

# Analiza Algoritmilor

## Problema colorarii unui graf

Iulia-Alina Marin  
Grupa 321CA  
iulia.marin@stud.acs.upb.ro

Universitatea Politehnica Bucuresti  
Facultatea de Automatica si Calculatoare

**Abstract.** **Rezumat** Documentul are ca scop prezentarea si analizarea unor algoritmi ce isi propun rezolvarea problemei colorarii unui graf, intr-un mod cat mai eficient si corect. Vor fi prezentate 2 tipuri de algoritmi si implementarile lor. Se vor compara si analiza din punct de vedere al complexitatii lor.

**Keywords:** Graf · culori · noduri · complexitate

## 1 Introducere

### 1.1 Descrierea problemei rezolvate

Fie un graf neorientat cu noduri si muchii. Problema cere sa asociem o culoare fiecarui nod, astfel incat oricare doua noduri adiacente (conectate printr-o muchie directa) sa aiba culori diferite. Care este numarul minim de culori necesare pentru a colora toate nodurile conform restrictiei mentionate?

### 1.2 Aplicatii reale

Problema colorarii nodurilor unui graf, se muleaza foarte bine pe o problema foarte intalnita, anume "scheduling". De exemplu, in cazul programarii zborurilor avioanelor, putem considera nodurile drept zboruri, culorile drept avioane disponibile si muchiile drept suprapunerile temporale.

Astfel problema devine "Care este minimul de avioane necesare pentru indeplinirea unui program de zboruri dat?".

O alta problema interesanta pe care o putem rezolva cu aceasta logica este jocul Sudoku. Putem asocia liniile, coloanele si patratele cu interdependentele dintre noduri, numerele devenind noduri.

### 1.3 Specificarea solutiilor alese

#### Greedy coloring

Algoritmii greedy (greedy = lacom) sunt in general simpli si sunt folositi la probleme de optimizare.

Folosim urmatorul algoritm greedy: alegem o culoare si un varf arbitrar de pornire, apoi incercam sa coloram varfurile ramase, fara a schimba culoarea. Cand nici un varf nu mai poate fi colorat, schimbam culoarea si varful de start, repetand pasii.

”Rezulta ca, prin metoda greedy, nu obtinem decat o solutie euristica, care nu este in mod necesar solutia optima a problemei. De ce suntem atunci interesati intr-o astfel de rezolvare? Toti algoritmii cunoscuti, care rezolva optim aceasta problema, sunt exponentiali, deci, practic, nu pot fi folositi pentru cazuri mari. Algoritmul greedy euristic propus furnizeaza doar o solutie “acceptabila”, dar este simplu si eficient.”

#### Backtracking

Metoda backtracking se bazeaza pe construirea incrementală de soluții-candidat, renuntand la fiecare candidat parțial in momentul in care se observa că acesta nu poate deveni o soluție optima, valida.

Algoritmul ales utilizeaza metoda backtracking pentru a genera toate modurile de colorare a nodurilor, pana ajunge la o colorare valida cu un numar minim de culori. Se porneste cu un  $k$  de la 1 pana la numarul total de noduri si se cauta valoarea minima a lui  $k$  pentru care se obtine o solutie optima, in care graful poate fi colorat. Cu cat numarul de noduri si de muchii creste, cu atat metoda devine mai ineficienta.

### 1.4 Evaluarea solutiilor

Notand numarul cromatic(numarul de culori necesare colorarii) cu  $x$ , se genereaza 3 tipuri de grafuri :

- 9 grafuri cu putine noduri(aproximativ 20) si cu  $x$  de 5, 10, respectiv aproximativ 20
- 6 grafuri cu multe noduri(aproximativ 150) si cu  $x$  de 5 respectiv 10
- 5 grafuri cu valori aleatoare ale nodurilor cat si ale lui  $x$ (pe care analog le incadram in 5, 10, 15)

Calculam timpul mediu pentru fiecare tip de test in cadrul algoritmilor si observam rapiditatea algoritmului greedy fata de backtracking. Calculam variatia valorilor fata de raspunsul optim si observam erorile algoritmului greedy fata de cel recursiv.

