Universitatea Tehnică ”Gheorghe Asachi” din Iași
Facultatea de Automatică și Calculatoare
Domeniul Calculatoare și Tehnologia Informației
Specializarea Tehnologia Informației

Inteligență artificială
- proiect -

Cadru didactic îndrumător:

Ș.l.dr. Florin Leon

Studenți:

Ursachi Gabriela - 1410B

Marian Bogdan Costel - 1410B

Mihai Valentin - 1410B

An universitar 2022 - 2023

CUPRINS

1. Problema considerata

În contextul planificării unei călătorii aeriene, trebuie avute în vedere mai multe detalii, printre care și găsirea unui bilet de avion care să se plieze anumitor cerințe impuse (oras sursa/destinatie, interval/data preferat(a) de zbor, etc).

Aplicația propusă în cadrul acestui proiect vine în sprijinul utilizatorilor, simplificand procesul de căutare a unui astfel de bilet și oferind o gama largă de variante, din care se pot alege doar acelea considerate potrivite în raport cu resursele avute la dispoziție (resurse financiare/de timp). Plecand de la premisa ca nu se pot satisface simultan mai multe constrângeri contradictorii (un bilet de zbor cu un preț mic va necesita cu siguranța o perioada mai indelungata de la data achiziționării pana la data efectiva zborului, sau va avea un grad de inconvenienta mai ridicat), aplicația considerată se poate încadra cu succes în categoria celor ce presupun soluționarea unor probleme de optimizare multi-obiectiv.

2. Aspecte teoretice privind algoritmul

Optimizarea multi-obiectiv (cunoscută și sub numele de optimizare vectoriala sau optimizare Pareto) este un domeniu de luare a deciziilor cu criterii multiple (*Multiple-criteria decision analysis*). Se referă la probleme matematice care implică mai mult de o funcție obiectiv, ce urmează să fie optimizată simultan, și pentru care nu va exista o singura soluție, ci un set de solutii. Clasificarea/compararea soluțiilor are la bază conceptul de dominanță Pareto. Astfel, o soluție S1 domină o soluție S2 dacă și numai dacă:

- ➔ S1 nu este inferioară lui S2 în raport cu toate obiectivele: i, S1 (i) ≥ S2 (i)
- ➔ S1 este strict superioară lui S2 în raport cu cel puțin un obiectiv: i, S1 (i) > S2 (i)

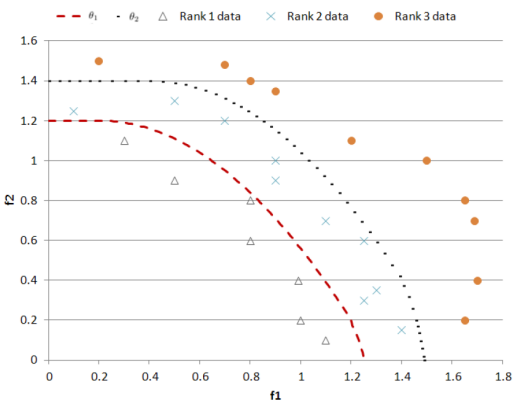
Mulțimea tuturor soluțiilor nedominate se numește **front Pareto**.

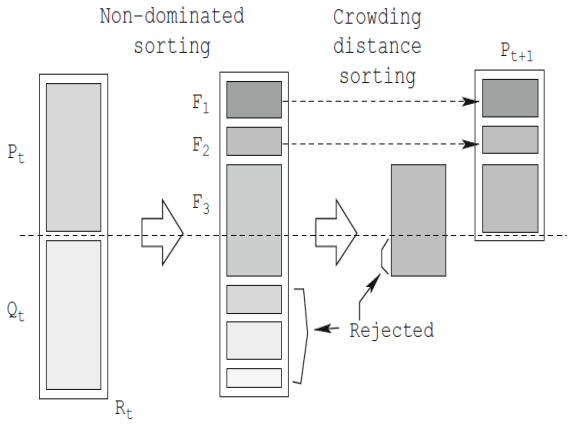
NSGA-II(Nondominated Sorting Genetic Algorithm) este unul dintre cei mai populari algoritmi de optimizare multi-obiectiv. Acesta utilizează un algoritm evolutiv simplu la care se adaugă calcularea frontului Pareto și are trei caracteristici speciale:

- ➔ folosește un principiu elitist, adică elitele unei populații au posibilitatea de a fi duse la generația următoare
- ➔ utilizează un mecanism explicit de conservare a diversității - distanța de aglomerare (engl. Crowding distance)
- ➔ pune în evidenta soluțiile care nu sunt dominate.

