****

**Universitatea Tehnică „Gheorghe Asachi” din Iași**

**Facultatea de Automatică și Calculatoare**

**Domeniul: Tehnologia Informației**

**Algoritmul NSGA-II**

**pentru optimizare multi-obiectiv**

Proiect la disciplina

Inteligența Artificială

**Studenți:** Muraru Alexandru

Enachi Vasile

Ilioi Alexandru

**Anul:** 4

**Grupa:** 1409B

Contents

[Capitolul 1. Descrierea problemei considerate 2](#_Toc61991059)

[Capitolul 2. Aspecte teoretice privind algoritmul 2](#_Toc61991060)

[Capitolul 3. Modalitatea de rezolvare 3](#_Toc61991061)

[3.1 Interacțiunea modulelor 3](#_Toc61991062)

[3.2 Diagrama de clase a modului de algoritm 4](#_Toc61991063)

[Capitolul 4. Analiza codului algoritmului 5](#_Toc61991064)

[4.1 Sortarea pe fronturi 5](#_Toc61991065)

[4.2 Încrucișarea 5](#_Toc61991066)

[4.3 Mutația 6](#_Toc61991067)

[4.4 Selecția pe o listă de cromozomi 6](#_Toc61991068)

[4.5 Selecția prin turnir 7](#_Toc61991069)

[4.6 Crowding Distance Sort 8](#_Toc61991070)

[4.7 Non Dominated Sort 8](#_Toc61991071)

[4.8 Algoritmul NSGA-II 10](#_Toc61991072)

[Capitolul 5. GUI 12](#_Toc61991073)

[Capitolul 6. Rezultatele obținute prin rularea programului în diverse situații 13](#_Toc61991074)

[Capitolul 7. Concluzii 16](#_Toc61991075)

[Capitolul 8. Testare 16](#_Toc61991076)

[Capitolul 9. Bibliografie 16](#_Toc61991077)

[Capitolul 10. Roluri 16](#_Toc61991078)

# Capitolul 1. Descrierea problemei considerate

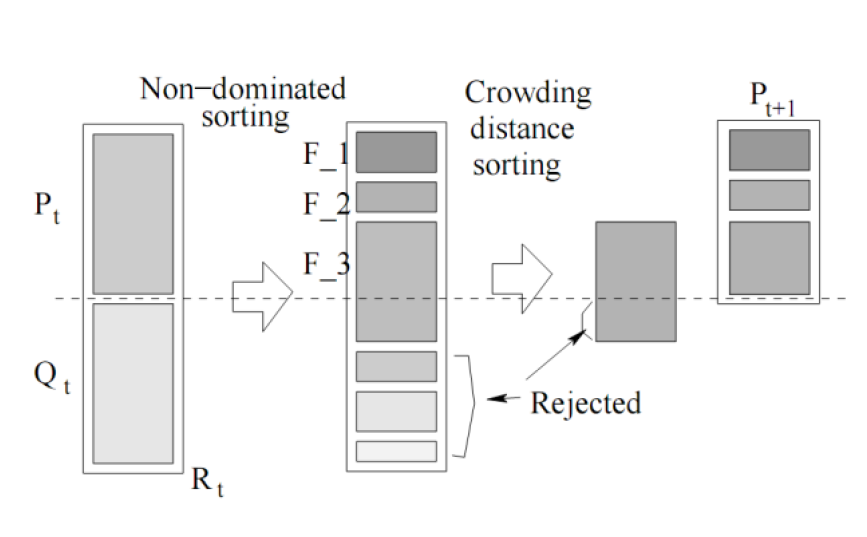
Problema clasică a rucsacului este o problemă de optimizare combinatorică.

Dată fiind o mulțime de elemente, fiecare item are o greutate și o valoare. Scopul problemei acela de a determina combinația optimă de elemente selectate, dată fiind o greutate maximă, astfel încât valoarea itemelor să fie maximă.

Problema rezolvată în proiectul nostru reprezintă o extindere a problemei rucsacului în sensul că valoarea unui item este reprezentată de perechea (timp, cost). În cazul clasic optimizare se realizează pe o singură dimensiune, dar în cazul nostru avem de a face cu o problemă multi-dimensională, care necesită o abordare ușor diferită și de aceea am ales rezolvarea ei prin tehnici de inteligență artificială.

# Capitolul 2. Aspecte teoretice privind algoritmul

Optimizarea multi-obiectiv (cunoscută și sub numele de programare multi-obiectivă, optimizarea vectorială, optimizarea multicriteriilor, optimizarea multi-atributului sau optimizarea Pareto) este o zonă de luare a deciziilor cu criterii multiple care se referă la probleme de optimizare matematică care implică mai multe funcții obiective care urmează să fie optimizate simultan.

