UNIVERSITATEA “ALEXANDRU IOAN CUZA” IAȘI
FACULTATEA DE INFORMATICĂ

Algoritmi pentru testarea izomorfismului grafurilor

Coordonator științific:
Lect. Dr. Cristian Frăsinaru

Student:
Ignat Gabriel-Andrei

Cuprins

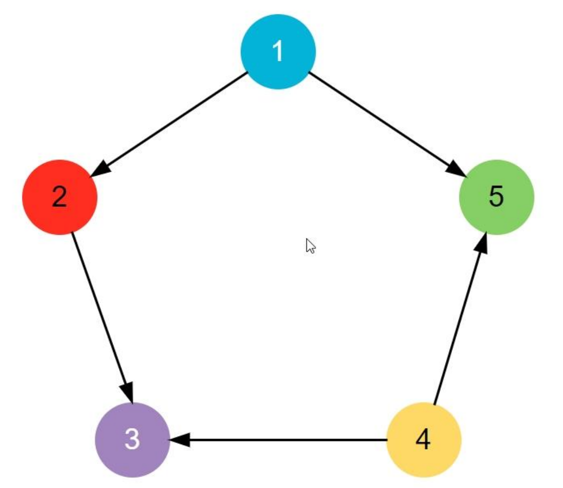
01

Introducere

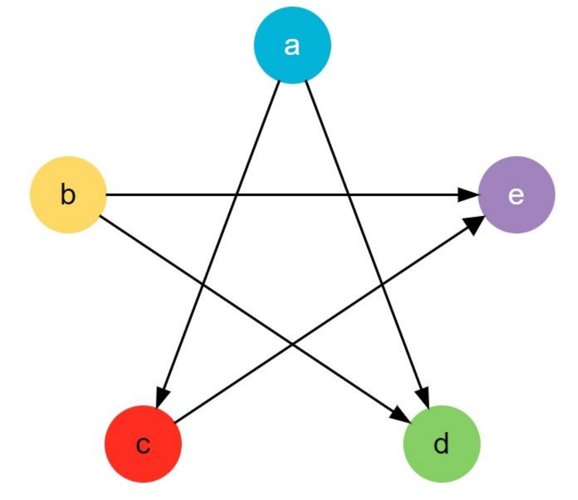
Definiție:

Două grafuri $\mathbf{G_1}$ și $\mathbf{G_2}$ sunt izomorfe dacă există o funcție bijectivă $\mathbf{f: V(G_1) \rightarrow V(G_2)}$ a.i. $\exists$ $(u,v) \in E(G_1)$ ddacă $\exists$ $(f(u),f(v)) \in E(G_2)$.

Exemplu:



Graf G_1

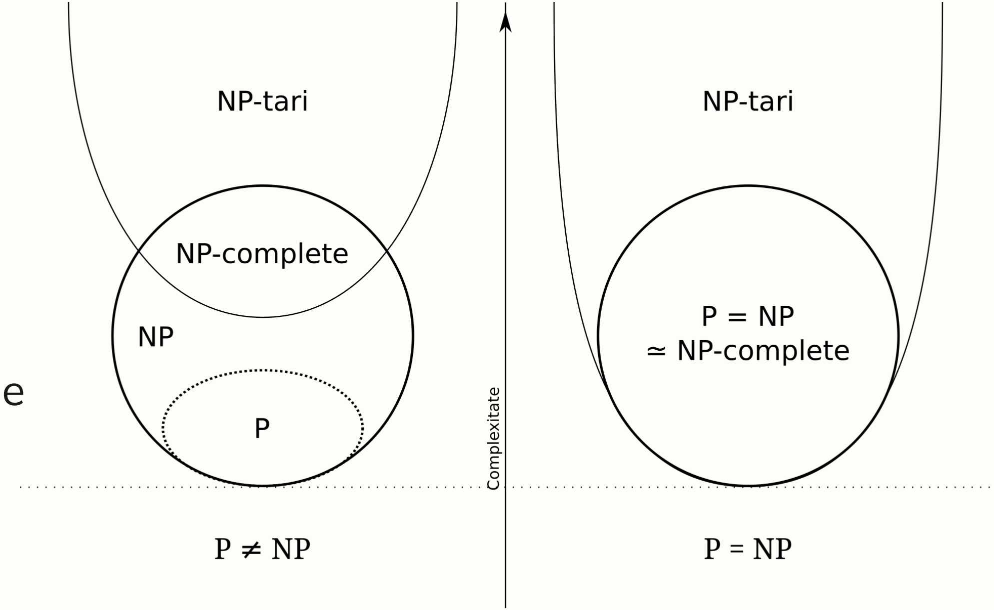


Graf G_2

$f(1) = a$
$f(2) = c$
$f(3) = e$
$f(4) = b$
$f(5) = d$

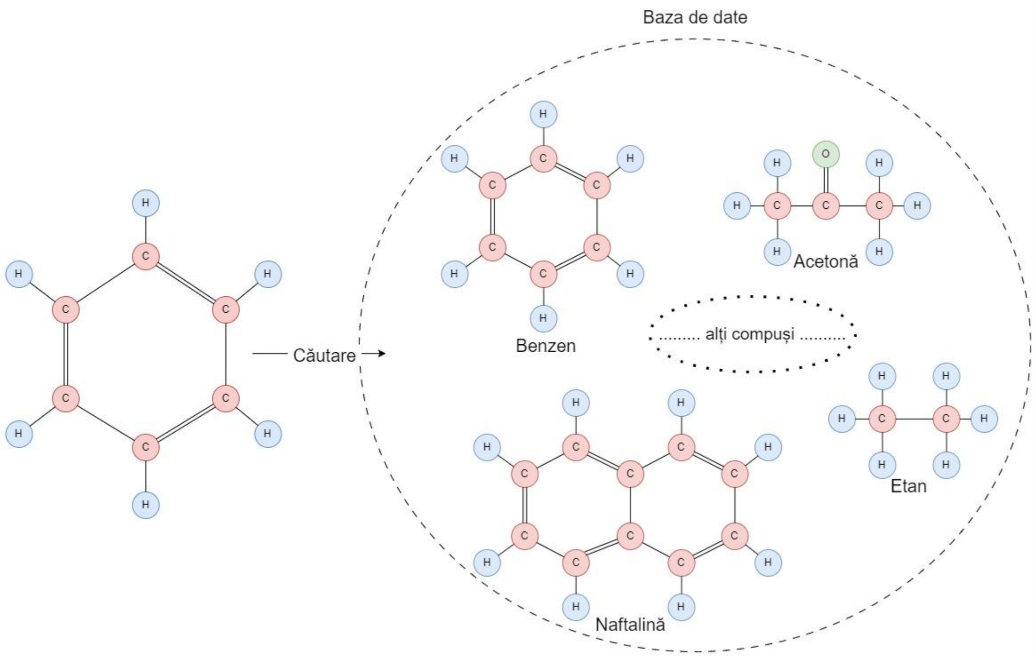
Complexitate

- aparține clasei **NP**;
- nu a fost demonstrat că aparține clasei problemelor **NP**-complete;
- nu a fost descoperit un algoritm de complexitate polinomială;
- NP-intermediară.