NSGA-II are la baza următoarea procedura:

1. Se efectuează o sortare non-dominată în combinația dintre populațiile de părinți și descendenți și se face o clasificare pe fronturi (se sorteaza crescator în funcție de un nivel ascendent de non-dominație ~ rank).





2. Se introduc în populatia următoare indivizii tinand cont de acest rank.

3. Dacă un front trebuie luat parțial, atunci se efectuează o sortare pe baza distanței de aglomerare, care este legată de densitatea de soluții în jurul fiecărei soluții.

4. Se creeaza o populație descendentă din aceasta noua populatie, utilizand selectia prin elitism, încrucișare și mutație.

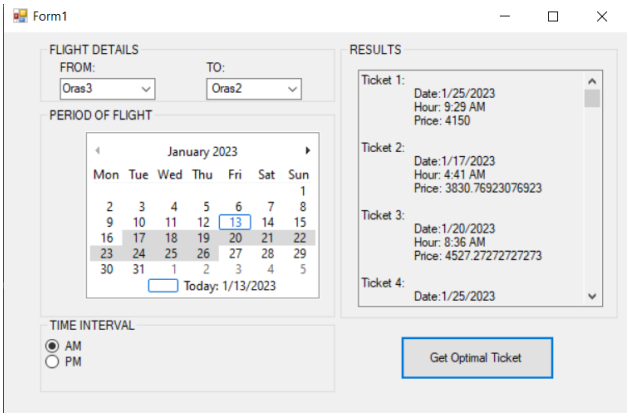
3. Modalitatea de rezolvare

Algoritmul este transpus în cadrul aplicației prin respectarea următoarelor conventii:

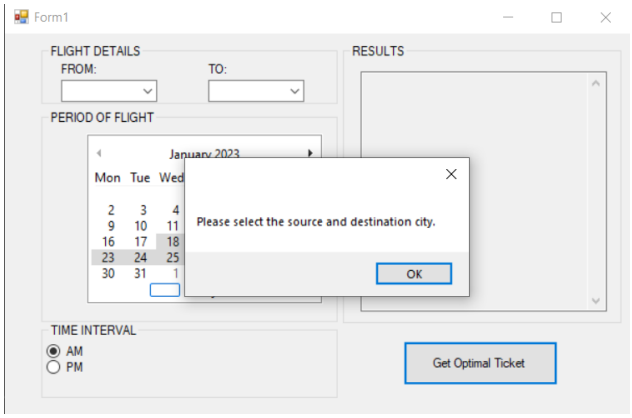
➔ ***Genele unui cromozom***: numărul de km parcursi de la sursa la destinatie, numărul de zile de așteptare până la data efectiva a zborului, ora la care are loc zborul

➔ ***Funcțiile obiectiv care se doresc a fi optimizate(minimizate)****:* durata pana la data de zbor, prețul biletului, gradul de inconvenienta în ceea ce privește ora zborului

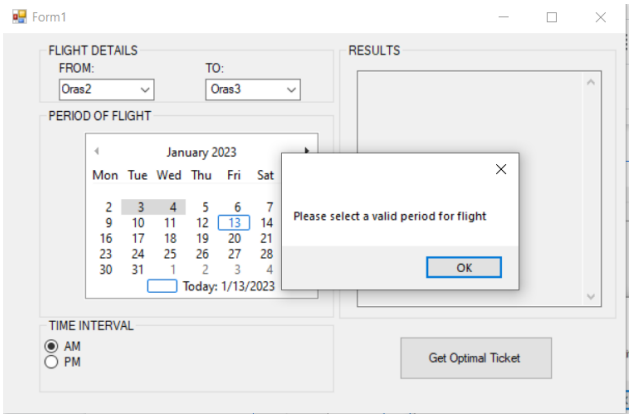
Un utilizator își poate alege perioada în care ar dori să găsească zbor, orașul de plecare/sosire, și dacă vrea ca zborul să fie în timpul zilei sau al nopții. După ce introduce toate aceste opțiuni, va primi ca răspuns o lista de zboruri disponibile (cu respectarea constrângerilor stabilite). Avand lista anterior mentionata, el va decide ce compromis este dispus sa faca pentru a cumpara biletul potrivit: va alege biletul cu preț minim, ori pe cel cu cea mai convenabila ora de plecare, biletul ce presupune așteptarea unui număr cat mai mic de zile pana la zbor, sau o combinație dintre toate acestea.



