Facultatea de Inginerie “Hermann Oberth”

Universitatea "Lucian Blaga" din Sibiu

**Proiect An IV Calculatoare**

**La disciplina Simularea şi Optimizarea Arhitecturilor de Calcul**

**Autori**: Ana-Maria Aftenie, Andreea-Maria Purta; semigrupa:241/1

**Indrumatori**: ing. Teodora Flavia Antonie

# **Tema proiectului**

D2. Implementarea unui Automatic Design Space Explorer aferent unei arhitecturi superscalare (Hatfield – vezi punctul A), pentru determinarea optimului din punct de vedere a ratei de procesare folosind algoritmi de căutare locali (în vecinătatea unei configurații microarhitecturale) de tip Hill Climbing, Simulated Annealing

Cuprins

[Tema proiectului: 3](#_Toc29053004)

[1 NOTIUNI TEORETICE GENERALE 7](#_Toc29053005)

[1.1 AUTOMATIC DESIGN SPACE EXPLORER. 7](#_Toc29053006)

[2 Strategii de cautare locala 8](#_Toc29053008)

[2.1 HILL CLIMBING 8](#_Toc29053009)

[2.1.2 HILL CLIMBING STOCHASTIC 9](#_Toc29053010)

[2.2 SIMULATED ANNEALING 10](#_Toc29053011)

[3 INSTRUCTIUNI DE UTILIZARE 11](#_Toc29053012)

[3.1 SIMULATORUL PSATSIM 11](#_Toc29053013)

[3.2 GHID DE UTILIZARE 11](#_Toc29053014)

[4 GHID DE DEZVOLTARE 14](#_Toc29053015)

[4.1 DIAGRAMA CLASELOR 14](#_Toc29053016)

[5 REZULTATELE SIMULARII 16](#_Toc29053017)

# NOTIUNI TEORETICE GENERALE

## AUTOMATIC DESIGN SPACE EXPLORER.

Un Automatic Design Space Explorer se refera la acel sistem prin care se poate realiza, analiza sistematic si reduce configuratiile nedorite bazate pe niste parametrii de interes. Acest Explorer este folosit pentru optimizarea sistemelor. Automatic Design Space Explorer nu incarca manual parametrii, ci ii genereaza automat, impreuna cu dezvoltarea si optimizarea lor.

Explorarea se face in doi pasi: cautarea si evaluarea obiectivelor.

De cele mai multe ori, cautarea se face prin algoritmi cu bucla inchisa, care incearca sa gaseasca seturi de parametrii optimi, fara a fi nevoie sa se evalueze performanta tuturor combinatiilor. Acesti algoritmi se folosesc de rezultatele simularilor anterioare, pentru a sti ce combinatii sa evalueze in continuare.

Un exemplu ar fi algoritmii genetici, unde, dupa un set de combinatii initiale (alese aleatoriu) care au fost analizate, algoritmul creeaza noi design-uri prin combinarea valorilor parametrilor de la combinatiile cele mai bune gasite in iteratia anterioara. Acest algoritm continua pana cand nu se mai poate face nicio optimizare sau se termina resursele sistemului. Un dezavantaj al acestui mod de simulare, ar fi ca nu se garanteaza returnarea unui optim global.

A doua etapa este evaluarea obiectivelor. De obicei, cand se incearca optimizarea unui sistem, se tine cont de mai multe obiective. Acest lucru poate duce la decizii grele de optimizare, deoarece, este posibil ca optimizarea unui obiectiv sa fie in detrimentul altuia.

In aceasta lucrare, obiectivele ce sunt tratate sunt performanta sistemului, precum si energia consumata. Adica se doreste determinarea optimului din punct de vedere a ratei de procesare.

# Strategii de cautare locala

## Hill Climbing

Căutarea avansează în direcţia îmbunătăţirii valorii stărilor succesor până când se atinge un optim.

Criteriul de acceptare a unei noi soluţii : cel mai bun vecin al soluţiei curente mai bun decât soluţia curentă.