Construim tabele pentru fiecare din cele 3 comparatii si punem in lumina caracteristicile timp/precizie ale algoritmilor,in raport cu x.

Testele vor fi generate atat local(in cazul testelor mici), cat si generate cu un program in rust.

## 2 Prezentarea Solutiilor

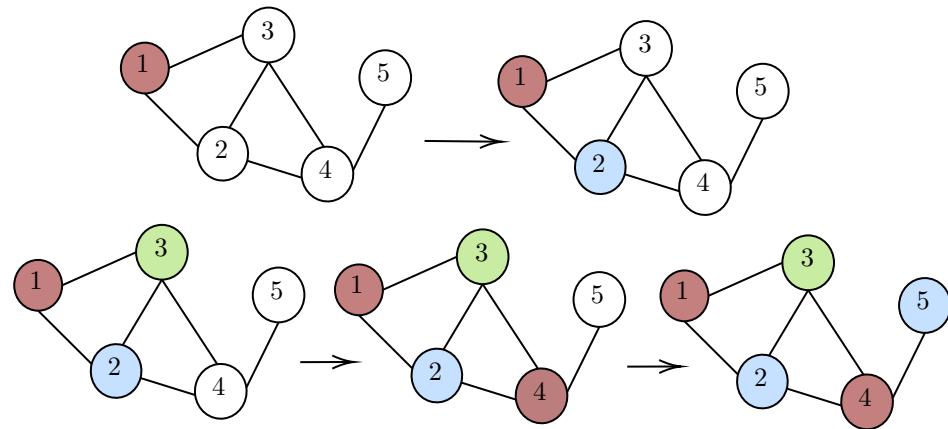
### 2.1 Descrierea modului in care funtioneaza algoritmii

#### Greedy coloring

Pseudocod:

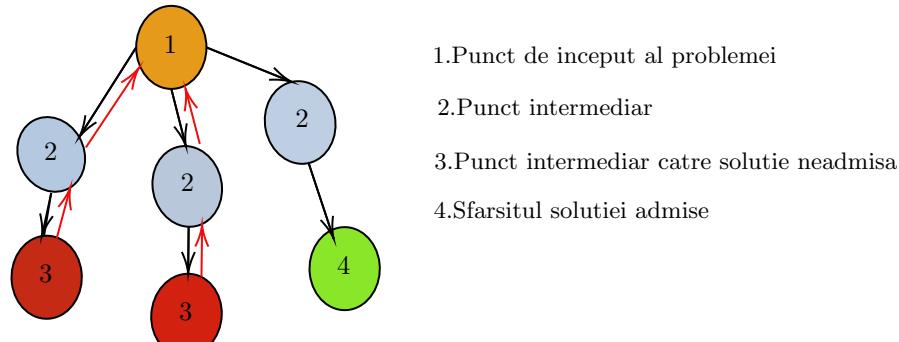
pas1: am colorat primul nod cu prima culoare disponibila.

pas2: pentru fiecare nod ramas, am asignat culoarea cu indexul cel mai mic, fara sa coincida cu nodurile adiacente cu acesta. Daca toate culorile corespund unui nod adicaent, asignam o noua culoare.(ordine culori: roz, albastru, verde)



## Backtracking

Idee generala backtracking:



Pseudocod:

```

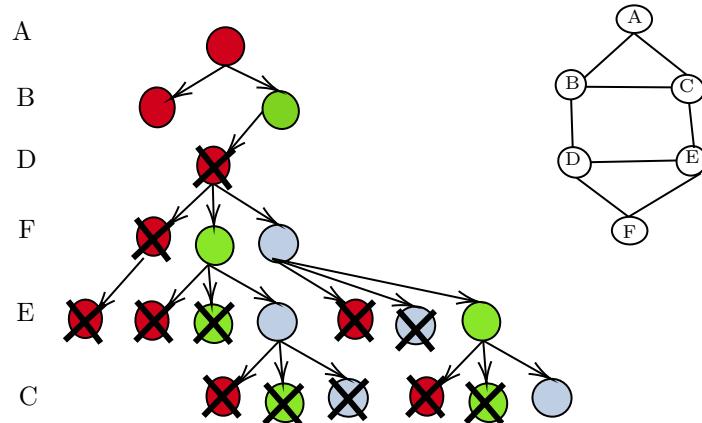
pas1 : Backtrack(solutie parțială)
daca solutia parțială , e o solutie convenabila , afiseaza solutie , daca nu re-
turn/quit .