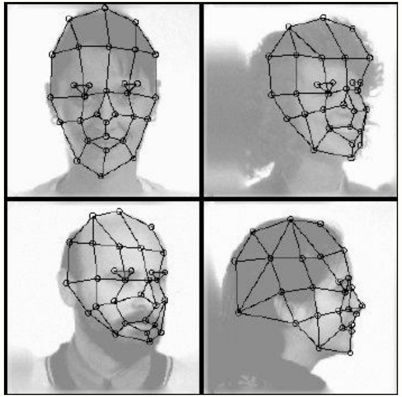
Aplicații: Chimie organică

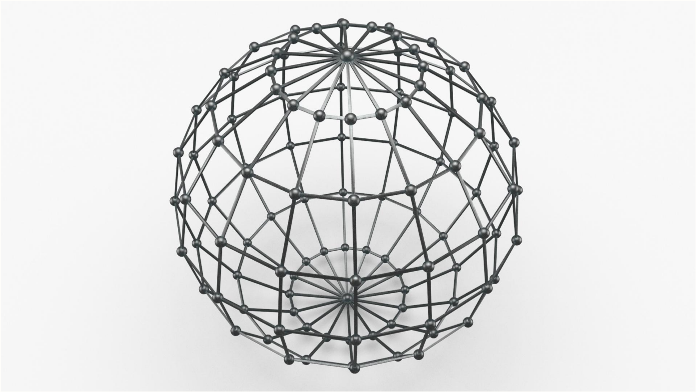
- reprezentare ușoară a compușilor sub forma de graf;
- căutare unui compus într-o bază de date;
- impact: industria farmaceutică, a materialelor și alimentară.



Aplicații: Recunoaștere de modele

- recunoașterea feței;
- recunoașterea amprentei;
- recunoașterea obiectelor 2D/3D.



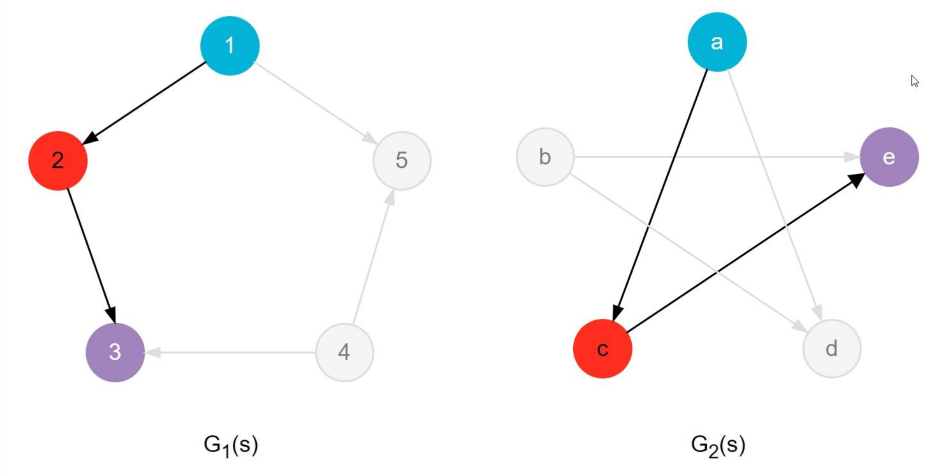


Algoritmi

Cazul general: Ullman

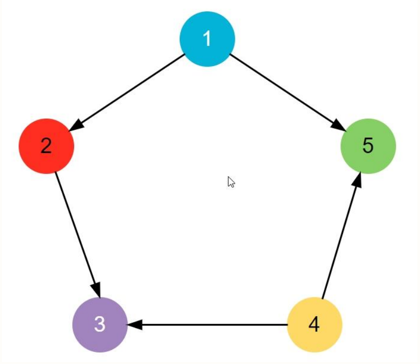
- inspirat din domeniul inteligenței artificiale;
- mapări consistente cu problema izomorfismului;
- matrice de compatibilitate;
- căutare de tip DFS;
- procedură de rafinare a candidaților.

Stare parțială:

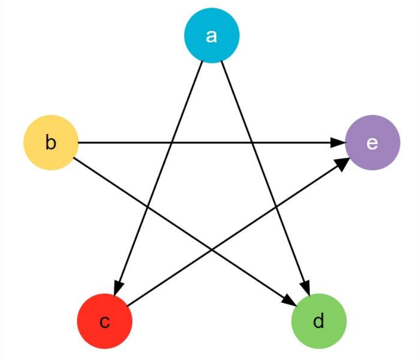


- $M(s) = \{(1,a), (2,c), (3,c)\}$
- $M_1(s) = \{1, 2, 3\}$
- $M_2(s) = \{a, c, e\}$
- subgrafurile induse $G_1(s)$ și $G_2(s)$ sunt izomorfe

Matricea de compatibilitate:



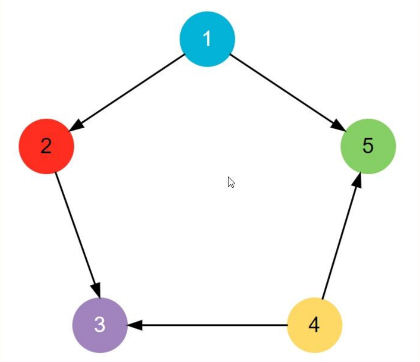
G_1



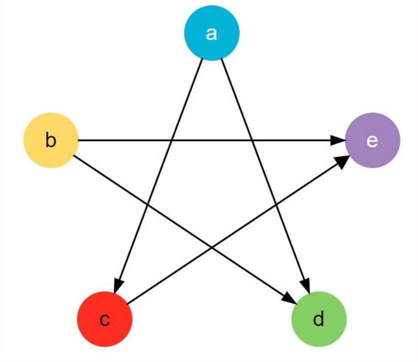
G_2

	a	b	c	d	e
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	0	0	0
5	0	0	0	1	1