Îmbunătăţire prin :

* Maximizarea calităţii unei stări : steepest ascent HC
* Minimizarea costului unei stări : gradient descent HC

Exemplu :

Text, letter

Description automatically generated

Analiza căutării

* convergenţa spre optimul local

Avantaje

* simplu de implementat
* se poate folosi usor pentru a aproxima soluţia unei probleme când soluţia exactă este dificil sau imposibil de găsit
* nu necesită memorie (nu se revine în starea precedentă)

Dezavantaje

* Funcţia de evaluare (eurisitcă) poate fi dificil de estimat
* Dacă se execută foarte multe mutări algoritmul devine ineficient
* Dacă se execută prea puţine mutări algoritmul se poate bloca
  + Într-un optim local (nu mai poate “coborî” din vârf)
  + Pe un platou – zonă din spaţiul de căutare în care funcţia de evaluare este constantă
  + Pe o creastă – saltul cu mai mulţi paşi ar putea ajuta căutarea

### HILL CLIMBING STOCASTIC

Este o varianta de cautare locala Hill Climbing, care presupune alegerea aleatoare a unui successor.

Se poate realiza prin 2 metode:

* HC cu prima optiune:
  + Generarea aleatoare a succesorilor până la întâlnirea unei mutări neefectuate
* HC cu repornire aleatoare: beam local search
  + Repornirea căutării de la o stare iniţială aleatoare atunci când căutarea nu progresează

## SIMULATED ANNEALING

Succesorii stării curente sunt aleşi şi în mod aleator

* Dacă un succesor este mai bun decât starea curentă
  + atunci el devine noua stare curentă
  + altfel succesorul este reţinut doar cu o anumită probabilitate
* Se permit efectuarea unor mutări “slabe” cu o anumită probabilitate p
  + evadarea din optimele locale
* Frecvenţa acestor mutări “slabe” şi mărimea lor se reduce gradual prin scăderea temperaturii
  + T = 0 => hill climbing
  + T = infinit => mutările “slabe” sunt tot mai mult executate
* Soluţia optimă se identifică dacă temperatura se scade treptat (“slowly”)
* Criteriul de acceptare a unei noi soluţii
  + Un vecin aleator mai bun sau mai slab (cu probabilitatea p) decât soluţia curentă

Analiza cautarii :

* Convergenţa (complet, optimal) lentă spre optimul global

Avantaje :

* Algoritm fundamentat statistic = garantează găsirea soluţiei optime, dar necesită multe iteraţii
* Uşor de implementat
* În general găseşte o soluţie relativ bună (optim global)
* Poate rezolva probleme complexe (cu zgomot şi multe constrângeri)

Dezavantaje :

* Algoritm încet – convergenţa la soluţie durează foarte mult timp
* Compromis între calitatea soluţiei şi timpul necesar calculării ei
* Depinde de anumiţi parametri (temperatura) care trebuie reglaţi
* Nu se ştie dacă soluţia oferită este optimă (local sau gloabal)
* Calitatea soluţiei găsite depinde de precizia variabilelor implicate în algoritm

# INSTRUCTIUNI DE UTILIZARE

## SIMULATORUL PSATSIM

Este necesara instalarea simulatorului PSATSim. Acesta va rula in background, prin aplicatia consola.

Aplicatia se ruleaza prin comanda “psatsim\_con.exe”, alaturi de cateva argumente necesare: fisierul xml de intrare, fisierul xml de iesire, si tipul de optimizare.

Fisierul xml de intrare contine toate variatiile de configuratii generate, iar prin fisierul xml, citim rezultatele simularilor.

In aceasta lucrare, s-a folosit doar simularea generala.

