Temă Analiza Algoritmilor.
Problema colorării nodurilor unui graf

Matișan Răzvan-Andrei, 324CA

Universitatea Politehnică București
Facultatea de Automatică și Calculatoare
razvan.matisan@stud.acs.upb.ro

Abstract: Acest paper oferă mai multe detalii despre problema colorării nodurilor unui graf și compară două modalități de rezolvare atât din punctul de vedere al timpului de execuție, cât și al rezultatului final.

Cuvinte cheie: Colorarea Grafurilor, Teorema celor 4 culori, NP-completă, NP-hard, număr cromatic, k-colorare, Greedy, Backtracking, Sudoku, timp polinomial, timp exponențial.

1 Introducere

1.1 Descriere problemei rezolvate

Problema colorării nodurilor unui graf constituie una dintre cele mai cunoscute din teoria grafurilor. Într-o formă simplificată, ea presupune colorarea nodurilor astfel încât să nu existe două noduri adiacente colorate la fel. De asemenea, o colorare care folosește maxim k culori se numește o *k-colorare*, iar numărul minim de culori neceesare pentru a colora un astfel de graf poartă denumirea de *număr cromatic*. Este important de menționat că această problemă este NP-completă, în timp ce determinarea numărului cromatic este chiar NP-hard.

1.2 Aplicații practice pentru problema colorării nodurilor unui graf

Această problemă a apărut pentru prima dată sub forma unei aplicații practice, ce se referă la găsirea unei modalități de colorare a unei hărți în așa fel încât să nu existe două țări cu granițe comune și de aceeași culoare. Rezolvarea ei a dat naștere primei teoreme demonstrate cu ajutorul calculatorului - *Teorema celor 4 culori*. Așa cum spune și numele, sunt suficiente 4 culori pentru a colora orice harta cu proprietatea de mai sus. În acest caz, se poate considera că fiecare țară este un nod și că fiecare muchie sugerează granița comună dintre două țări.

O altă aplicație o reprezintă *rezolvarea puzzle-urilor Sudoku*. Pentru simplitate, voi explica cum poate fi folosită colorarea nodurilor unui graf pentru jocul clasic de Sudoku (pe un șablon de dimensiune 9 x 9). Într-un mod abstract, Sudoku poate fi privit ca un graf cu 81 de noduri, unde fiecare pătrat din tabla de joc constituie un nod. Mai mult, fiecare nod este adiacent cu toate nodurile de pe aceeași linie, aceeași coloană, respectiv același pătrat de dimensiune 3 x 3 din care face parte). Astfel, putem spune că avem de a face cu o problemă de colorare a unui graf, al cărui număr cromatic este 9.

1.3 Validarea unei colorări

O colorare a unui graf este validă dacă fiecare nod are o culoare diferită de cea a nodurilor adiacente acestuia. Altfel spus, orice muchie aș alege, nodurile care o creează nu trebuie să aibă aceeași culoare. Secvența de cod de mai jos verifică dacă un graf a fost colorat corect sau nu.

```
def isValid(colors: List[int]):
    for edge in edges:
        if colors[edge[first_node]] == colors[edge[last_node]]:
            return False

    return True
```

Fig. 1. Pseudocod pentru validarea colorării unui graf

În plus, este important de menționat faptul că nu contează ce colorare validă particulară am obținut atâta vreme cât ea este obținută folosind cât mai puține culori. O colorare care se realizează folosind un număr de culori egal cu numărul cromatic al grafului se numește o colorare optimă.

1.4 Specificarea soluțiilor alese

Pentru rezolvarea problemei, voi lua în considerare două tipuri de soluții:

- **exactă** – Backtracking;
- **aproximativă** – O soluție de tip Greedy.

Pentru fiecare algoritm în parte, am determinat numărul minim de culori folosite, astfel încât să existe cel puțin o colorare validă, **plecând de la un nod inițial**. Astfel, fiecare algoritm va testa dacă poate să coloreze graful cu ajutorul a m *culori*, parametru ce este inițial egal cu 1. În cazul în care nu a reușit să realizeze nici măcar o colorare validă, va incrementa valoarea lui m și va relua procedeul.