Pentru fiecare încercare incorectă a utilizatorului de a selecta o optiune, acesta va fi notificat prin intermediul unui mesaj sugestiv de eroare.



Neselectarea orașului sursa/destinatie



Introducerea unei perioade preferate de zbor care este anterioara datei curente

4. Părți semnificative din cod

```
/// <summary>
/// Given a value for time, it assigns a degree of inconvenience for a chromosome
/// </summary>
/// <param name="timeOfTheDay">Time value</param>
/// <returns>Inconvenience degree</returns>
private int ComputeInconvenienceDegree(DateTime timeOfTheDay)
{
    if( ((0 <= timeOfTheDay.Hour) && (timeOfTheDay.Hour <= 5)) || (23 == timeOfTheDay.Hour) )
    {
        return (int)InconvenienceDegree.Maximum;
    }
    else if (((6<= timeOfTheDay.Hour) && (timeOfTheDay.Hour <= 8)) || ((20 <= timeOfTheDay.Hour) &&
(timeOfTheDay.Hour <= 22)))
    {
        return (int)InconvenienceDegree.Medium;
    }
    else
    {
        return (int)InconvenienceDegree.Minimum;
    }
}
```

Listing 1: Stabilire grad de inconvenienta în funcție de ora

Pentru zborurile din timpul nopții va exista un grad de inconvenienta mai ridicat decat pentru cele din timpul zilei.

```
/// <summary>
/// Establishes the values of the objectives functions according to a certain formula.
/// </summary>
public void ComputeFitness()
{
    Objectives[(int)Objective.Duration] = NoOfDays;
```

```
        Objectives[(int)Objective.Inconvenience] = ComputeInconvenienceDegree(HourOfTheDay);
        Objectives[(int)Objective.Price] = 10 * (1 + Kilometers) / (NoOfDays + Objectives[(int)Objective.Inconvenience]);
    }
```

Listing 2: Stabilire valori pentru funcțiile obiectiv

Valoarea prețului variază direct proporțional cu numărul de km parcursi de la sursa la destinatie și invers proporțional cu numărul de zile de așteptare, respectiv gradul de inconvenienta.

```
  /// <summary>
/// It verifies if one solution c1 is better than another one (c2)
/// </summary>
/// <param name="c1">The first compared solution</param>
/// <param name="c2">The second compared solution</param>
/// <returns>
///     True if the first solution is better than the second one.
///     False if the first solution is not better than the second one.
/// </returns>
private bool Dominates(Chromosome c1, Chromosome c2)
{
   bool indicator = false;

   // Se analizează fiecare obiectiv
   for(int i = 0; i < c1.Objectives.Length; i++)
   {
      double objC1 = c1.Objectives[i];
      double objC2 = c2.Objectives[i];

      // S1 este strict superioară lui S2 în raport cu cel puțin un obiectiv
      if (objC1 < objC2)
      {
         indicator = true;
      }

      // S1 nu este inferioară lui S2 în raport cu toate obiectivele
      // dacă e inferioară în raport cu cel puțin un obiectiv => S1 nu domina S2
      if (objC2 < objC1)
      {
         return false;
      }
   }

   return indicator;
  }
```

Listing 3: Verificare dacă o soluție c1 este mai buna (domina) decat o solutie c2

Se compara doua soluții și se determina dacă S1 domina S2 după regula:

S1 nu este inferioară lui S2 în raport cu toate obiectivele: i, S1 (i) ≥ S2 (i)