****NSGA-II reprezintă un algoritm ce face parte din categoria algoritmilor de optimizare multi-obiectiv.

Acesta este alcătuit din 4 pași de execuție:

* Producerea populației viitoare(Qt) prin operațiile genetice aplicate părinților(Pt) și reinserarea lor în vectorul populației(Rt):
  + Selecție (în cazul prezent prin Turnir)
  + Crossover
  + Mutație
  + Reinserție
* Sortarea Non-Dominantă, în urma căreia populația este împărțită în fronturi, urmând ca parte din acestea (numărul elementelor din Pt initial) să refacă vectorul parinte(Pt)
* Aplicarea Crowding Distance Sorting ultimului front ce trebuie împărțit înainte de reinserare
* Reinserarea datelor în populația părinte(Pt)

# Capitolul 3. Modalitatea de rezolvare

Au fost create 4 module:

* **Algorithm –** implementarea funcționalității algritmului NSGA-II
* **CLI** – rularea algoritmului din command line folosind un mock data
* **GUI** – rularea algoritmului din interfața grafică, cu posibilitatea de adăugare iteme noi, vizualizare rezultate
* **Tests** – modul pentru testarea unor funcționalități a cromozomilor

Structurile de date construite și folosite sunt:

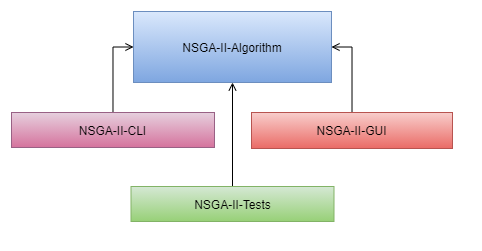
**Item –** conține informațiile despre denumirea acelui item, valoarea, greutatea și timpul necesar colectării acestuia.

**Chromosome –** conține un tablou de bool-uri care semnifică dacă un item dintr-o listă de iteme este selectat sau nu. Conține funcții de calculare a valorii fitness după valoarea și timp necesar colectării

**CrowdingDistanceAtom –** conține un cromozom țintă și o distanță de aglomerare.

**NonDominatedSort –** conține un cromozom țintă, dominationCount care arată de câți atomi este dominat și dominates care reprezintă o listă de atomi dominați.

## 3.1 Interacțiunea modulelor

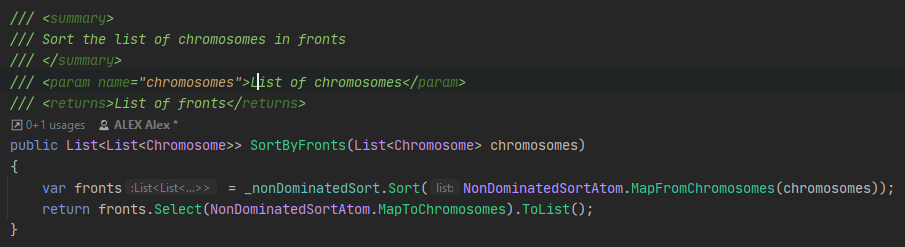


## 3.2 Diagrama de clase a modului de algoritm

# Capitolul 4. Analiza codului algoritmului

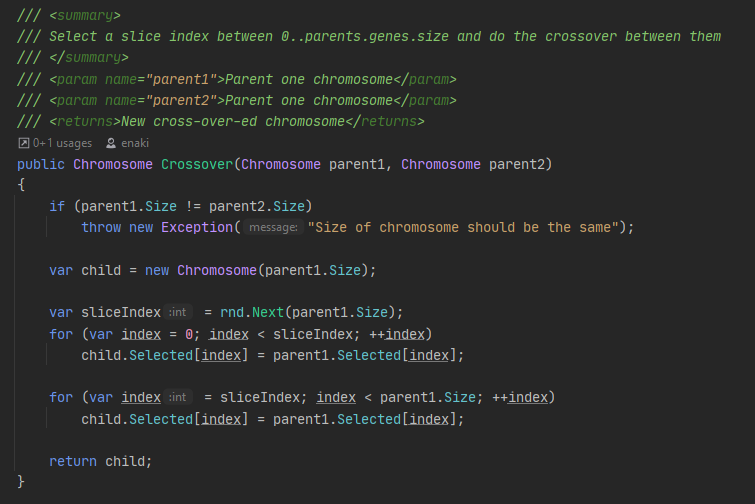
## 4.1 Sortarea pe fronturi

1. Primește o listă de cromozomi
2. Cromozomii sunt sortați pe front-uri folosind NonDominatedSort
3. Este returnată lista de front-uri



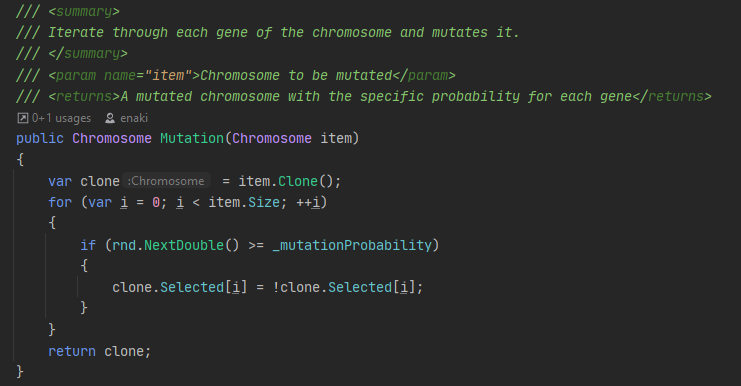
## 4.2 Încrucișarea

1. Primește ca parametru 2 cromozomi
2. Este generat un index al genei în intervalul [0..gene\_size]
3. Se creează un nou cromozom care conține până la indexul selectat genele de la primul părinte și după index acel index – genele de la al doilea părinte