La final, fiecare algoritm va returna atât numărul minim de colorări folosite, cât și o colorare validă. Determinarea numărului minim de culori folosite de fiecare soluție este NP-hard, întrucât așa este și determinarea numărului cromatic al unui graf, însă calcularea acestuia este un criteriu important de diferențiere al celor două soluții.

De accea, în următoarele paragrafe, ne vom concentra, mai degrabă, asupra analizării modului de colorare a unui graf, aceasta reprezentând o problemă NP-completă și principalul subiect al lucrări.

1.5 Criteriile de evaluare pentru soluția aleasă

În ceea ce privește modalitatea de evaluare pentru cele două tipuri de soluții, am creat un set de teste variat care să verifice eficiența timpului de execuție și cât de optimă este colorarea validă obținută.

Astfel, cele 12 teste folosite includ:

- Testul 1 – Graf simplu, cu 4 noduri și 5 muchii;
- Testele 2-3 conțin aceeași topologie a unui graf cu 8 vârfuri și 12 muchii, însă nodurile sunt indexate diferit;
- Testul 4 – Graf 3-regulat;
- Testul 5 – Graf coroană cu 10 noduri (caz particular de graf bipartit));
- Testul 6 – Graf neregulat cu 11 noduri și 17 muchii;
- Testul 7 – Graf simplex cu 8 noduri ce se obține din graful ce are topologie de triunghi;
- Testul 8 – Graf cu 100 de noduri și fără muchii;
- Testul 9 – Graf de tip grid (5 x 5) cu 25 de noduri;
- Testul 10 – Graf complet cu 11 noduri;
- Testul 11 – Graf complet cu 12 noduri;
- Testul 12 – Graf planar cu 31 de vârfuri și 46 de muchii.

2 Descrierea soluțiilor abordate

În continuare, voi detalia cei doi algoritmi folosiți, punând accentul atât pe complexitatea temporală, cât și pe acuratețea cu care fiecare reușește să realizeze cel puțin o colorare optimă.

Așadar, vom considera un graf $G(V, E)$, unde $|V| = n$, cu n număr natural.

2.1 Metoda Brute-Force (Backtracking)

Backtracking este un algoritm exact care încearcă toate colorările posibile pentru un graf. El primește un număr m de culori pe care le poate folosi ca să coloreze graful și va asocia fiecărui nod fiecare culoare pe care o are la dispozitie, verificând mereu dacă acea colorare a grafului este validă.

Complexitate. Astfel, el ajunge să genereze, în cel mai rău caz, m^n moduri de a colora nodurile. Deducem de aici că algoritmul are o complexitate de $O(m^n)$ și se dovedește, astfel, a fi foarte ineficient pentru grafurile cu multe noduri (adică majoritatea celor folosite în practică).

De aceea, este nevoie de o modalitate mult mai eficientă din punctul de vedere al timpului de execuție pentru a obține o colorare validă de maxim m culori.

2.2 Metoda de tip Greedy

Spre deosebire de algoritmul de Backtracking, o soluție de tip Greedy este un tip de euristică secvențială ce ordonează nodurile într-un anumit mod și atribuie fiecărui nod cea mai mică culoare disponibilă (care nu este deja utilizată de vreun nod adiacent). Această abordare poate folosi un număr variabil de culori, iar pentru a fi unul cat mai optim, este nevoie să ordonăm nodurile cât mai convenabil cu putință. Altfel spus, un factor important care influențează cât de optimă este o colorare este reprezentat de nodul de la care începe să se coloreze graful.

Complexitate. Deși se poate întâmpla ca numărul de culori folosite să fie unul foarte mare în comparație cu numărul cromatic, acest algoritm are o complexitate de $O(n^2)$, fiind mult mai rapid ca algoritmul de Backtracking. Ca să calculăm complexitatea, ne putem gândi la cazul cel mai defavorabil, și anume cel în care avem de a face cu un graf complet. Fiecare nod este adiacent cu toate celelalte, astfel că pe măsură ce colorăm nodurile, vom ignora primele k - 1 culori atunci cand ajungem la al k-lea nod. Deci, pentru că avem n noduri (și, implicit, n iterații), vom avea $0 + 1 + 2 + ... + (n - 1) = \frac{n \cdot (n-1)}{2}$ operații, fapt ce demonstrează încadrarea în clasa de complexitate $O(n^2)$.