S1 este strict superioară lui S2 în raport cu cel puțin un obiectiv: i, S1 (i) > S2 (i)

```
    /// <summary>
   /// It assign each chromosome of the given population to the corresponding front
   /// based on its non-domination rank
   /// </summary>
   /// <param name="population">All chromosomes of population</param>
   /// <returns>All generated fronts</returns>
```

```
private List<Front> FastNondominatedSort(ref Chromosome[] population)
{
    int currentIndex = 0;
    List<Front> fronts = new List<Front>();

    // deep copy of population
    List<Chromosome> copyOfPopulation = new List<Chromosome>();
    for (int i = 0; i < population.Length; i++)
    {
        copyOfPopulation.Add(new Chromosome(population[i]));
    }

    // populatia primita ca parametru este reinitializata pentru a contine elementele sortate la final
    population = new Chromosome[population.Length];

    while (copyOfPopulation.Count > 0)
    {
        Front currentFront = new Front();
        Boolean isDominant = true;

        // se compara fitness-ul unui cromozom cu fitness-ul tuturor celorlalti cromozomi
        // și se adaugă într-un front toți cromozomii care nu sunt dominati
        // după ce se finalizează crearea unui astfel de front, se reia procesul pana cand
        // nu mai raman cromozomi neasignati vreunui front
        for (int i = 0; i < copyOfPopulation.Count; i++)
        {
            isDominant = true;

            for (int j = 0; j < copyOfPopulation.Count; j++)
            {
                if (i == j)
                {
                    continue;
                }

                if (Dominates(copyOfPopulation[j], copyOfPopulation[i]))
                {
                    isDominant = false;
                    break;
                }
            }

            // dacă soluția curentă nu este dominată de o alta solutie, atunci se adauga
            // la frontul curent
            if (isDominant)
            {
                currentFront.AddChromosome(copyOfPopulation[i]);
                population[currentIndex] = copyOfPopulation[i];
                currentIndex++;
            }
        }

        fronts.Add(currentFront);

        for (int k = 0; k < currentFront.Chromosomes.Count; k++)
        {
            copyOfPopulation.Remove(currentFront.Chromosomes[k]);
        }
    }

    return fronts;
```

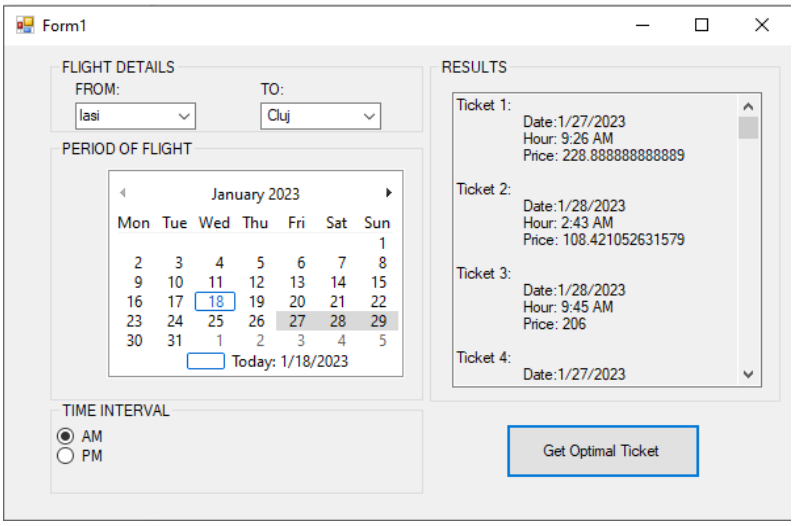
```
}
```

Listing 4: Sortare a populatiei pe fronturi în funcție de gradul de non-dominanta (cu cat soluția domina mai multe soluții, cu atat se va găsi pe un front de rank mai mic)

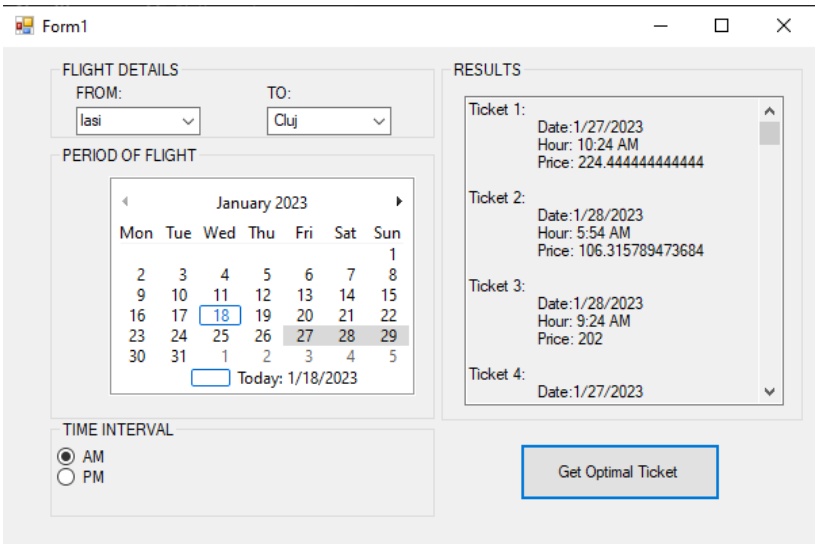
5. Rularea programului în diverse situații