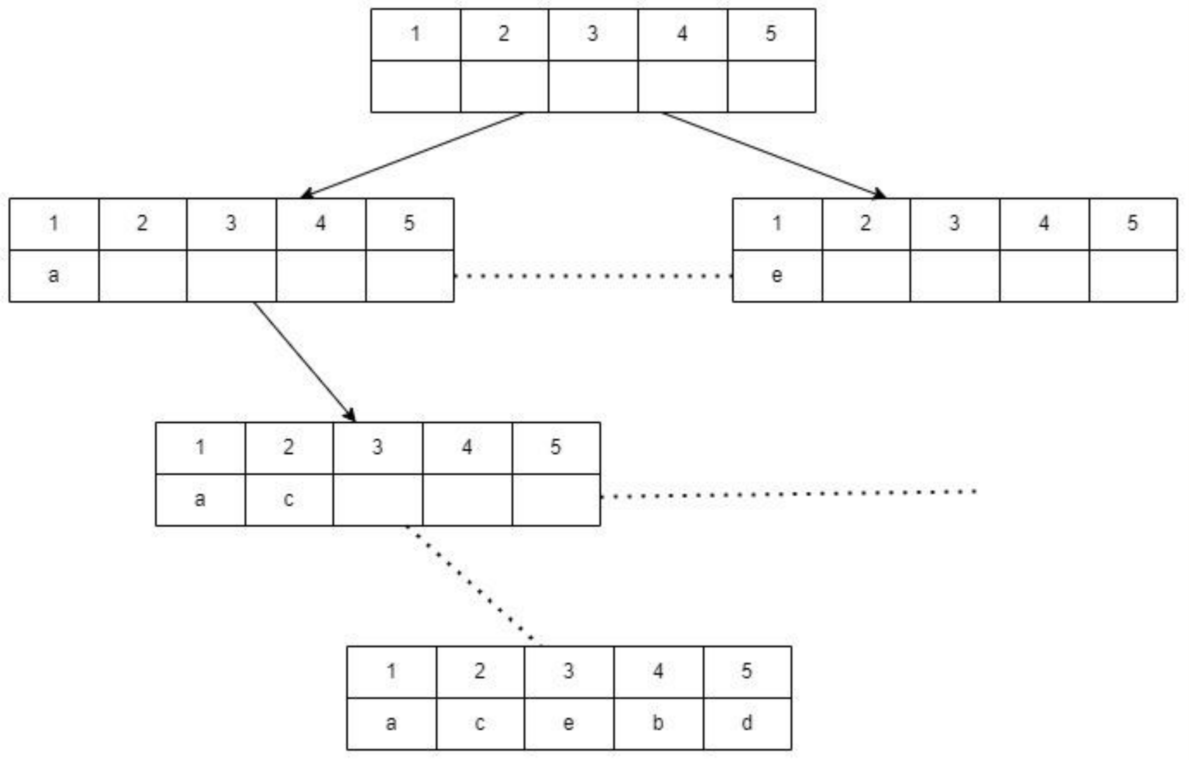
Arbore de căutare:



G_1



G_2

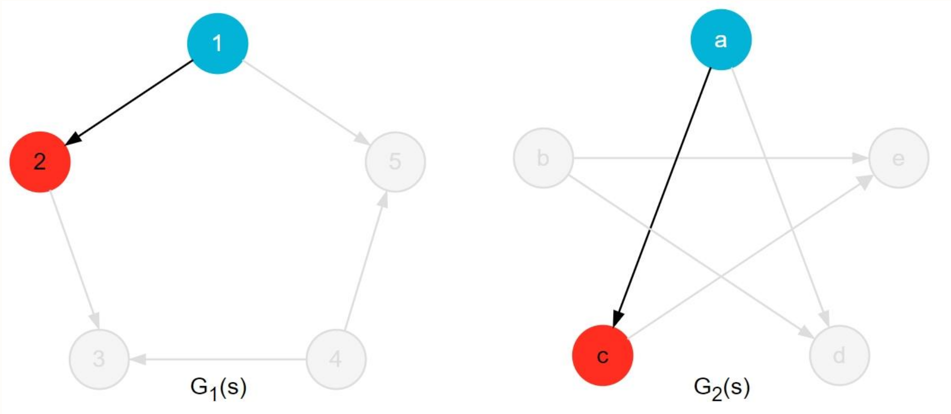


Procedura de rafinare a candidaților:

Pentru a exista izomorfismul trebuie respectată următoarea proprietate:

- Pentru fiecare pereche de noduri compatibile (i, j), cu $i \in G_1$ și $j \in G_2$:
    - fiecare vecin x al lui i trebuie să aibă măcar un candidat $y \in G_2$ vecin cu j

Exemplu:



	a	b	c	d	e
1	1	0	0	0	0
2	0	0	1	0	0
3	0	0	0	1 -> 0	1
4	0	1	0	0	0
5	0	0	0	1	1 -> 0

Schița algoritmului:

Algorithm 1: Algoritmul de căutare Ullman

Input : o stare parțială s (starea inițială s_0 are $M(s_0) = \emptyset$)

Output: maparea izomorfă dintre grafuri dacă există

Procedure `match(`s`):`
- **if** $M(s)$ *este mapare completă* **then**
  - **return** $M(s)$;
- Calculează mulțimea $P(s)$ a perechilor candidat pentru a fi incluse în maparea $M(s)$;
- **foreach** p *in* $P(s)$ **do**
  - Creează o nouă stare s' identică cu s;
  - Adaugă perechea p la maparea $M(s')$;
  - `match(`s'`);`

Calculul perechilor candidat:

- O pereche (i, j), cu $i \in G_1$ și $j \in G_2$, unde:
  - i este următorul nod nemapat din G_1
  - j este nod din G_2 compatibil cu i

Complexitate:

Complexitate timp

- caz favorabil: $O(n * n^2) = O(n^3)$
- caz nefavorabil: $O(n! * n^2)$

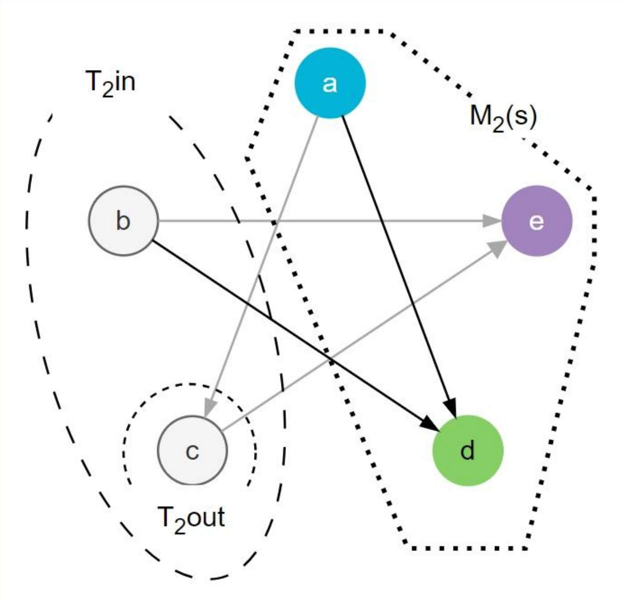
Complexitate spațiu

- $O(n * n^2) = O(n^3)$

Cazul general: VF2

- foarte folosit în practică;
- abordare similară cu algoritmul lui Ullman;
- mulțimi terminale ale stării curente;
- reguli de consistență și fezabilitate;
- partajare de variabile între stări;

Mulțimi terminale:



- maparea curentă $M_2(s) = \{a, e, d\}$
- mulțimi terminale:
  - $T_2in(s) = \{b, c\}$
  - $T_2out(s) = \{c\}$
  - $T_2in(s) \cap T_2out(s) = \{c\}$

Reguli de consistență:

O pereche candidat (i, j), cu $i \in G_1$ și $j \in G_2$, este consistentă dacă:

➔ R_PRED: pentru fiecare predecesor *x* al lui *i* care este mapat la un nod *y*, trebuie ca *y* să fie predecesorul lui *j;*

➔ R_SUCC: similar;

Reguli de fezabilitate:

O pereche candidat (i, j), cu *i* $\in G_1$ și *j* $\in G_2$, este fezabilă dacă:

- ➔ R_IN: numărul de succesori/predecesori ai lui *i* care se află în mulțimea T_1in(s) trebuie să fie egal cu numărul de succesori/predecesori ai lui *j* care se află în mulțimea T_2in(s);
- ➔ R_OUT: similar;
- ➔ R_NEW: similar.

Complexitate:

Complexitate timp

- caz favorabil: $O(n * n) = O(n^2)$
- caz nefavorabil: $O(n! * n)$

Complexitate spațiu

- $O(n + c * n) = O(n)$

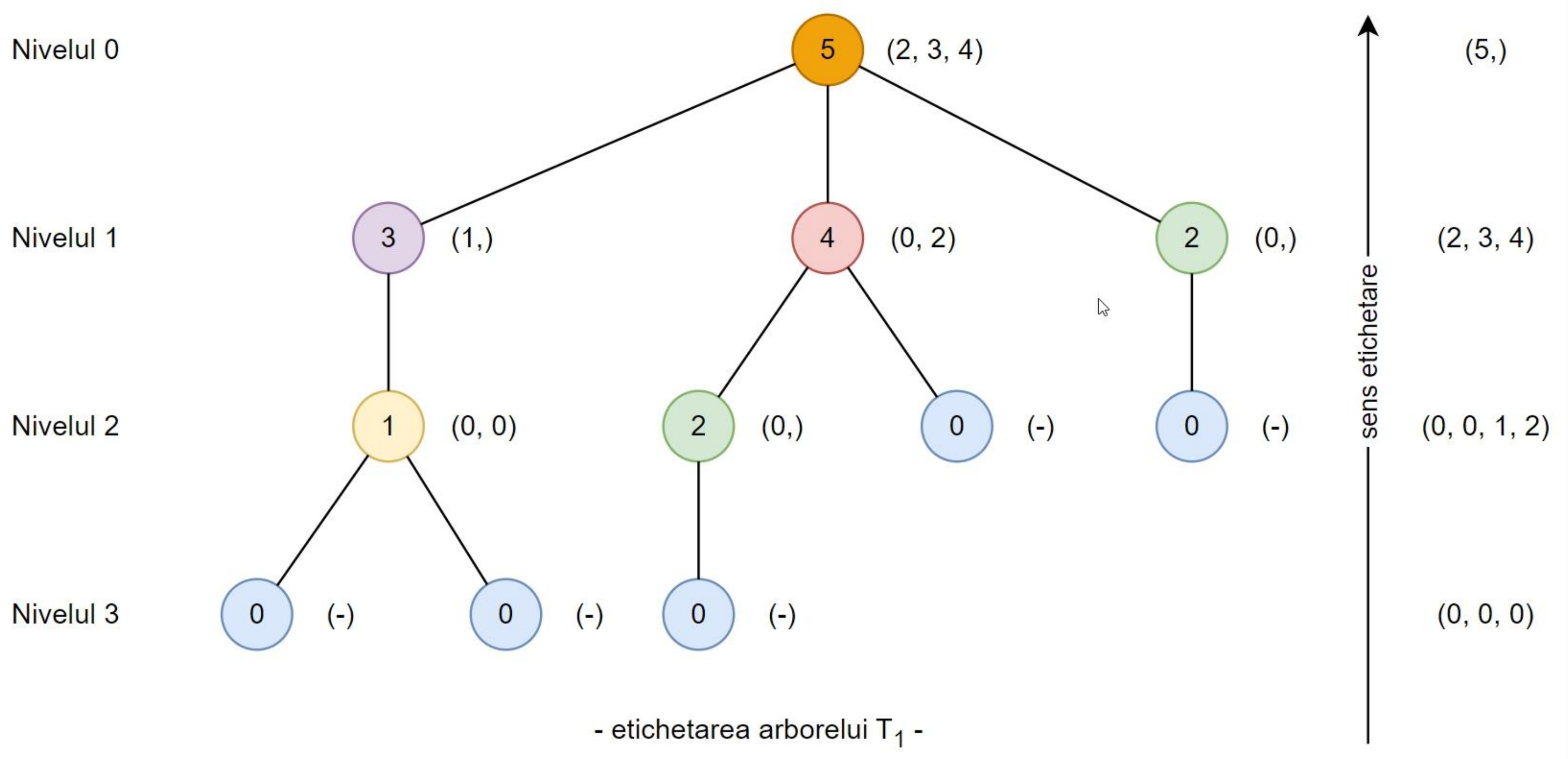
Alți algoritmi:

- Iterații ulterioare ale lui VF2: VF2Plus, VF2++, VF3, VF3-light;
- Nauty: reprezentare canonică;
- Babai:
    - Teoria grupurilor;
    - Complexitate cvasipolinomială.