```

```

pas 2 : pentru toate solutiile parțiale viabile daca extensia solutiei e conven-
abila , atunci intoarce Backtrack(extension)

```



## 2.2 Analiza complexității

**Spatiu** Din punct de vedere al resurselor spatiale, ambeii algoritmi ajung la o complexitate polinomială  $O(n)$ , fapt datorat necesitatii de stocare a culorilor asignate fiecarui nod (asumand  $n$  noduri).

## Timp

**Backtracking** Stiind ca Backtrackingul testeaza pe rand daca graful poate fi colorat pe rand cu 1, 2... k culori, fiecare nod putand fi colorat in i culori. Asumand ca graful are n noduri, pentru teastare cu o culoare se efectueaza  $1^n$  operatii, pentru testarea culorii 2 se efectueaza  $2^n$  ... operatii pana la k (numarul cromatic real) pentru care se efectueaza  $k^n$  operatii. Astfel, complexitatea devine :

$$1^n + 2^n + 3^n + \dots + k^n$$

Pentru cel mai rau caz, graful complet, avem k = n, avem un rezultat care este o functie de  $n^{n+1}$ , care incadreaza logaritmul intr-o complexitate de  $O(n^n)$ . Astfel, algoritmul are mereu o complexitate mai buna sau echivalenta cu  $O(n^n)$ .

**Greedy algorithm** Pentru algoritmul greedy, ne gandim la cea mai costisitoare colorare. Astfel daca la fiecare nod, este necesara o culoare noua, este costisitoare parcurgerea culorilor disponibile. Incepem cu prima culoare disponibila, nu parcurgem nimic, deci complexitate 0. Pentru al 2-lea nod trebuie sa testam prima culoare, care dupa ipoteza nu convine, apoi folosim una noua, rezulta complexitate 1... se repeta pasii pana la nodul n, unde trebuie parcuse n culori care nu convin pentru a folosi o noua culoare. Observam ca la fiecare pas, complexitatea creste cu 1, astfel avem:

$$0 + 1 + 2 + 3 + \dots + n = \frac{(n - 1) \cdot n}{2}$$

Deci, complexitatea temporal'a a algoritmului greedy este  $O(n^2)$ .

### 2.3 Avantaje și dezavantaje

**Greedy:** Punctul slab al algoritmului este colorarea aproximativa, nu mereu optima a grafului. Punctele tari constau in colorarea totusi corecta, cu un numar aproximativ de culori si rapida (timp polinomial) a acestuia.

**Backtracking:** Dezavantajul algoritmului este timpul excesiv necesar colorarii, acesta devenind exponential pentru grafuri cu peste 20 noduri. Avantajul de necontestat al acestuia este gasirea numarului cromatic minim, pe orice tip de graf.

## 3 Evaluare

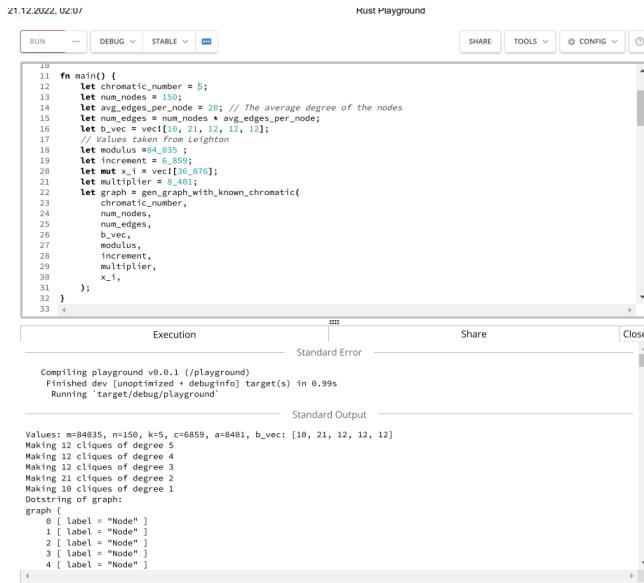
### 3.1 Construirea setului de teste

Am generat teste cu ajutorul unui program scris in Rust pentru generarea grafurilor in functie de numarul cromatic. Algoritmul este un "multiplicative