## GHID DE UTILIZARE

* La rulare, se va deschide urmatoarea fereastra, care are urmatoarele componenete:

Chart area unde se vor afisa rezultatele sub forma de grafic

Alegerea algoritmului genetic

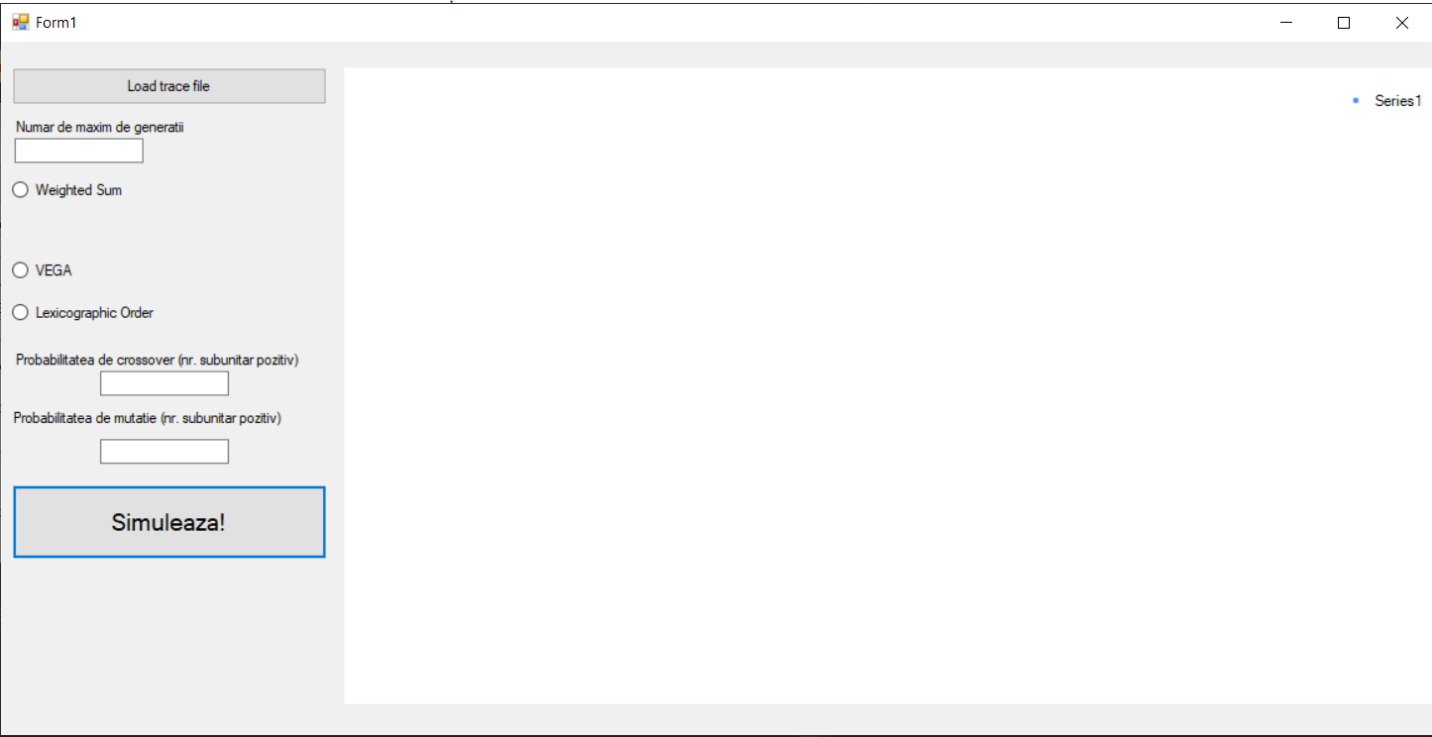
Textbox pentru a introduce nr. Maxim de generatii