Izomorfismul arborilor: AHU

- etichetarea nodurilor;
- eticheta nodului ≅ codificarea structurii subarborelui cu rădăcina în nodul respectiv;
- complexitate liniară **O(n)**;
- construirea soluției prin parcurgere BFS.

Etichetarea nodurilor:



Detalii de implementare

Biblioteca de grafuri:

Graph4J

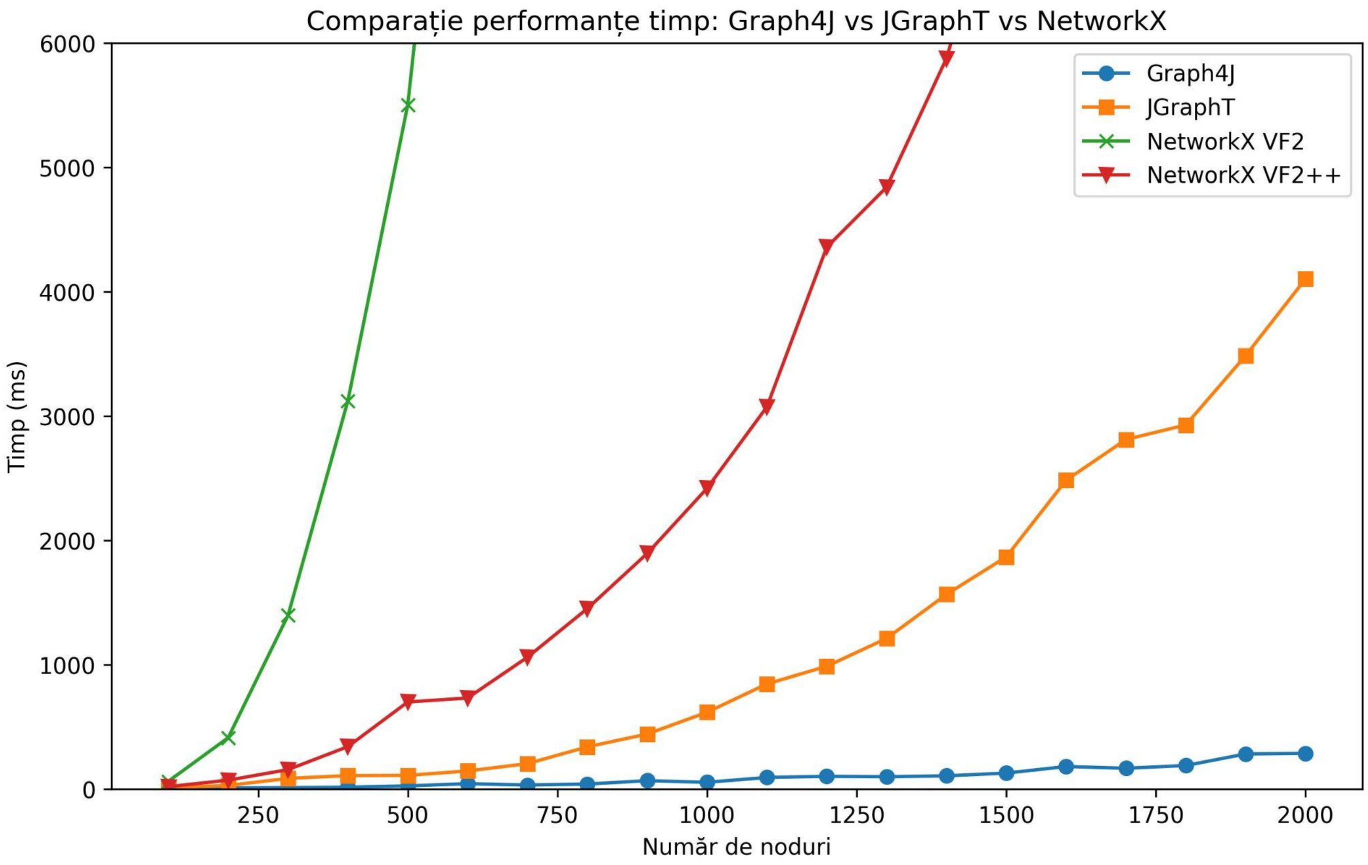
- Model matematic;
- Variabile primitive;
- Operații de bază eficiente.

Îmbunătățiri ale algoritmilor de bază:

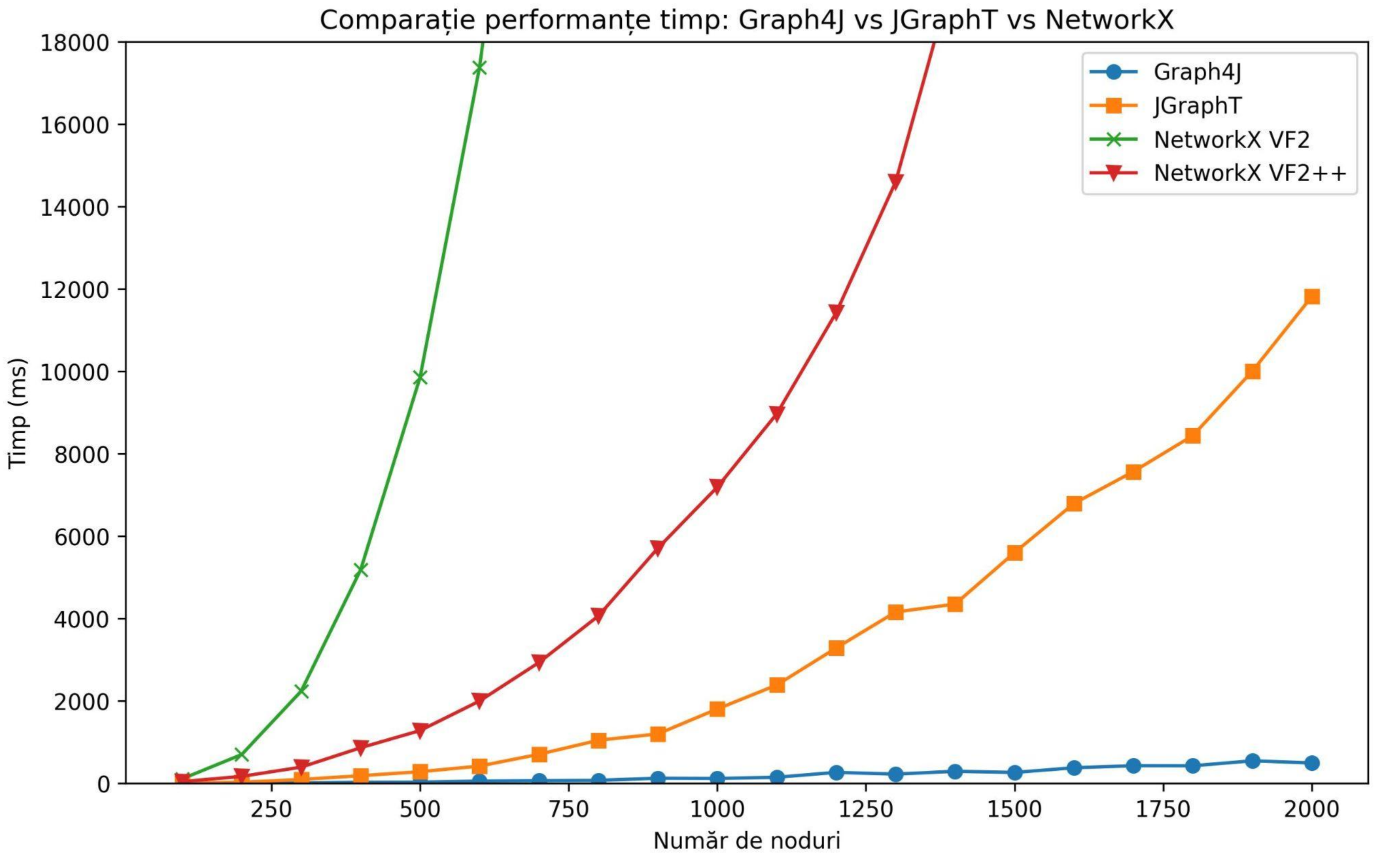
- Ullman: matricea de compatibilitate partajată între stări;
- Ullman și VF2:
  - căutare iterativă;
  - pseudografuri, multigrafurile;
  - problemă derivată: izomorfismul pe subgraf indus;
- AHU: arbori fără rădăcină, păduri de arbori.

PERFORMANȚE

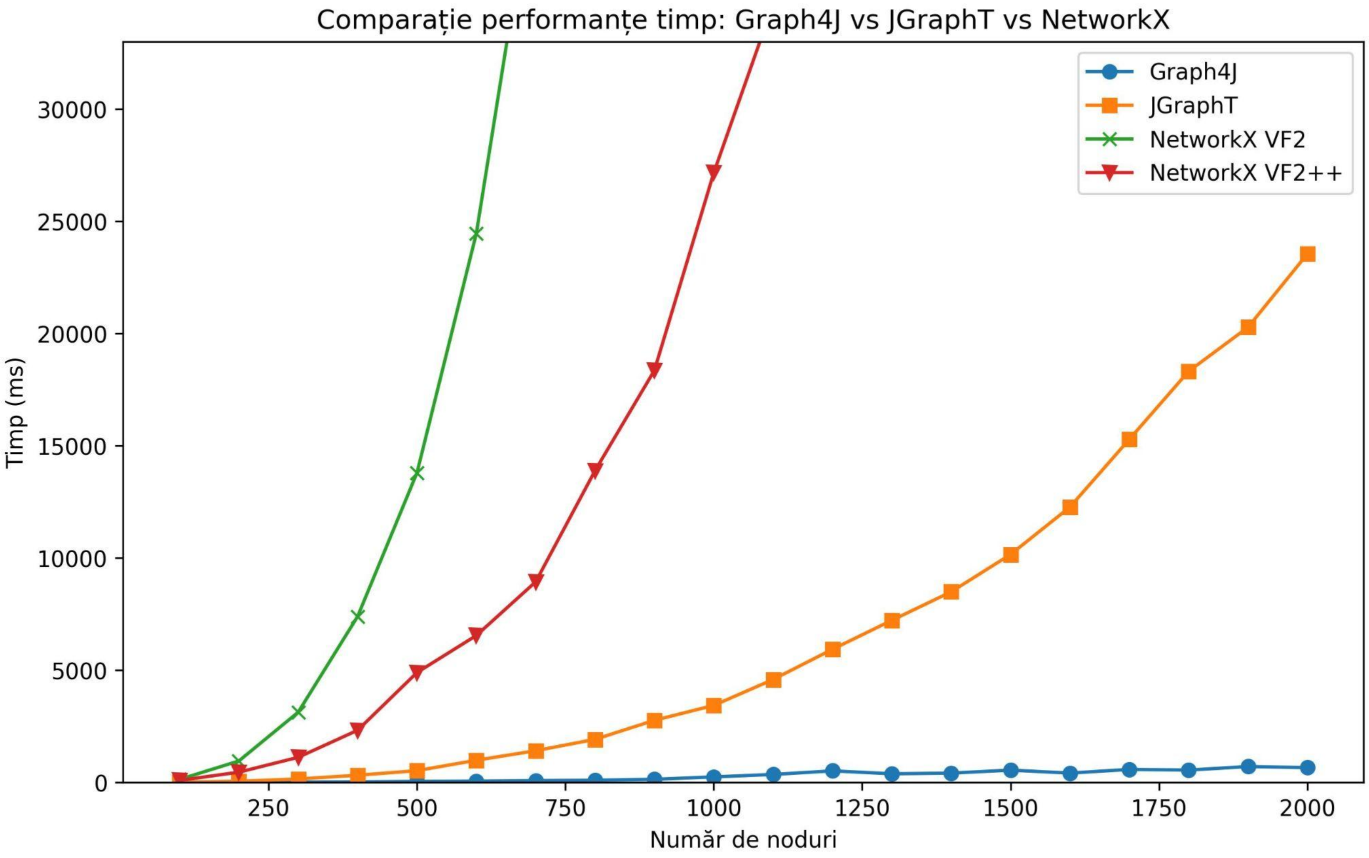
Grafuri rare (p = 0.3):