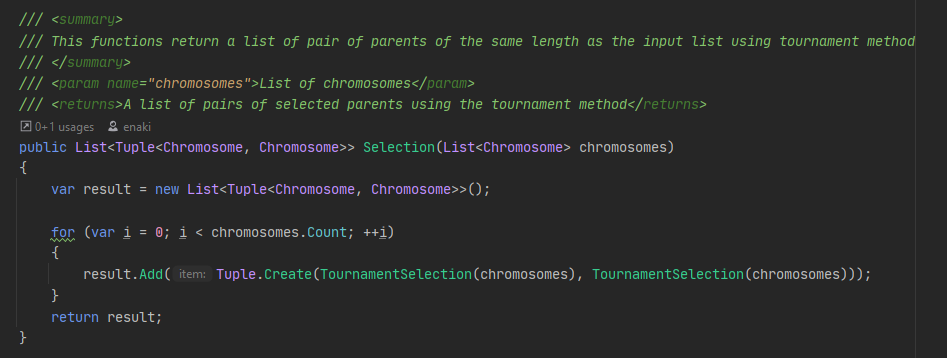
## 4.3 Mutația

1. Primește ca parametru un cromozom
2. Iterează prin genele acelui cromozom și le mută cu o probabilitate de mutație
3. Cromozomul generat este returnat



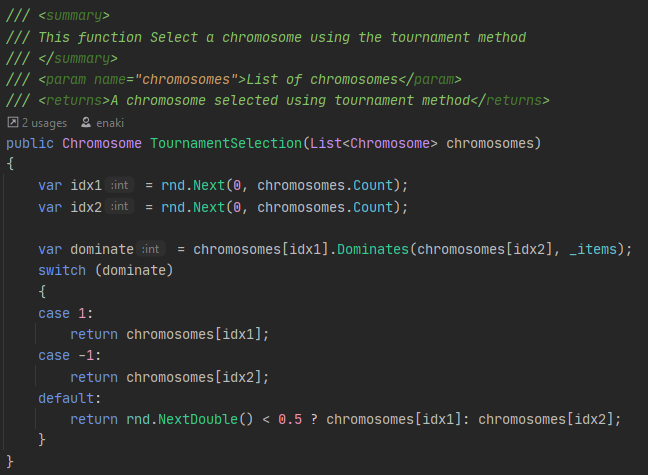
## 4.4 Selecția pe o listă de cromozomi

1. Primește o listă de cromozomi
2. Se creează o listă de n perechi tuple, unde n este numărul de cromozomi din lista parametru
3. Perechea este creată prin metoda turnirului



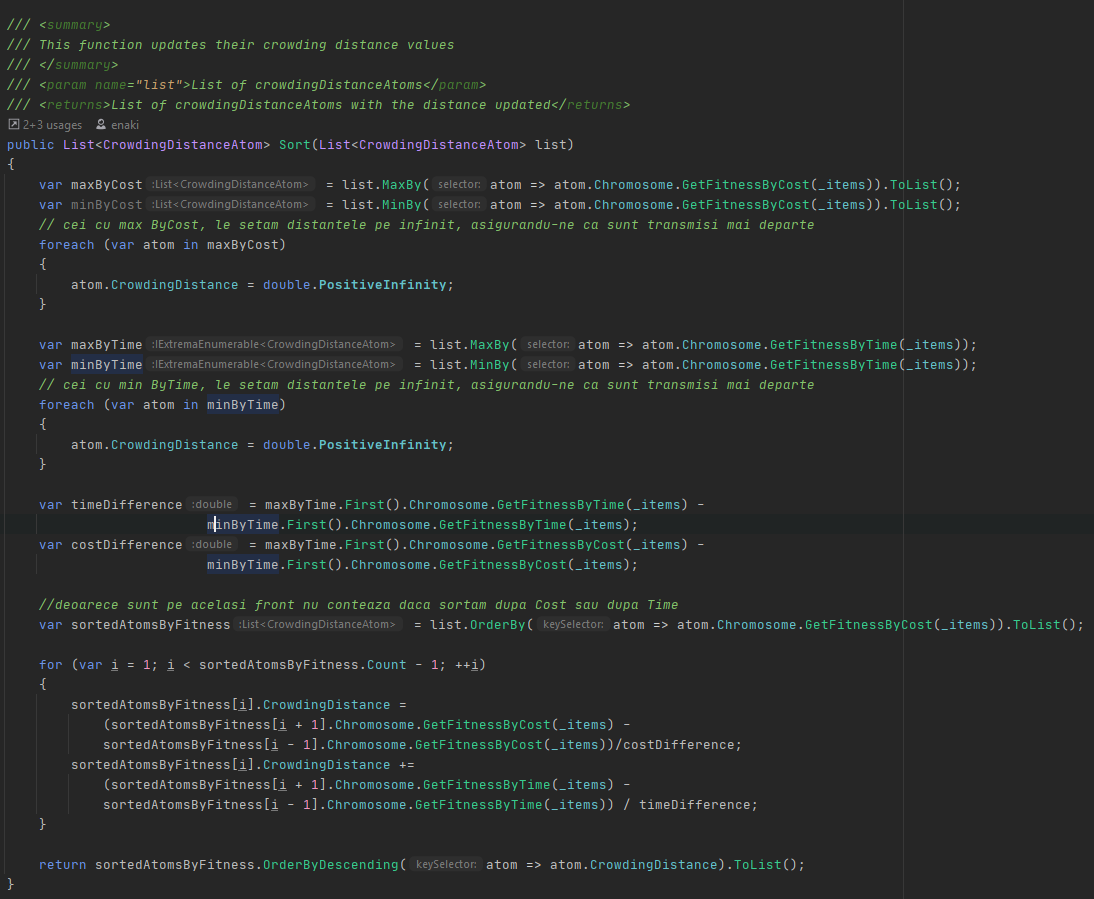
## 4.5 Selecția prin turnir

1. Primește o listă de cromozomi
2. Sunt selectați noi indecși random din lista de cromozomi și este returnam cromozomul dominant
3. Dacă nici unul dintre cromozomi nu este dominant, este returnat cu o probabilitate de 50% unul dintre ei