Pentru mai multe informații referitoare la algoritmul Greedy, puteți accesa secțiunea 2.1 din referința [4].

2.3 Avantajele și dezavantajele metodelor folosite

Avantajele și dezavantajele celor doi algoritmi se determină pe baza a două criterii:

- timpul de execuție;
- cât de optimă este colorarea obținută.

Pe de o parte, algoritmul de Backtracking va garanta de fiecare dată că va colora optim graful, însă timpul de execuție este unul exponențial. Putem aștepta chiar și minute bune pentru a găsi o colorare validă pentru grafuri cu un număr de noduri de ordinul zecilor! De exemplu, un graf cu 10 noduri care folosește 3 culori va putea fi colorat, în cel mai rău caz, după alte $3^{10} - 1$ colorări invalide, enorm dacă ne raportăm la dimensiunea datelor de intrare!

Pe de altă parte, o euristică de tip Greedy va realiza o colorare validă în timp polinomial, ceea e o face să fie mult mai performantă decât metoda Brute-Force. Principalul dezavantaj, însă, este posibilitatea de a nu obține o colorare optimă, deoarece numărul de culori folosit depinde de ordonarea nodurilor. Un exemplu de graf cu număr mic de noduri, pentru care algoritmul nu realizează întotdeauna o colorare optimă, este cel din Figura 2. [1]

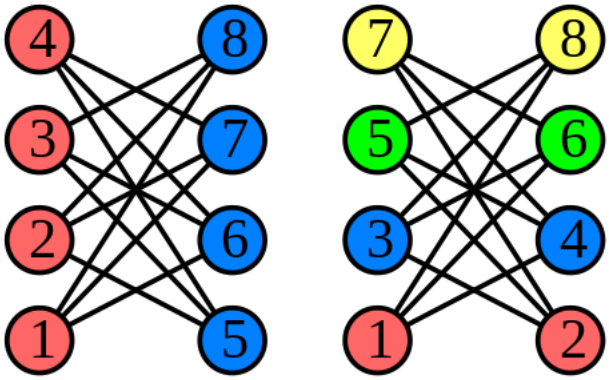


Fig. 2. Colorări Greedy ale aceluiași graf, utilizând indexări diferite ale nodurilor. [1]

3 Modul de evaluare al solutiilor

3.1 Descrierea modalității de construire a setului de teste folosite

Setul de teste a fost creat manual, utilizând un editor de grafuri online [5] pentru o vizualizare mai ușoară a topologiei grafurilor. Testele sunt variate, tocmai pentru a verifica corectitudinea și eficiența celor două soluții pe situații cât mai diversificate.

Astfel, fiecare dintre cele 12 fișiere de input a fost creat de dinainte, având următorul format:

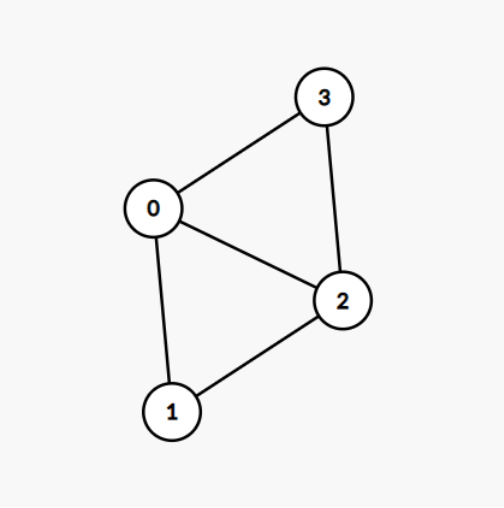
- prima linie conține numărul cromatic al grafului;
- a doua linie conține numărul de noduri ale grafului;
- următoarele linii conțin câte 2 numere, ce reprezintă nodurile care formează împreună o muchie.

