 746
solutia: Pret: 1576 , Autonomie: 797
solutia: Pret: 1577 , Autonomie: 797
solutia: Pret: 1533 , Autonomie: 712
solutia: Pret: 1539 , Autonomie: 723
solutia: Pret: 1538 , Autonomie: 721
solutia: Pret: 1681 , Autonomie: 852
solutia: Pret: 1540 , Autonomie: 724
solutia: Pret: 1530 , Autonomie: 710
solutia: Pret: 1534 , Autonomie: 714
solutia: Pret: 1684 , Autonomie: 855
solutia: Pret: 1669 , Autonomie: 847
solutia: Pret: 1568 , Autonomie: 785
solutia: Pret: 1578 , Autonomie: 806
solutia: Pret: 1542 , Autonomie: 732
solutia: Pret: 1628 , Autonomie: 835
solutia: Pret: 1535 , Autonomie: 717
solutia: Pret: 1650 , Autonomie: 835
solutia: Pret: 1546 , Autonomie: 745
solutia: Pret: 1552 , Autonomie: 774
solutia: Pret: 1537 , Autonomie: 720
solutia: Pret: 1678 , Autonomie: 851
solutia: Pret: 1523 , Autonomie: 702
```

Numarul de generatii = 20
Dimensiunea populatiei = 50
Rata de mutatie = 0.17
Crossover = 0.9
Numarul de generatii este prea mic iar algoritmul nu parcurge destule etape pentru a optimiza solutiile.



Numarul de generatii = 100
Dimensiunea populatiei = 20
Rata de mutatie = 0.17
Crossover = 0.9
Dimensiunea populatiei este prea mica iar din aceasta cauza diversitatea solutiilor este redusa.





6. Concluzii

Algoritmul NSGA-II reprezinta un algoritm pentru optimizare multi-obiectiv, utilizat in domeniul Inteligentei Artificiale, bazat pe utilizarea unui algoritm evolutiv si calcularea fronturilor Pareto care clasifica solutiile pentru a le determina pe cele mai optime. Acesta rezolva problema identificarii solutiilor optime in cazul existentei obiectivelor contradictorii care nu pot fi imbunatatite simultan.

7. Bibliografie

- Curs 5 - Inteligenta Artificiala
- Laborator 8 - Inteligenta Artificiala
- https://www.youtube.com/watch?app=desktop&v=SL-u_7hIqjA&t=443s
- https://www.geeksforgeeks.org/non-dominated-sorting-genetic-algorithm-2-nsga-ii/
- https://web.njit.edu/~horacio/Math451H/download/Seshadri_NSGA-II.pdf

8. Lista de atributii pentru fiecare membru

Duca Tiberiu-Mihai - 1409A:

- Functia NonDominatedSort
- Functia EvaluatePopulation
- Functia Dominates

Cobzariu Stefan-Alexandru - 1409A:

- Functia CalculateCrowdingDistance
- Functia ArithmeticCrossover
- Functia Mutation
- Functia SelectedByRankAndDistance

Functiile realizate in comun de catre ambii studenti:

- Functia main
- Functia InitializePopulation
- Clasa Car