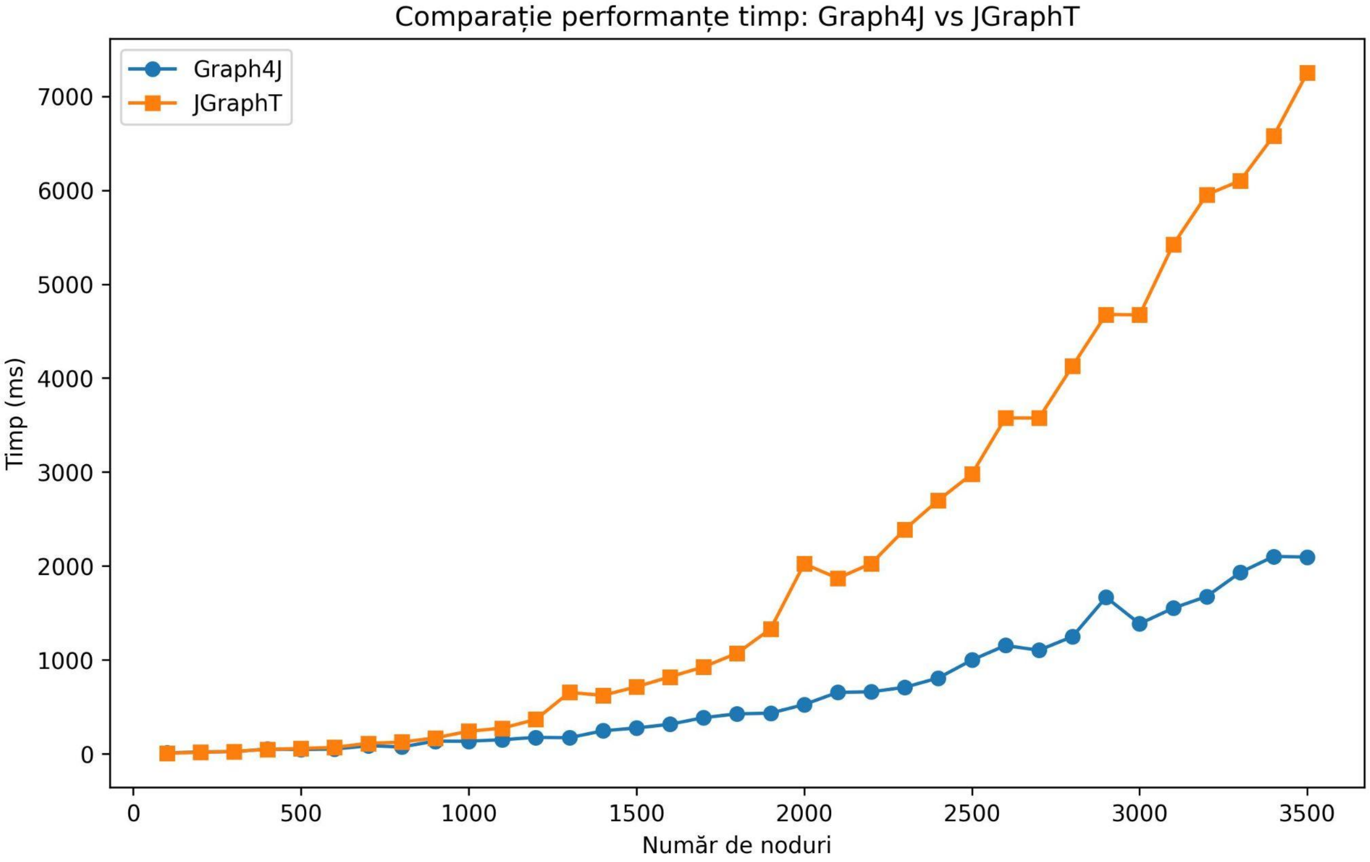
Grafuri de densitate medie (p = 0.6):



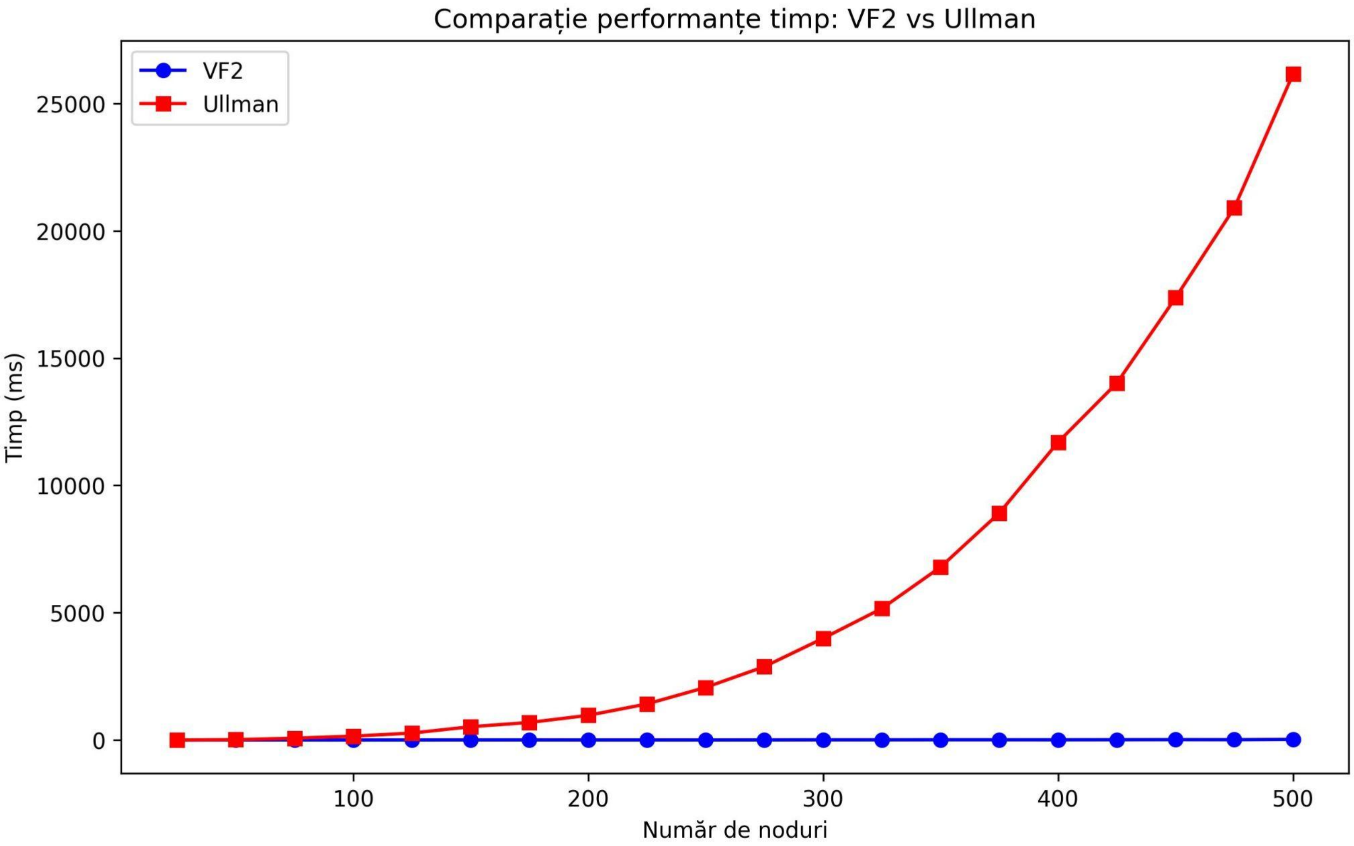
Grafuri dense (p = 0.9):