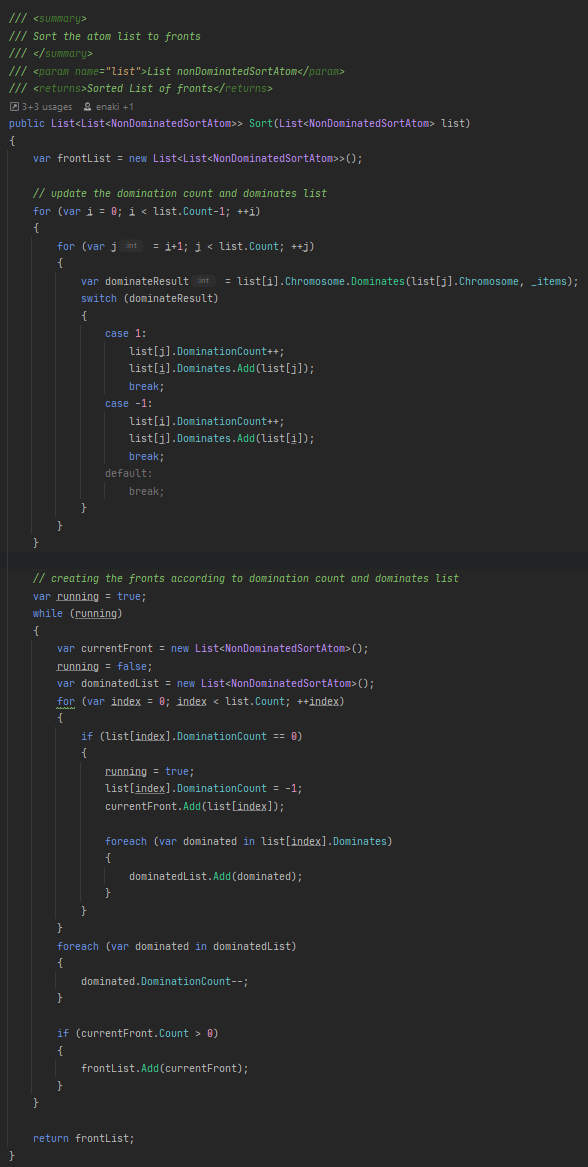
## 4.6 Crowding Distance Sort

1. Primește o listă de modele de tip CrowdingDistanceAtom
2. Atomii care au cost maxim și cei care au timp minim le sunt setați distanța de aglomerare pe Infinit, asigurându-ne că vor fi transmiși în populația următoare
3. Este calculată crowding distance-ul pentru fiecare atom în parte și se returnează o listă descrescătoare după distanța de aglomerare



## 4.7 Non Dominated Sort

1. Primește o listă de atomi de tip NonDominatedSortAtom
2. Se updatează indicele de tip DominationCount si lista de Dominates din cadrul unui atom
3. Se separă pe front-uri lista de atomi folosindu-ne de parametrii specificați mai sus
4. Se returnează lista de front-uri