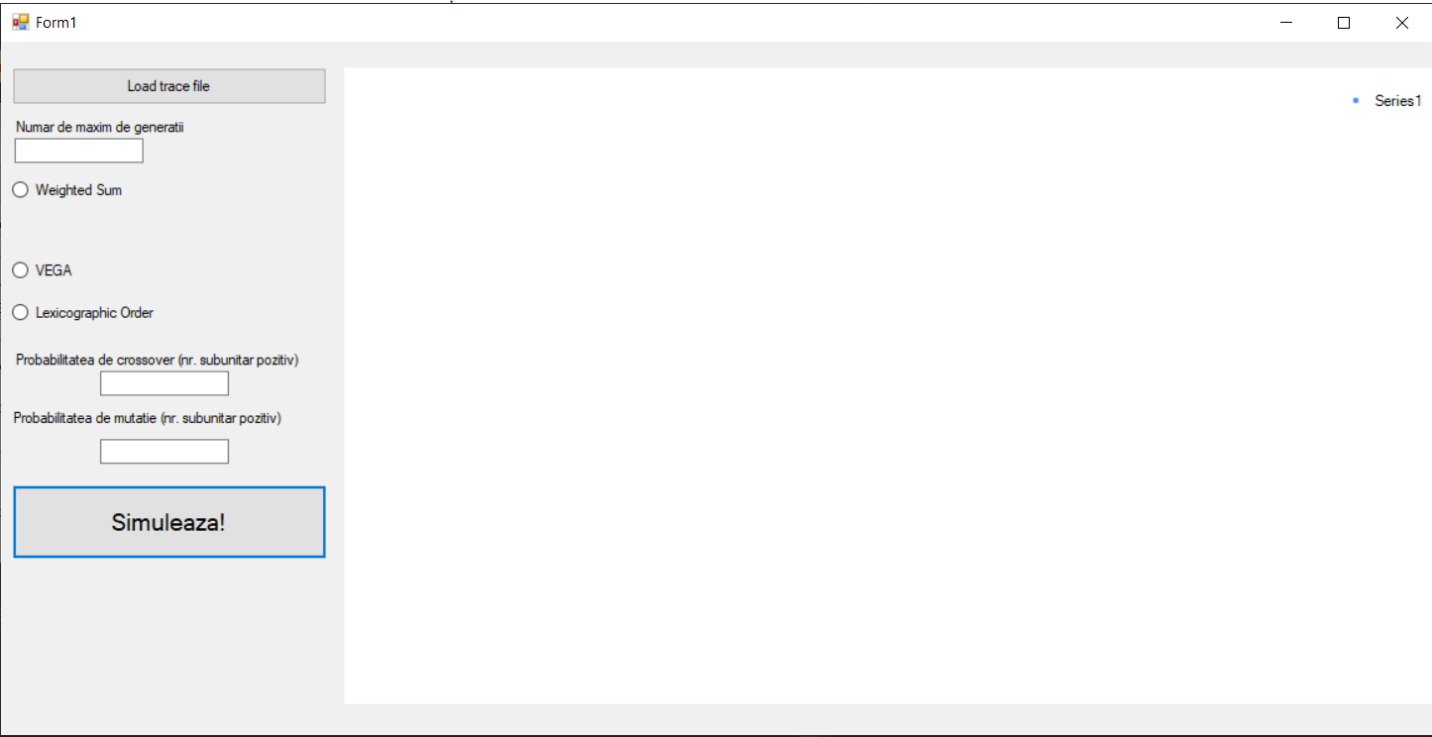
Buton de incarcare a fisierului“Trace”



Buton care porneste simularea

Textbox-uri pentru introducerea probabilitatilor de crossover si mutatie

Pasi:

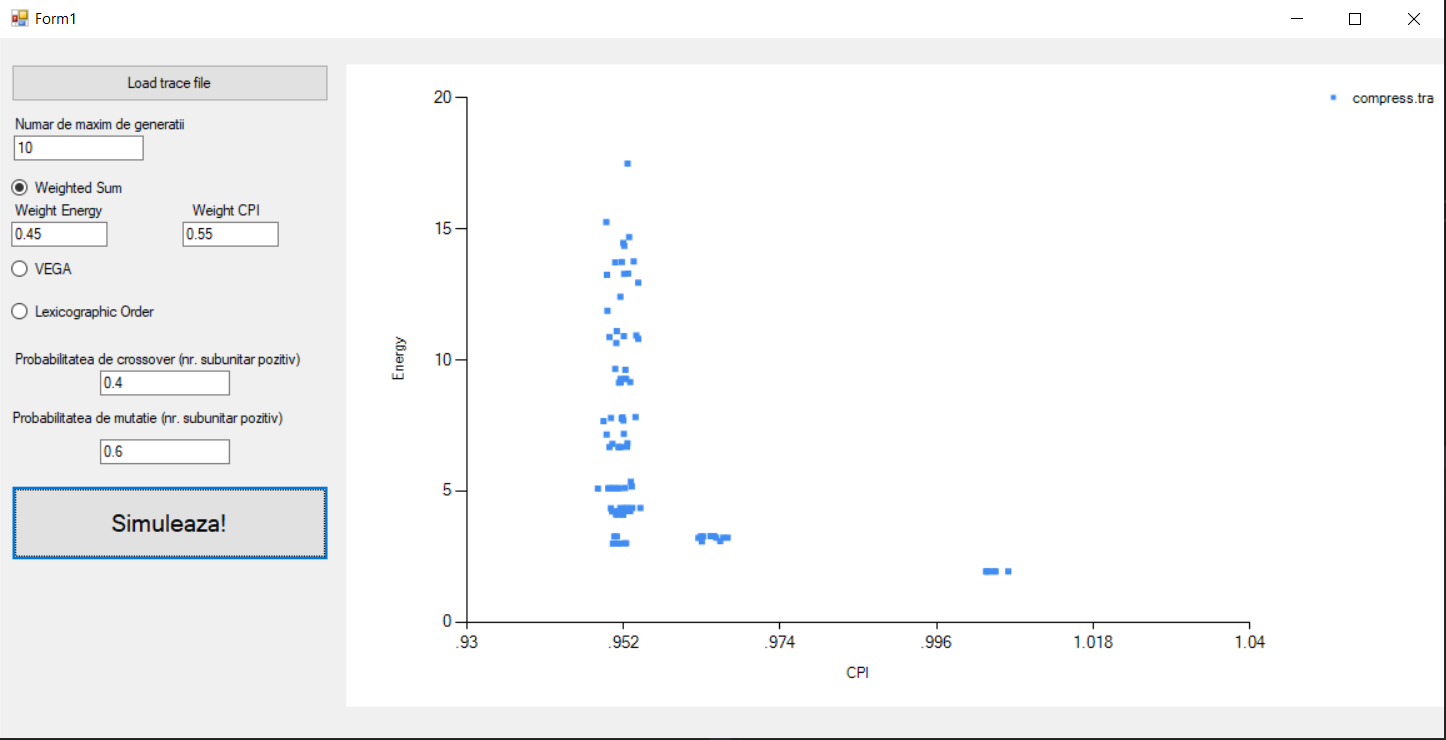
1. Se incarca un fisier “trace”
2. Se scrie numarul maxim de generatii (numar intreg pozitiv)
3. Se alege algoritmul cu care se doreste simularea



1. Se inscriu cele doua probabilitati (numere subunitare)



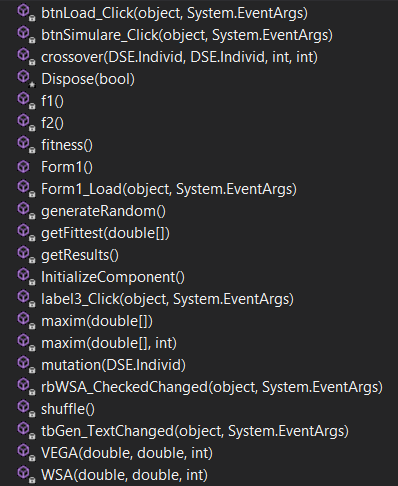
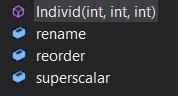
1. Se apasa butonul “Simuleaza” si se asteapta afisarea rezultatelor.