I. PopulationSize = 50
CrossoverRate = 0.9
MutationRate = 0.07

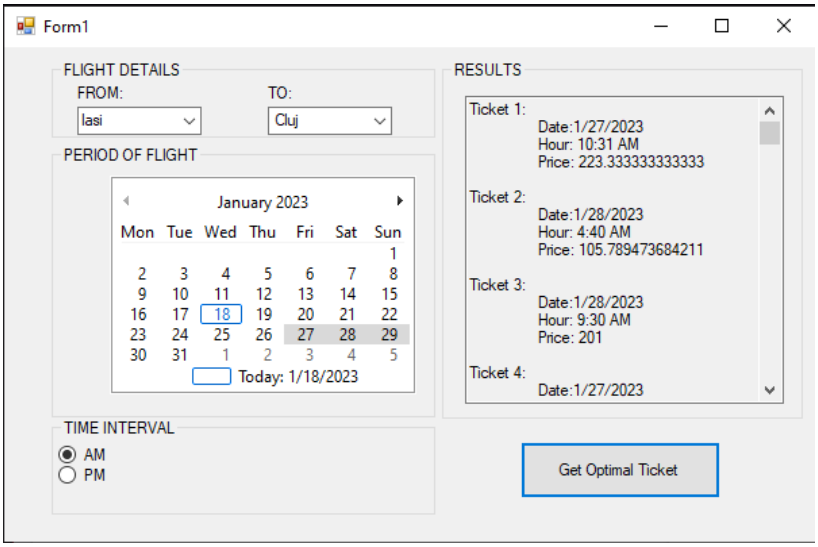
A. MaxNumOfGenerations = 30



B. MaxNumOfGenerations = 250

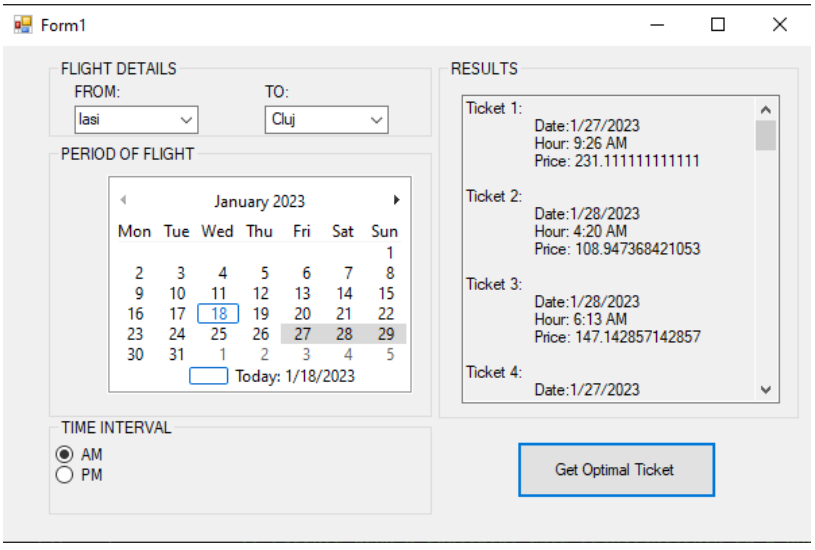


C. MaxNumOfGenerations = 500

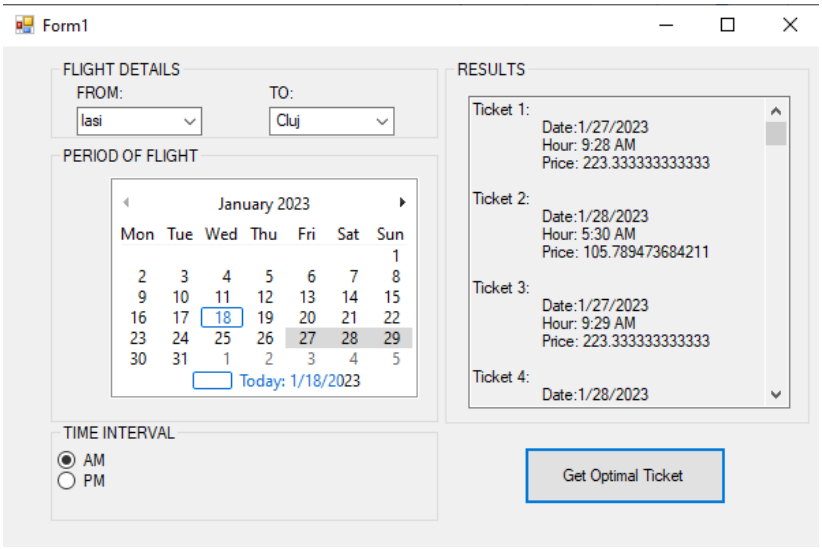


II. PopulationSize = 100
CrossoverRate = 0.95
MutationRate = 0.04

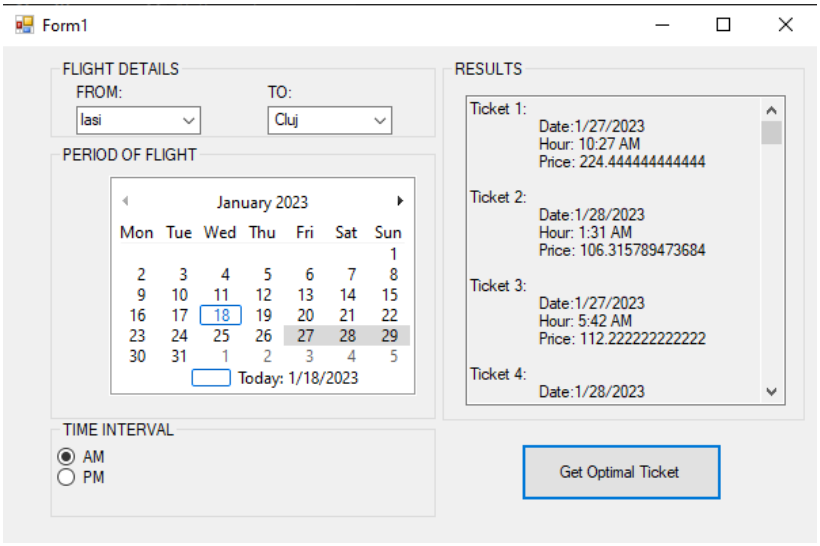
A. MaxNumOfGenerations = 30



B. MaxNumOfGenerations = 250



C. MaxNumOfGenerations = 500



Am ales sa testam aplicație pe doua seturi de date, unde pe aceleași date de intrare din partea utilizatorului, modificăm doar numărul de generații.
Se poate observa cum, in ambele seturi de date, prima rulare cu un număr mic de 30 de generatii are rezultate diferite fata de a doua rulare, unde avem 200 de generații.
De asemenea, de la a doua la a treia rulare, nu sunt schimbări ale datelor de ieșire, deci se poate observa ca, de la un anumit număr de generații în sus, algoritmul da aceleași valori.

6. Concluzii

Aplicația dezvoltată demonstreaza utilitatea algoritmului de optimizare NSGA-II în probleme din viața reala. Cu ajutorul acestuia, s-a realizat o aplicație care oferă soluții pentru căutarea și achiziționarea unui bilet de avion, cu respectarea unor condiții impuse de un posibil cumparator.