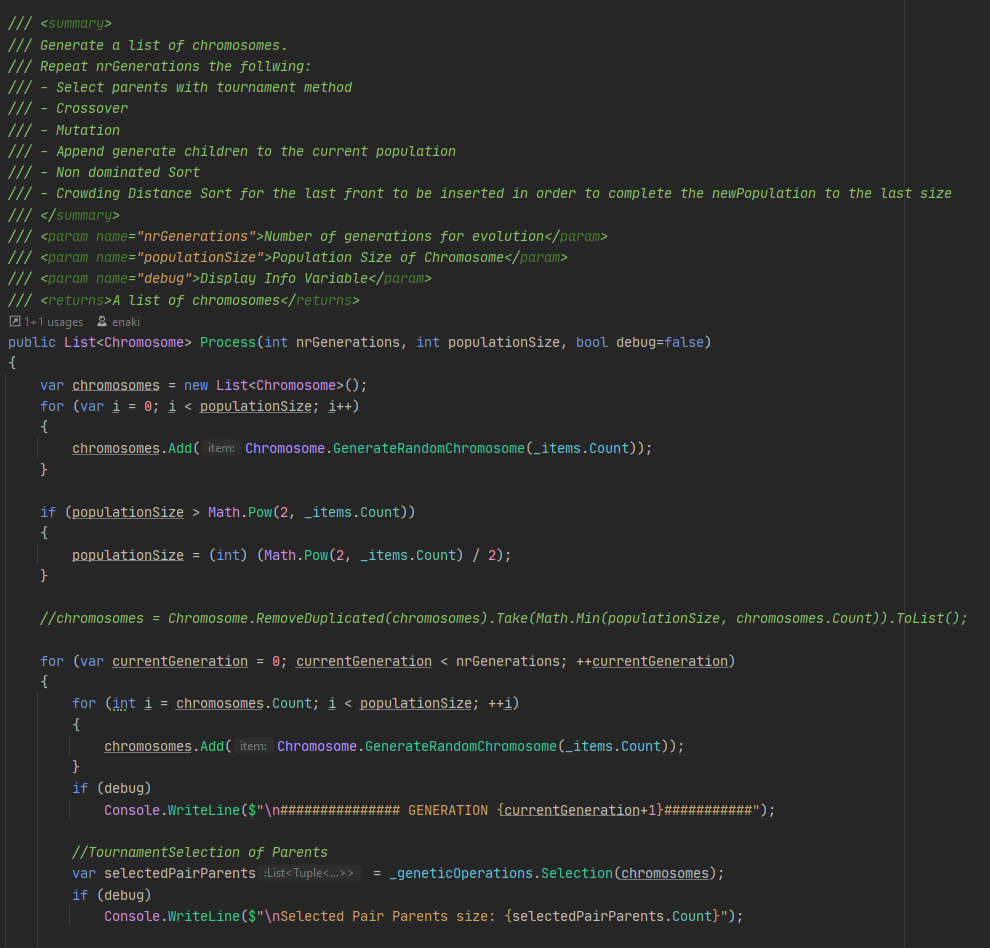


## 4.8 Algoritmul NSGA-II

1. Primește numărul de generații și dimensiunea populației
2. Se generează populația inițială de cromozomi
3. La fiecare generație se repetă următoarele operații:

* selectează perechile de părinți prin turnament
* încrucișare și generare de copii
* operatorul de mutare pe copii și inserarea lor în populație
* sortarea non dominantă pe lista de cromozomi
* sortarea de aglomerare pentru ultimul front de inserat în populație necesar stabilirii dimensiunii inițiale a populației

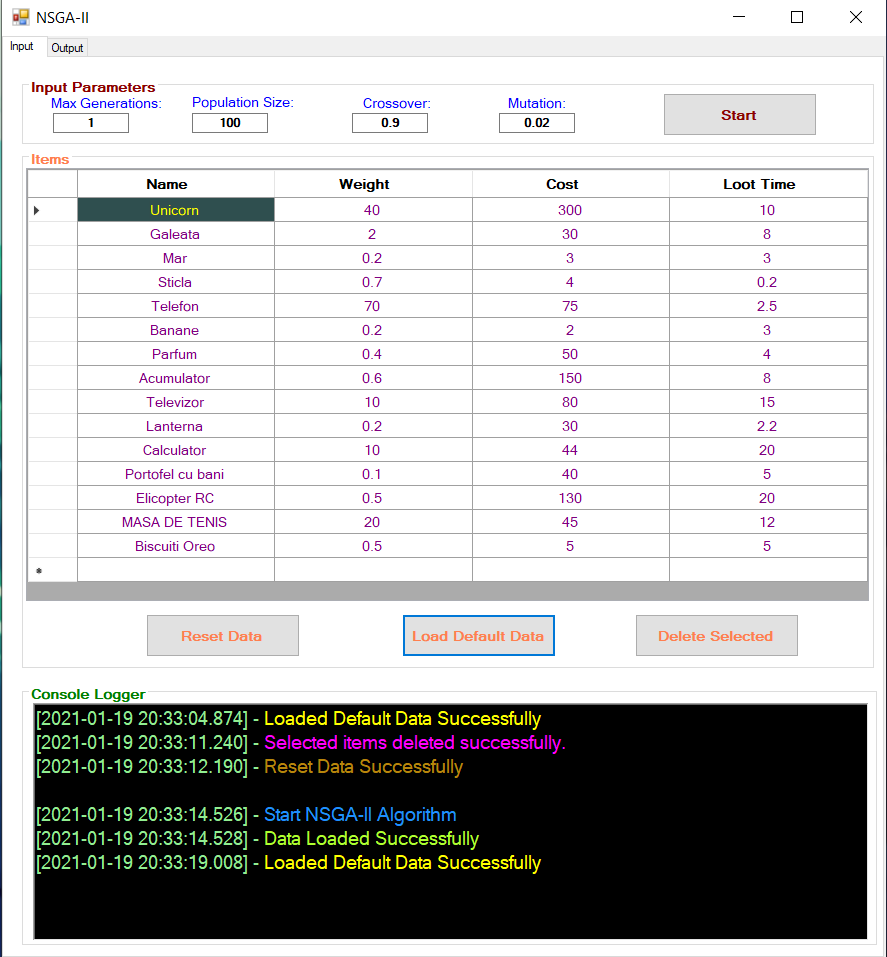
1. Returnează lista de cromozomi în urma evoluției





# Capitolul 5. GUI

Partea de interfață grafică a fost implementată folosind Windows Forms, iar pe partea de analiză a rezultatului s-a folosi pachetul **Live Charts**.



Au fost tratate și erorile de input, user-ul fiind atenționat de date invalide în console logger.