# GHID DE DEZVOLTARE

## DIAGRAMA CLASELOR

Form1 class



Individ class

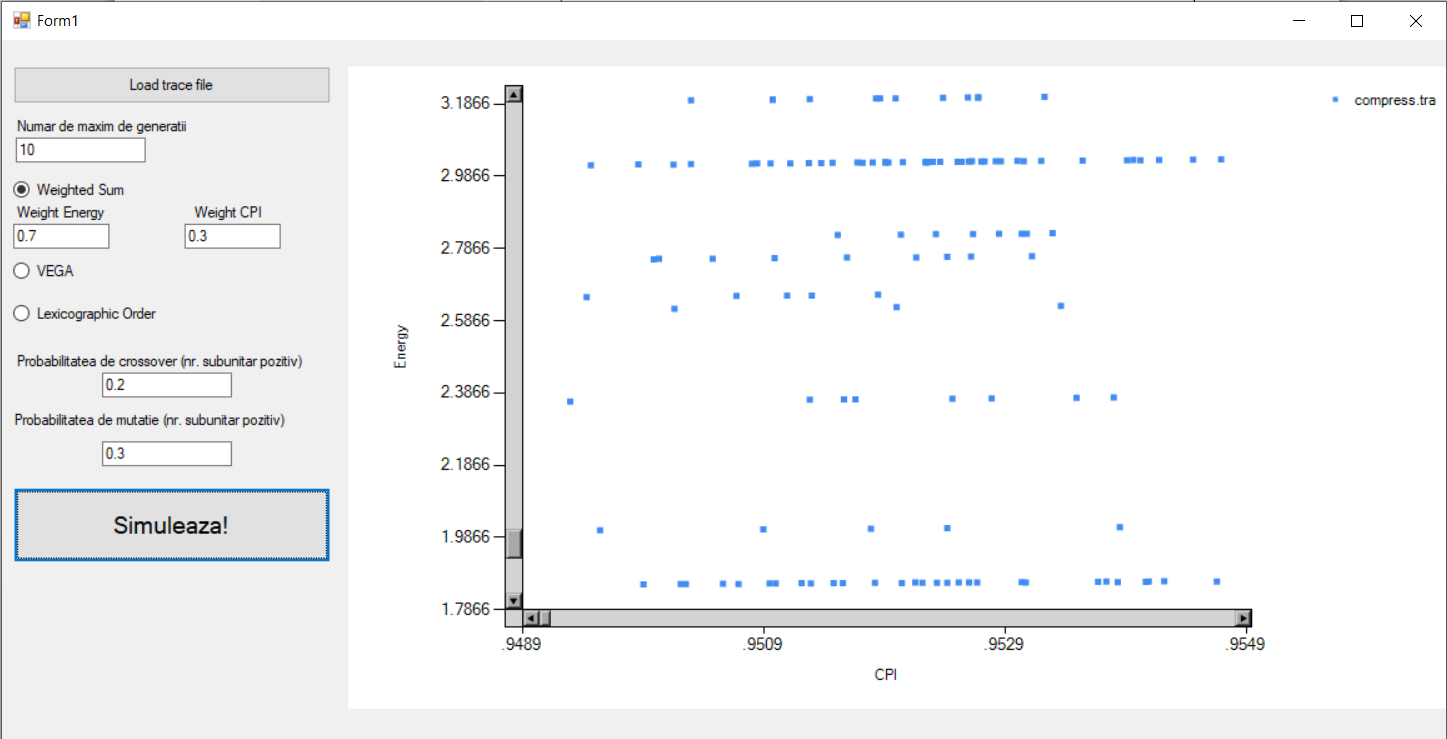
**CLASA INDIVID:**

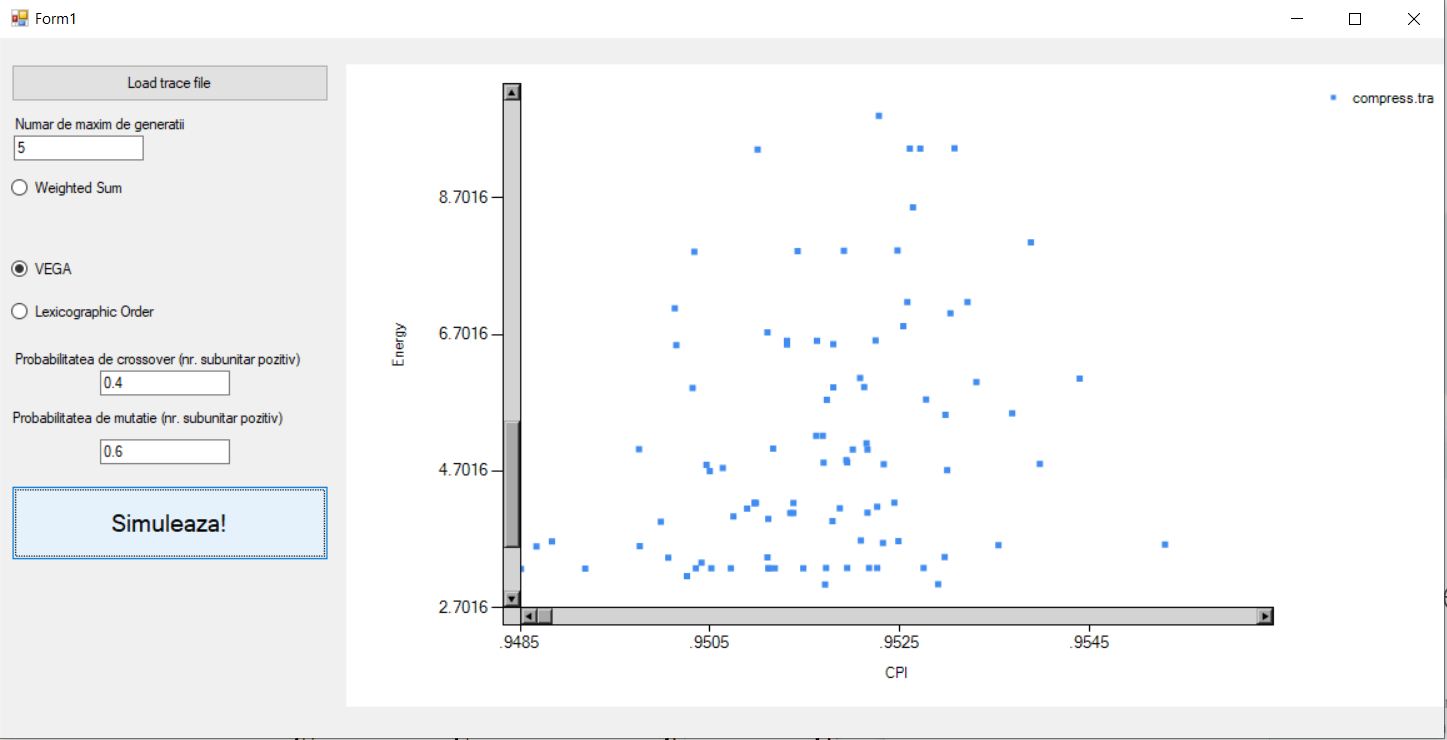
* Contine constructorul Individ(), care initializeaza indivizii populatiei, adica o configuratie, cu urmatoarelel campuri: superscalar factor, reorder units, rename units;