Pentru fiecare fișier de input există, în mod evident, câte un fișier de output. Acesta afișează, în ordine, următoarele informații:

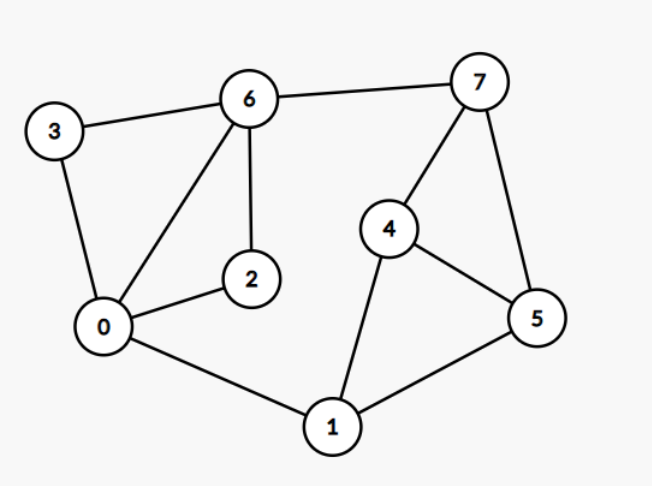
- numărul cromatic al grafului;
- numărul de culori folosite de algoritmul Greedy și un exemplu de colorare validă;
- numărul de culori folosite de algoritmul de Backtracking și un exemplu de colorare validă.

În continuare, voi detalia fiecare tip de graf folosit în setul de teste, atașând câte o poză pentru fiecare topologie prezentată:

– **Testul 1** – Graf simplu, cu 4 noduri și 5 muchii;

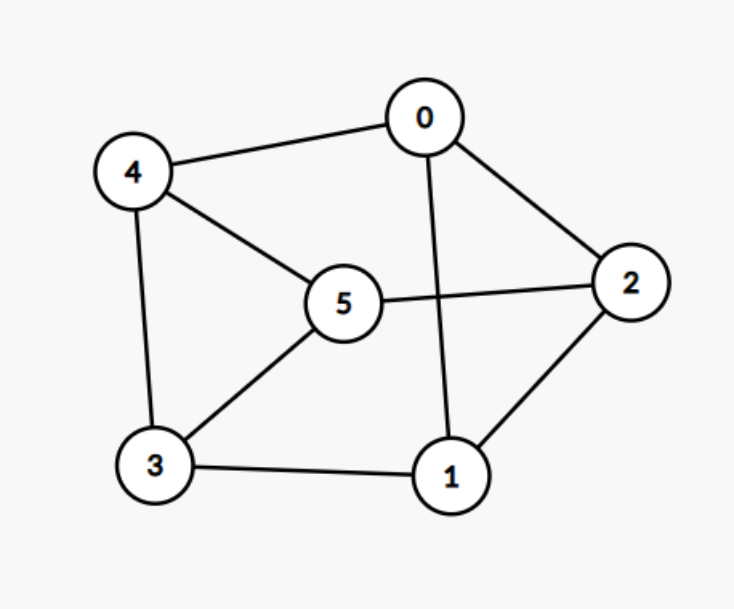


– **Testele 2-3** – conțin aceeași topologie a unui graf cu 8 vârfuri și 12 muchii, însă nodurile sunt indexate diferit;



– **Testul 4** – Graf 3-regulat;

Un graf se numește n-regulat dacă fiecare nod are exact n vârfuri cu care se învecinează. În cazul nostru, $n = 3$.