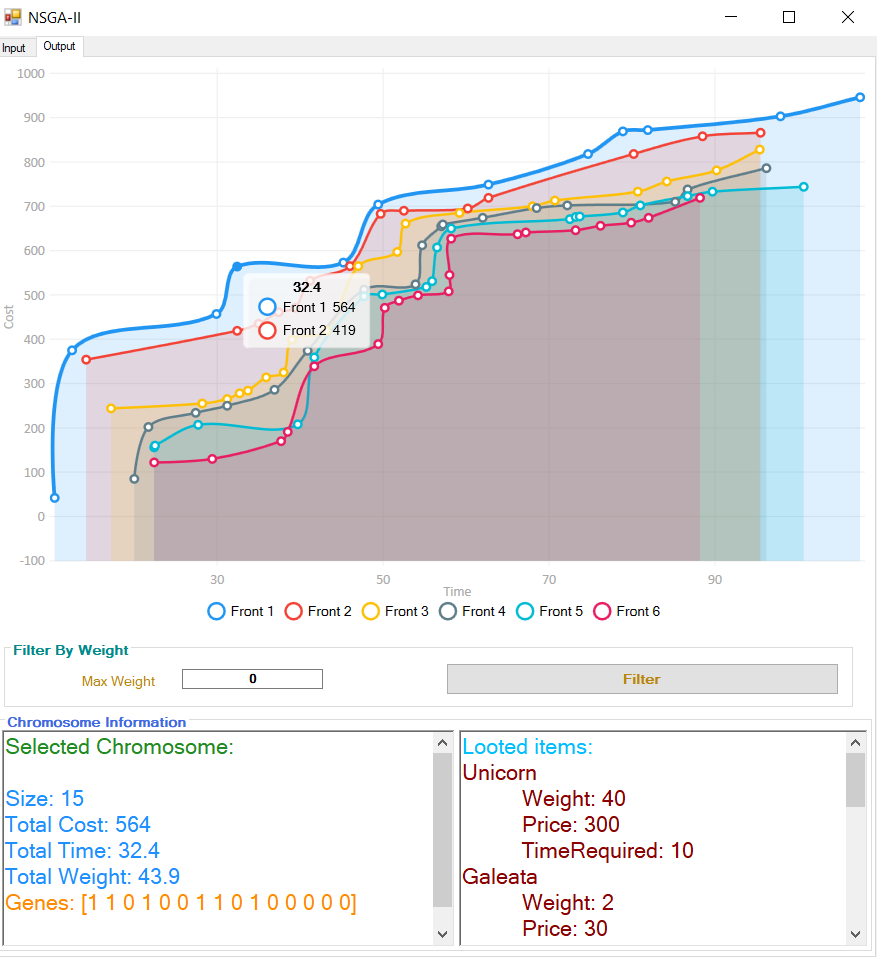
Funcționalități interfața grafică:

* **console logger -**  pentru înregistrarea fiecărei operații efectuate
* **adăugare item în lista de itemi**
* **resetare lista de itemi**
* **loadare lista de itemi de default**
* **șterge itemi selectați**
* **setare parametri algoritm**
* **vizualizare front-uri după executarea algoritmului**
* **filtrare cromozomi după greutate**
* **vizualizare detalii a unui cromozom din populația finală**

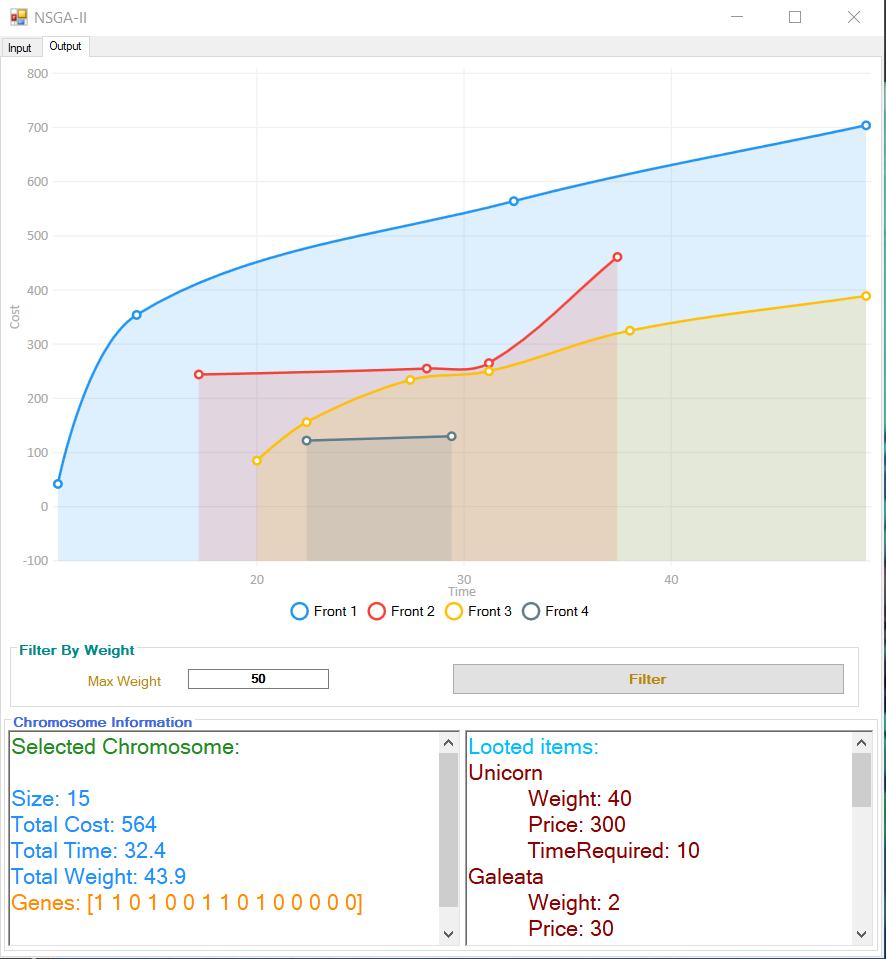
# Capitolul 6. Rezultatele obținute prin rularea programului în diverse situații

Datele de default pot fi vizualizate în figura de la capitolul GUI.