7. Bibliografie

Optimizare multi-obiectiv - Multi-objective optimization - abcdef.wiki
Analiza deciziilor cu criterii multiple - Multiple-criteria decision analysis - abcdef.wiki
NSGA-II explained! - analytics lab @ OU (oklahomaanalytics.com)
Overview of NSGA-II for Optimizing Machining Process Parameters (sciencedirectassets.com)

8. Atribuire sarcini

Ursachi Gabriela:
- Definire structura pentru un individ (fișier Chromosome.cs)
- Implementare funcție 'FastNondominatedSort' din fișierul Algorithm.cs
- Implementare funcție 'Dominates' din fișierul Algorithm.cs
- Implementare functie 'CrowdingDistanceAssignment' din Algorithm.cs
- Definire structuri de date pentru front, funcții obiectiv și grade de inconvenienta

Marian Bogdan Costel:
- Initializare parametri algoritm evolutiv prin citire din fișier (AlgParameters.txt)
- Implementare funcție 'GenerateChildren' din fișierul Algorithm.cs
- Implementare funcție 'SelectParents' din fișierul Algorithm.cs
- Realizare teste automate
- Rulare program in diverse situatii

Mihai Valentin:
- Initializare destinatii valide de călătorie (din fișierul ValidDestinations.txt)
- Implementare funcție 'GeneratePopulation' din fișierul Algorithm.cs
- Implementare funcție 'CrossoverAndMutation' din fișierul Algorithm.cs
- Implementare interfața cu utilizatorul