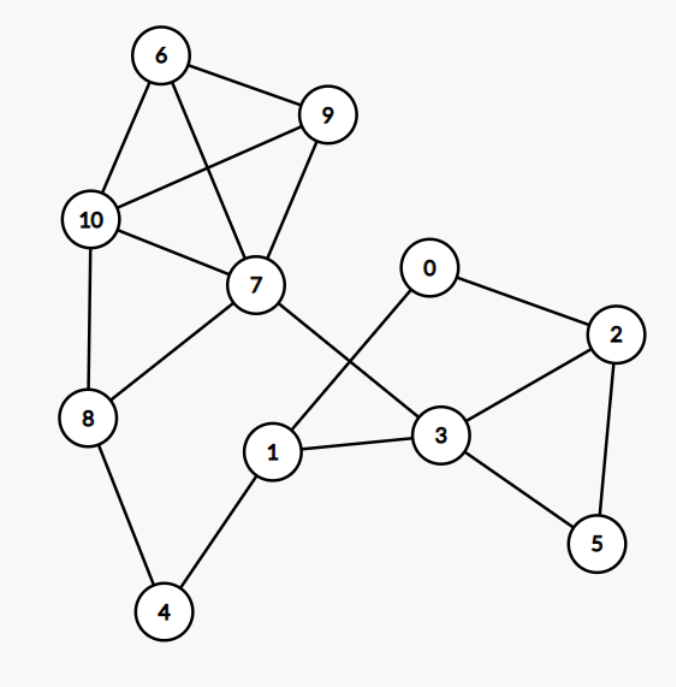


– **Testul 5** – Graf coroană cu 10 noduri;

Un graf coroană cu 2n noduri este un caz particular de graf bipartit, care se obține prin eliminarea a n muchii dintr-un graf bipartit complet, unde cele n muchii se aleg astfel încât fiecare nod să apartină exact uneia dintre ele.

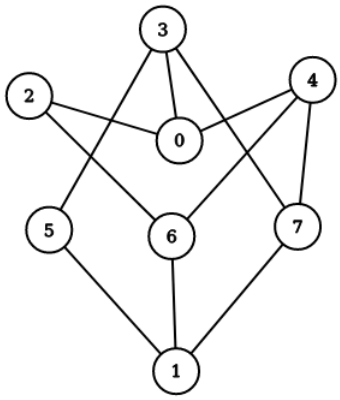
Un exemplu de graf coroană se poate observa în Figura 2 din secțiunea 2.3 a lucrării.

– **Testul 6** – Graf neregulat cu 11 noduri și 17 muchii;
Un graf neregulat, spre deosebire de unul regulat, are proprietatea că oricum am alege două vârfuri adiacente, ele vor avea un număr de vecini distincți.

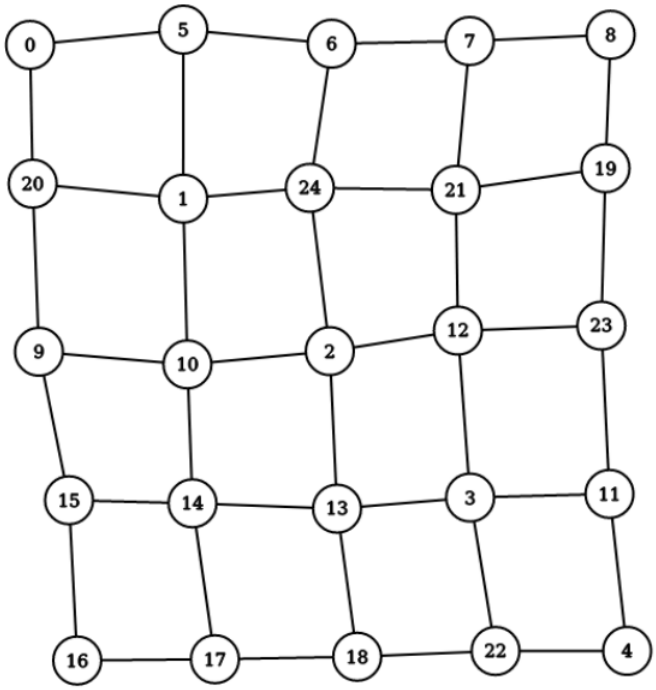


– **Testul 7** – Graf simplex cu 8 noduri ce se obține din graful ce are topologie de triunghi;

Un graf simplex este un caz particular de graf bipartit, cu proprietatea că se poate obține dintr-un graf oarecare, având numărul de noduri egal cu numărul de clici ale grafului din care derivă. De asemenea, două noduri se pot lega între ele printr-o muchie dacă ele provin din clici ce diferă printr-un singur nod.