congruential generator (MCG)" , cea mai cunoscuta si veche metoda de a genera numere pseudo-aleatoare iterativa, ce functioneaza dupa relatia :

$$x[ i ] = ( x[ i - 1 ] * a + c ) \% m$$

Am ales convenabil , k , modulul, incrementorul, multiplicatorul si seedul astfel incat sa respecte criteriile impuse de Leighton in algoritm pentru fiecare tip de test.



```

11 fn main() {
12     let chromatic_number = 5;
13     let num_nodes = 15;
14     let avg_degree_per_node = 2.5; // The average degree of the nodes
15     let num_edges = num_nodes * avg_degree_per_node;
16     let b_vec = vec![10, 21, 12, 12, 12];
17     // Values taken from Leighton
18     let modulus = 68485;
19     let increment = 6859;
20     let mut x_i = vec![36876];
21     let multiplier = 8461;
22     let graph = gen_graph_with_known_chromatic(
23         chromatic_number,
24         num_nodes,
25         num_edges,
26         modulus,
27         increment,
28         multiplier,
29         x_i,
30     );
31 }
32 }
33 
```

Execution Standard Error Share Close

Compiling playground v0.0.1 (/playground)  
 Finished dev [unoptimized + debuginfo] target(s) in 0.99s  
 Running "target/debug/playground"

Values: m=68485, n=15, k=5, c=6859, a=8461, b\_vec: [10, 21, 12, 12, 12]  
 Making 12 cliques of degree 5  
 Making 12 cliques of degree 4  
 Making 12 cliques of degree 3  
 Making 21 cliques of degree 2  
 Making 10 cliques of degree 1  
 Done creating graph:  
 graph {  
 0 [ label = "Node" ]  
 1 [ label = "Node" ]  
 2 [ label = "Node" ]  
 3 [ label = "Node" ]  
 4 [ label = "Node" ]  
 }

Am generat apoi vizual cu ajutorul platformei GraphvizOnline de pe GitHub, pentru a se vedea raportul noduri / muchii si a intelege complexitatea celor 2 parcurgeri.

Astfel, am construit cum am propus 20 teste astfel :

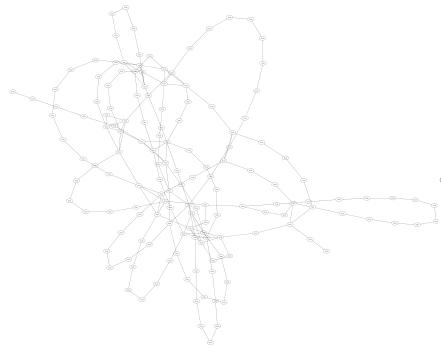
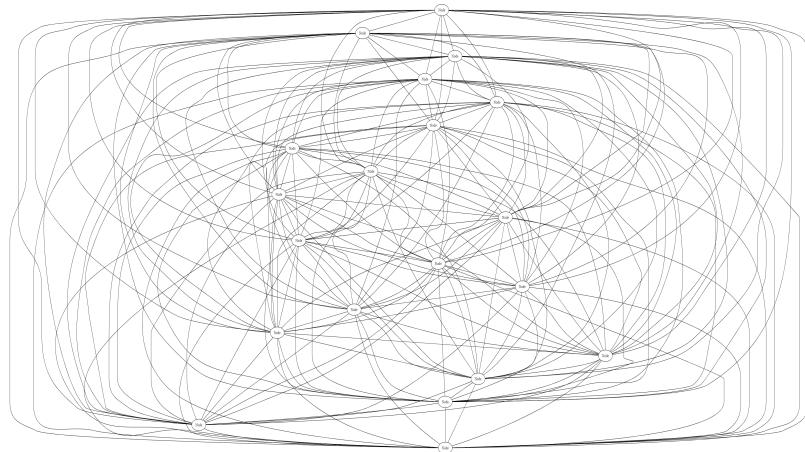


Figure 1



Figure 2

6 teste cu 150 noduri(figura 1) , 9 teste cu 20 noduri(figura 3)  
 5 teste aleatoare(grafuri bipartite sau cu multe noduri ex. figura 2 )pentru a  
 putea compara eficient rezultatele.