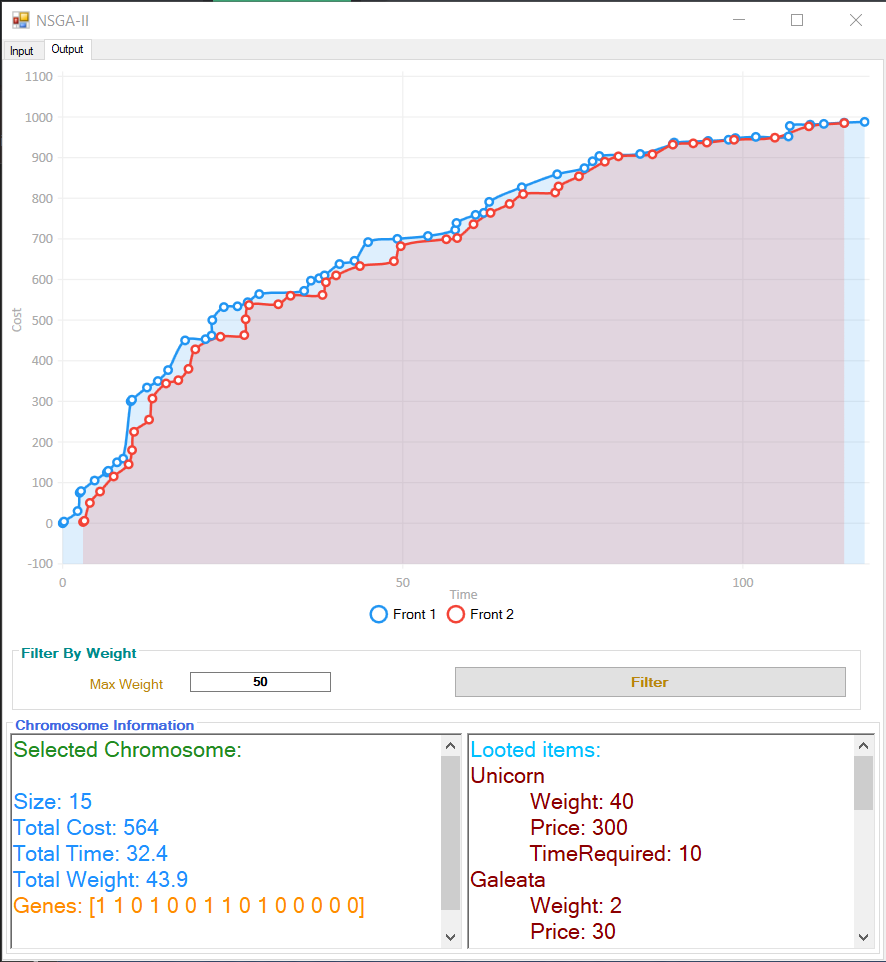
1. **Rulare algoritm cu date inițiale de default, 1 singură generație, fără filtrare după greutate**

****

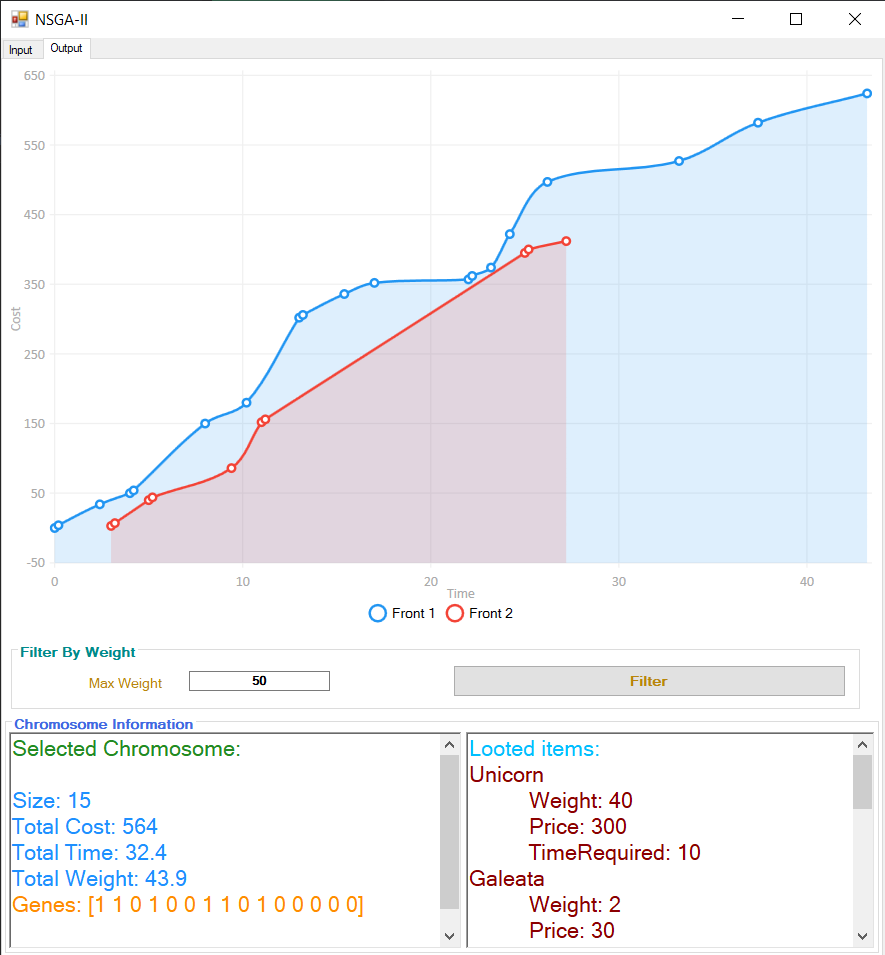
1. **Rulare algoritm cu date inițiale de default, 1 singură generație, cu filtrare greutate < 50**