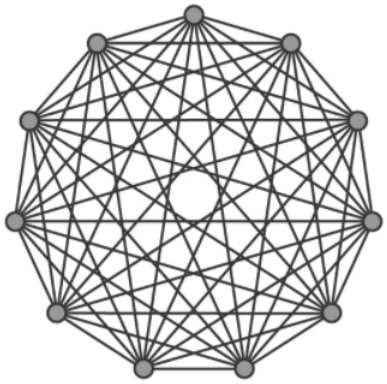


– **Testul 8** – Graf cu 100 de noduri și fără muchii;
– **Testul 9** – Graf de tip grid (5 x 5) cu 25 de noduri;



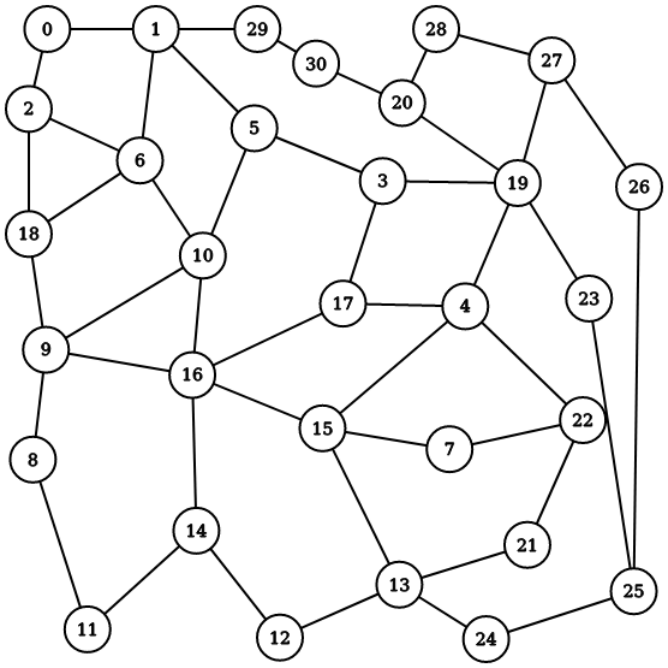
– **Testul 10 - 11** – Grafuri complete (cu 11, respectiv 12 noduri);

Sursa pozei se găsește la [8].


Complete graph with 11 vertices

– **Testul 12** – Graf planar cu 31 de vârfuri și 46 de muchii.

Graful planar este acel tip de graf cu proprietatea că poate fi desenat astfel încât muchiile să se intersecteze între ele numai prin nodurile care le mărginesc.



3.2 Specificațiile sistemului de calcul pe care am rulat testele

Procesor: Intel(R) Core(TM) i7-8550U CPU @1.80GHz
Memorie: 8GB

3.3 Rezultatele evaluării soluțiilor pe setul de teste

Pentru calcularea timpilor de execuție, am folosit biblioteca $< chrome >$. De asemenea, pentru că timpii de execuție pot varia ușor de la o rulare la alta (diferența fiind vizibilă atunci când avem unități mici de măsură precum microsecunda), am rulat programul de 10 ori și am făcut o medie a rezultatelor.

Atât timpii, cât și numărul de culori folosite de fiecare soluție în parte, se pot observa în tabelul de mai jos:

	Numărul cromatic	Numărul de culori folosite		Timpul de execuție (microsec)	
		Backtracking	Greedy	Backtracking	Greedy
Testul 1	3	3	3	48	26
Testul 2	3	3	3	87	28
Testul 3	3	3	4	47	30
Testul 4	3	3	4	33	33
Testul 5	2	2	5	43	29
Testul 6	4	4	4	62	30
Testul 7	2	2	3	27	39
Testul 8	1	1	1	42	78
Testul 9	4	4	4	91	37
Testul 10	11	11	11	2400981 (aprox. 2.4 sec)	29
Testul 11	12	12	12	aprox. 1 minut	58
Testul 12	3	3	4	112	39

3.4 Prezentarea succintă a valorilor obținute pe teste

Rezultatele obținute scot în evidență cât de rapizi și optimi (din punctul de vedere al colorării) sunt cei două soluții.