### 3.2 Specificațiile sistemului de calcul

**Hardware:** Programele au fost rulate pe un sistem de calcul cu procesor Intel(R) Core(TM) i7-6500U CPU 2.50GHz 2.60 GHz , RAM 8.00 GB (7.57 GB usable) cu tip sistem 64-bit operating system, x64-based processor.

**Software:** programele au fost scrise in Java, rulate in IntelliJ:  
 Runtime version: 17.0.4.1+7-b469.62 ,amd64 VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.

**Benchmark:** Cores: 2 , Threads: 4, Typical TDP: 15 W, Turbo Speed: 3.1 GHz .

### 3.3 Ilustrarea rezultatelor evaluarii

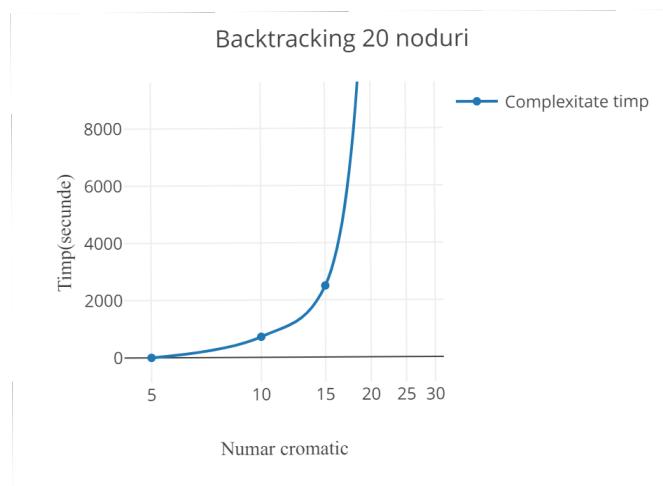
.... Observam ca Backtrackingul incepe sa creasca exponential ca timp la testul 4(care dureaza pana la 3.5 minute pentru un graf cu 20 de noduri si numar cromatic 10). Rezultate rulari in secunde:

Nr. cromatic *rulare 1* *rulare 2* *rulare 3*

<b>4</b>	0.016	0.009	0.011
<b>5</b>	0.13	0.12	0.016
<b>10</b>	70.42 318,84 1768.74	71.42 322,14 1749,65	69.2 319,34 1771,37
<b>20</b>	+20min	+20min	+20min

} 3 grafuri diferite ca desime

Reprezentarea mediei valorilor timpului pe fiecare numar cromatic dupa mai multe rulari:



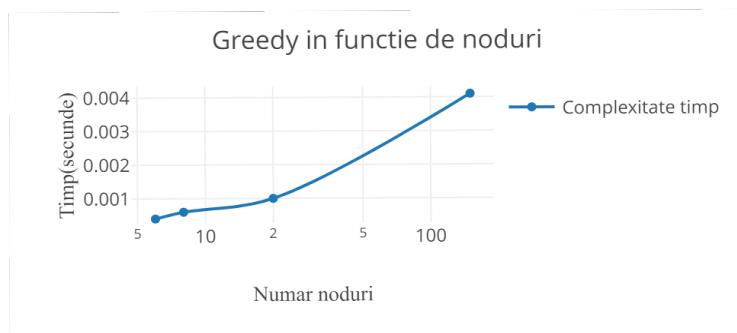
Pentru aceleasi tested, cu 20 noduri, am rulat de 3 ori si algoritmul greedy. Rezultatele in secunde sunt:

Nr. cromatic *rulare 1* *rulare 2* *rulare 3*

Nr. cromatic	<i>rulare 1</i>	<i>rulare 2</i>	<i>rulare 3</i>
4	0.001	0.0012	0.0011
5	0.014	0.12	0.018
10	0.01 0.038 0.068	0.012 0.021 0.045	0.0103 0.032 0.037
20	0.046	0.059	0.056

} 3 grafuri diferite ca desime

Observam diferențe infime odată cu creșterea culorii cromatice pentru algoritmul greedy. Observăm creșterea timpului pentru creșterea numărului de noduri.