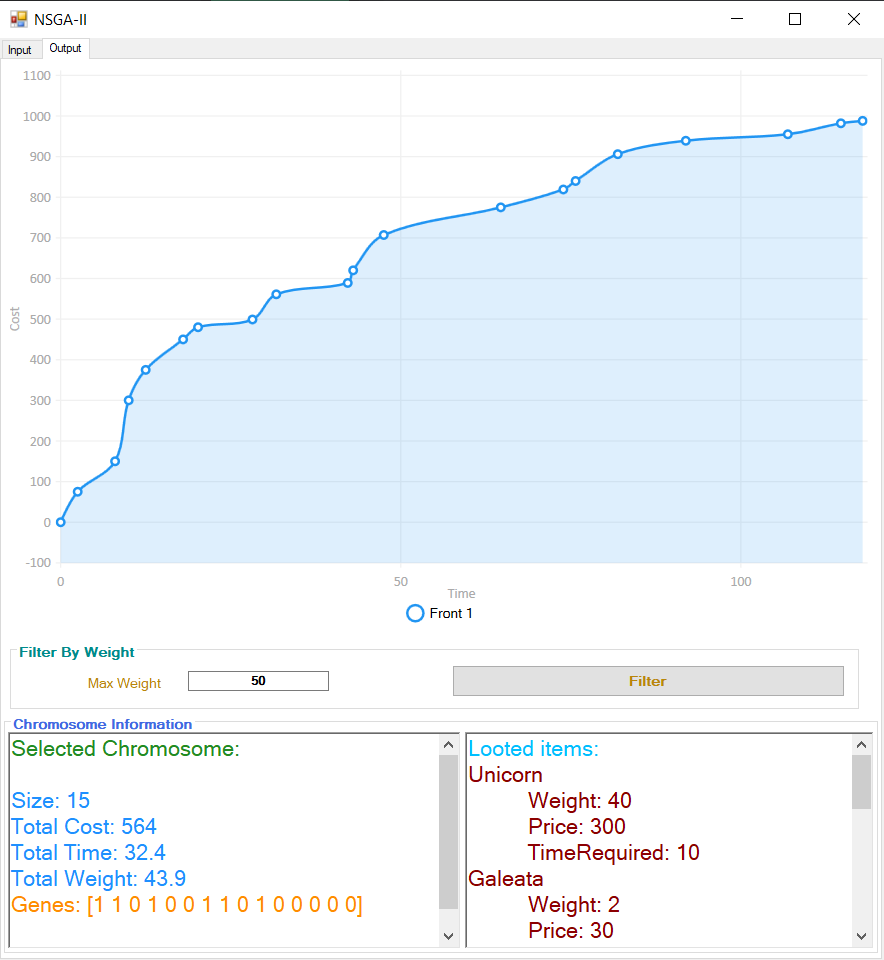
1. **Rulare algoritm cu date inițiale de default, 100 de generații, fără filtrare după greutate**



1. **Rulare algoritm cu date inițiale de default, dimensiunea populației de 20 indivizi, 100 de generații, filtrare după greutate <= 50**



1. **Rulare algoritm cu date inițiale de default, dimensiunea populației de 20 indivizi, 100 de generații, fără filtrare după greutate**



# Capitolul 7. Concluzii

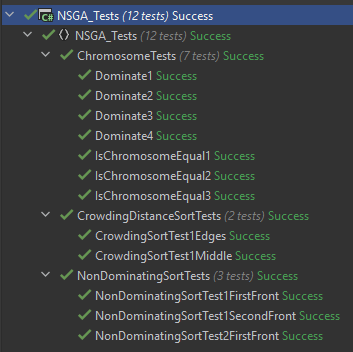
Algoritmul NSGA-II reprezintă o metodă eficientă de optimizare multi-obiectiv care dă rezultate foarte bune în combinație cu algoritmii genetici.

**Observația 1:** Se observă tendința ca cu cât mărim numărul de generații, numărul de fronturi să se reducă.

**Observația 2:** Se observă tendința ca cu cât micșorăm numărul de indivizi, numărul de fronturi să se reducă.

Aceste rezultate sunt apropiate de cele reale, iar posibilitatea reprezentării grafice a soluțiilor ajută la ușurarea luării unei decizii în funcție de parametrii de interes.

# Capitolul 8. Testare

Au fost testate:

* funcțiile de Dominates și IsEqual din clasa Chromosome
* sortarea „non dominated sort”
* sortarea „crowding distance sort”

# 

# Capitolul 9. Bibliografie

* <https://www.sciencedirect.com/science/article/pii/S1877705811022466>
* <http://florinleon.byethost24.com/lab_ia.html>
* <https://arxiv.org/pdf/1901.00577.pdf>
* <https://ocw.mit.edu/courses/institute-for-data-systems-and-society/ids-338j-multidisciplinary-system-design-optimization-spring-2010/lecture-notes/MITESD_77S10_lec14.pdf>
* <https://web.stanford.edu/group/sisl/k12/optimization/MO-unit5-pdfs/5.8Pareto.pdf>

# Capitolul 10. Roluri

|  |  |
| --- | --- |
| **Student** | **Rol** |
| Enachi Vasile | Elaborarea modului de algoritm si CLI |
| Muraru Alexandru | Elaborarea GUI și integrarea cu algoritmul, modulul de test |
| Ilioi Alexandru | Documentație și proiectarea modulelor |