Așa cum era de așteptat, algoritmul de Backtracking oferă o colorare optimă pe toate cele 12 teste, indiferent de complexitatea grafurilor. În schimb, se poate observa că algoritmul Greedy nu oferă întotdeauna o colorare optimă (nici măcar pe grafurile mici), acest lucru fiind ilustrat chiar de primele teste. De exemplu:

- testele 2 și 3 au aceeași tipologie a grafului, însă indexarea diferită a nodurilor face ca testul 3 să nu realizeze o colorare optimă;
- testele 4 și 12 includ grafuri ce au complexități diferite (primul cu 8 noduri și 12 muchii, iar al doilea – 31 noduri și 46 muchii), însă algoritmul nu returnează o colorare optimă;
- testul 5 este un caz particular de graf pe care algoritmul Greedy nu funcționează deloc optim. Fiind vorba de un graf coroană, ce are numărul cromatic egal cu 2, dacă ordonarea nu se realizează cum trebuie de la început, atunci putem ajunge ca pentru un graf cu 2n noduri să folosim chiar și n culori!

Deși algoritmul de Backtracking a colorat optim toate grafurile, timpul de execuție al acestuia este cu mult mai mare pentru grafuri complexe. De pildă, dacă analizăm rezultatele testelor 10 și 11, ce constituie grafuri complete cu 11, respectiv 12 noduri, vom vedea că performanța este una foarte scăzută în comparație cu celelalte teste. Chiar și diferența de timpi între cele două teste este enormă (aproximativ un minut), deși al doilea are în plus doar un nod și alte 11 muchii.

Privind testele în ansamblu, putem observa că în ceea ce privește performanța algoritmilor, ambele abordări prezintă rezultate similare pe testele mici, diferențele fiind vizibile abia la testele mai complexe pentru metoda Brute-Force. De asemenea, euristica Greedy realizează o colorare într-un timp asemănător pe toate

testele, însă poate rata colorări optime pe orice tip de graf, indiferent de complexitatea sa.

4 Concluzii

După o analiză amănunțită a celor doi algoritmi, putem trage concluzia că problema colorării nodurilor unui graf nu are o soluție generală.

Ea depinde foarte mult de ceea ce vrem să obținem, pentru că fiecare abordare vine la pachet cu avantajele și dezavantajele ei. Astfel, dacă scopul nostru principal este de a găsi o colorare validă, atunci putem opta pentru folosirea algoritmului Greedy, deoarece el poate realiza o astfel de colorare în timp polinomial. Pe de altă parte, dacă beneficiem de un caz banal de graf, atunci o idee bună ar fi să optăm pentru metoda Brute-Force, căci ea ne garantează o colorare optimă într-un timp asemănător cu cel al unui algoritm de aproximare.

În practică, ambii algoritmi sunt folosiți în diverse aplicații. Un exemplu de utilizare a euristicii Greedy este pentru alocarea registrelor, construindu-se, la modul abstract, un graf ale cărui noduri reprezintă valorile ce trebuie asignate registrelor și ale cărui muchii constituie conflictele dintre două valori ce nu pot fi atribuite aceluiași registru [6]. De asemenea, o aplicație pentru Backtracking o constituie rezolvarea jocului de Sudoku despre care am amintit în introducerea lucrării.

În încheiere, voi lăsa o întrebare deschisă pentru toți cititorii. Răspunsul ei ar putea fi, de ce nu, cheia rezolvării problemei: "Merită să renunț la o colorare optimă obținută ineficient pentru una nu la fel de optimă, dar obținută mai rapid?".

5 Referințe

1. Colored Crown Graph
2. Sudoku
3. Graph Coloring Applications – GeeksForGeeks
4. Lewis, R.: A guide to graph colouring, vol. 7. Springer
5. Graph editor
6. Lecture Notes on Register Allocation, 15-411: Compiler Design, by Frank Pfenning, Rob Simmons, and Andre Platzer, 2015
7. Regular Graphs – GeeksForGeeks
8. Quanta Magazine