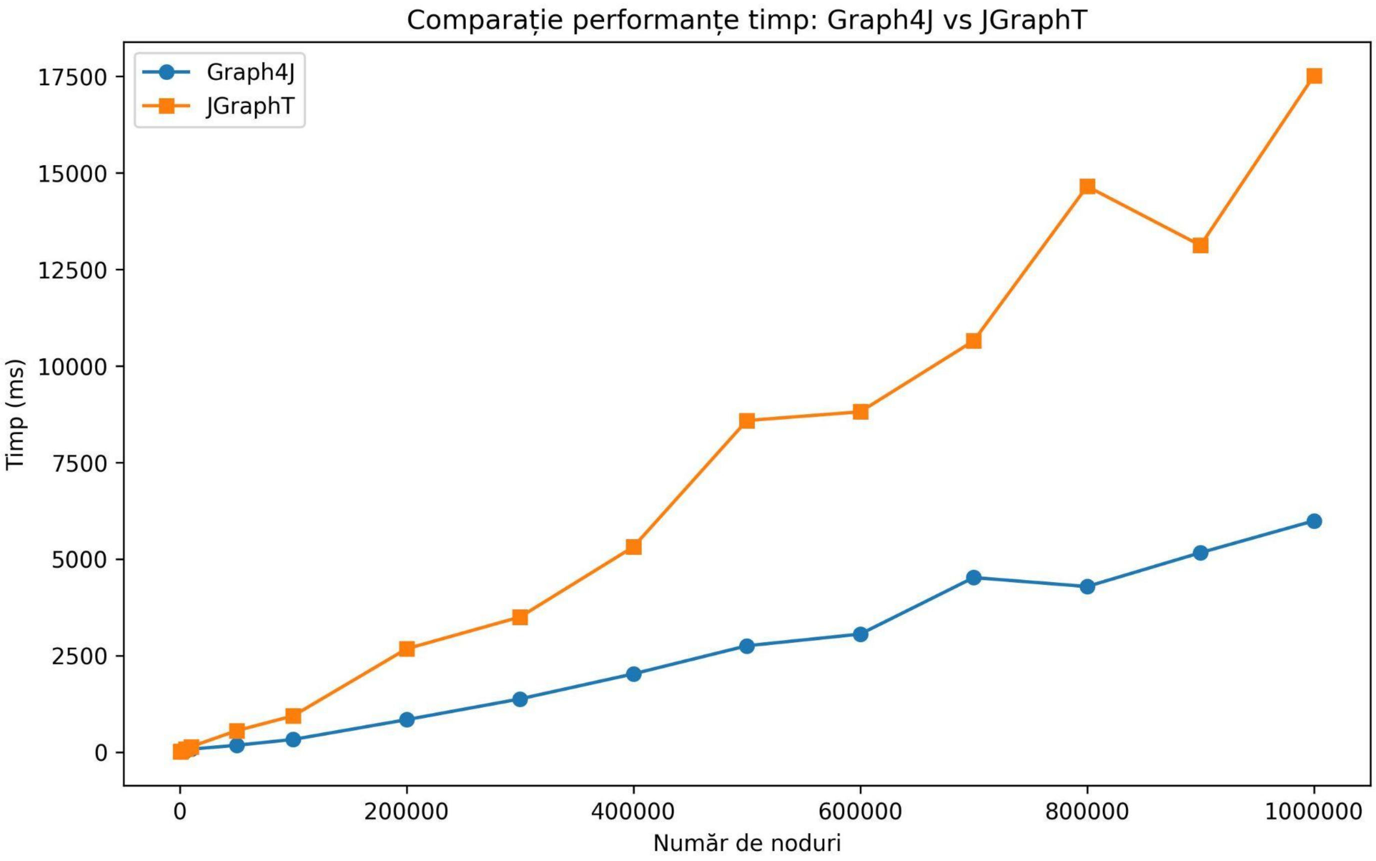
Grafuri Barabási:



Ullman vs VF2 (grafuri rare):



Arbori:


Comparație performanțe timp: Graph4J vs JGraphT
Graph4J
JGraphT
17500
15000
12500
10000
7500
5000
2500
0
Timp (ms)
0
200000
400000
600000
800000
1000000
Număr de noduri

Demo Interfață

Concluzii

Concluzii:

01 Contribuții

- Implementare algoritmi eficienți într-o bibliotecă de grafuri open-source, Graph4J;
- Depistarea, corectarea unor probleme în cadrul bibliotecii;
- Spre deosebire de alte biblioteci, algoritmii implementați au tratat toate cazurile de grafuri.

02 Îmbunătățiri

- Precondiție necesară Weisfeiler-Leman.

03 Direcții de viitor

- Implementarea unor algoritmi mai eficienți: VF3, VF3-light;
- Abordarea problemei izomorfismului inexact.

Mulțumesc pentru atenție!