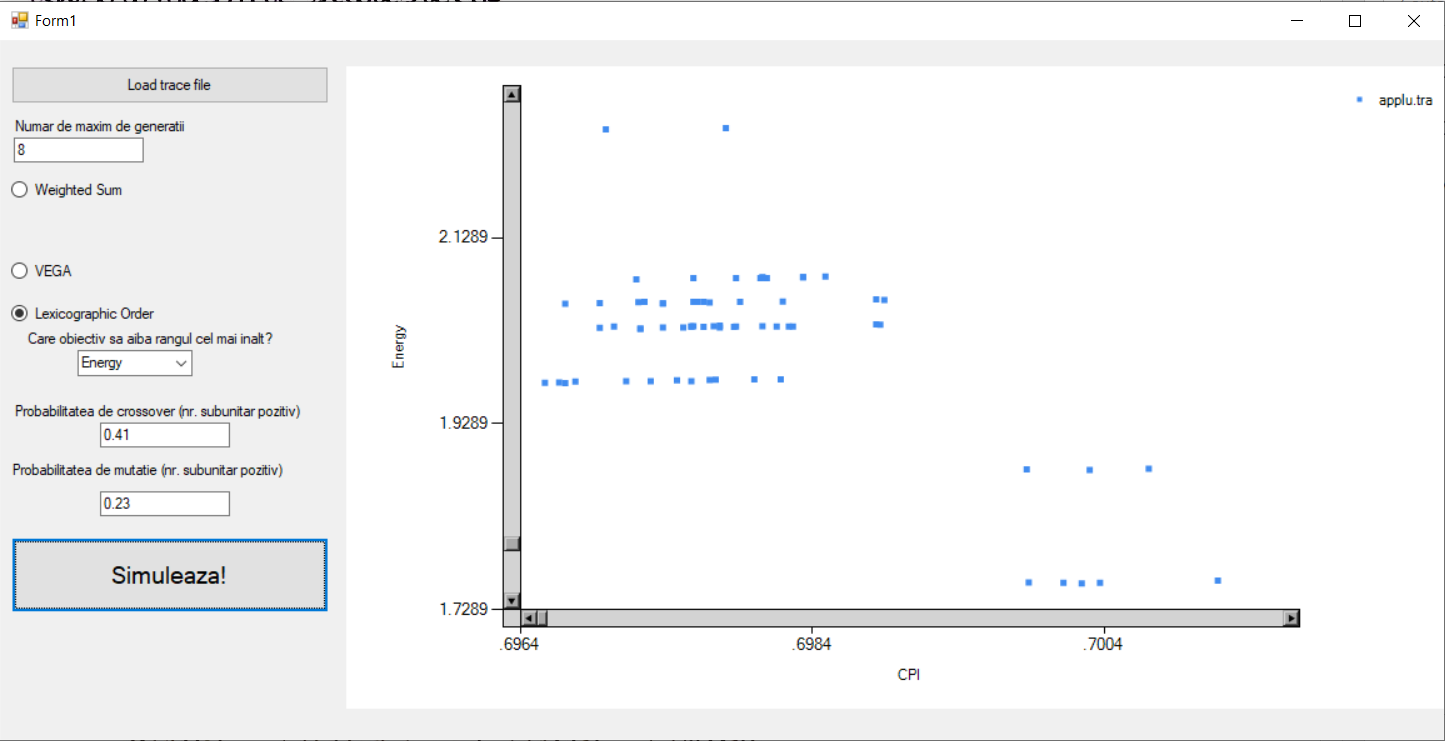
**CLASA FORM1** (care contine toate metodele utilizate):

* crossover() – metoda ce realizeaza operatia de crossover cu punct simplu. Punctul se allege aleatoriu de fiecare data cand e apelata.
* f1() - returneaza subpopulatia in functie de obiectivul Energie (pentru VEGA) – le allege pe cele mai bune
* f2() - returneaza subpopulatia in functie de obiectivul Performanta (pentru VEGA) – le allege pe cele mai bune
* fitness() – calculeaza fitness-ul insumat pentru fiecare individ (pentru Wieghted Sum)
* generateRandom() – genereaza random o populatie
* getFittest() – returneaza o subpopulatie cu cei mai optimi indivizi in functie de fitness-ul calculate pe baza Weighted Sum
* getResults() – simuleaza configuratiie prin generarea fisierului xml de input, genereaza comanda pentru simulatorul PSATSim, citeste rezultatele din fisierul xml de iesire si le afiseaza pe grafic
* maxim() – calculeaza maximul dintr-un array in vederea alegerii celei mai proaste configuartii
* mutation() – realizeaza mutatia indivizilor cu numere alese random
* VEGA() – implementeaza algoritmul VEGA
* WSA()– implementeaza algoritmul Wieghted Sum
* LexicographicOrder() – implementeaza algoritmul Lexicographic Order

# REZULTATELE SIMULARII







**Concluzii:**

* Se poate observa ca la VEGA, simularea aduce optimizari mai bune decat la Weighted Sum, deoarece VEGA incearca optimizarea celor doua obiective simultan, pe cand Weighted Sum adduce totul la un singur obiectiv in functie de niste ponderi date de noi.
* Cele mai bune optimizari sunt cele de pe fronturile Pareto (cele desenate cu rosu), toate celelalte fiind dominate de acestea.
* Se poate observa la Lexicographic Order cum exista o tendinta ca rezultatele sa mearga in extreme. Acest lucru se intampla, deoarece cele doua obiective se optimizeaza pe rand.
* Lexicographic Order gaseste doar optimul local, dar e foarte posibil ca optimul global sa fie pierdut in algoritm. Deoarece ia in considerare, la un moment dat, doar un obiectiv, algoritmul converge la un maxim local.