Pentru același graf bipartit dat în testele 16 și 17 în ordine diferită a muchiilor, observăm cum algoritmul greedy da pentru unul răspunsul corect, iar pentru celalalt răspuns eronat (dublat față de cel optim), backtrackingul oferind răspunsul corect.

```

Test16-----
2 Chromatic number: 4
3 Time taken: 7151700
4 Space taken: 2MB
5 -----
6

Run: Backtracking < Greedy <
--GREEDY ALGORITHM--
Graph:
0: 3 5 7
1: 2 4 6
2: 1 5 7
3: 0 4 6
4: 1 3 7
5: 0 2 6
6: 1 3 5
7: 0 2 4
numar cromatic 4
Vertex 0 ---> Color 1
Vertex 1 ---> Color 1
Vertex 2 ---> Color 2
Vertex 3 ---> Color 2
Vertex 4 ---> Color 3
Vertex 5 ---> Color 3
Vertex 6 ---> Color 4
Vertex 7 ---> Color 4
-----
Process finished with exit code 0

Test16-----
2 Chromatic number: 2
3 Time taken: 8546600 nanoseconds
4 Space taken: 2MB
5 -----
6

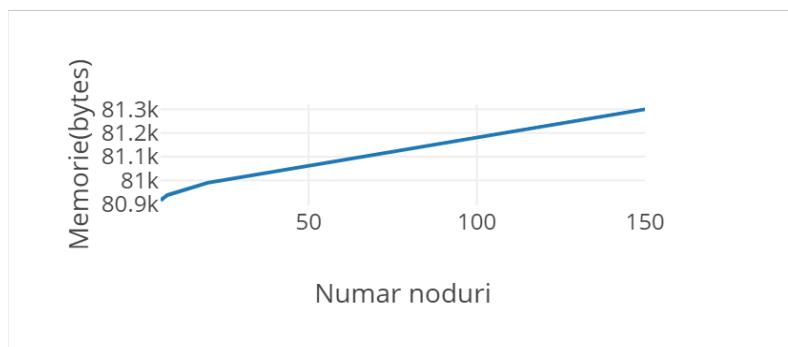
Run: Backtracking < Greedy <
--BACKTRACKING ALGORITHM--
Graph:
0: 3 5 7
1: 2 4 6
2: 1 5 7
3: 0 4 6
4: 1 3 7
5: 0 2 6
6: 1 3 5
7: 0 2 4
Chromatic number2
Vertex 0 ---> Color 1
Vertex 1 ---> Color 2
Vertex 2 ---> Color 1
Vertex 3 ---> Color 2
Vertex 4 ---> Color 1
Vertex 5 ---> Color 2
Vertex 6 ---> Color 1
Vertex 7 ---> Color 2
-----
Process finished with exit code 0

```

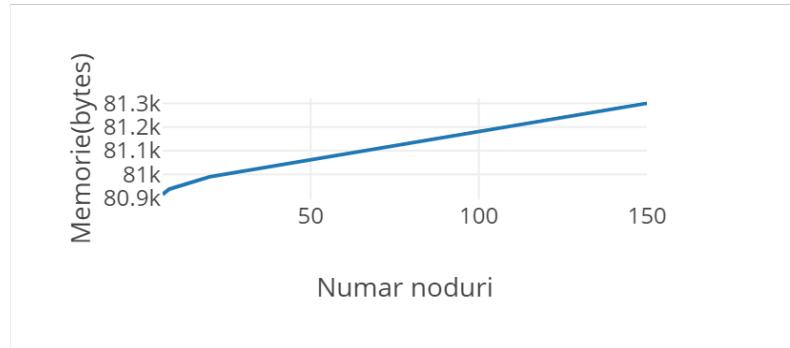
Testele 18 si 19 arata inca inca o data calculul eronat al algoritmului greedy in comparatie cu backtracking. Algoritmul se comporta imprevizibil pe grafuri bipartite in functie de ordinea in care primeste nodurile.

Din punct de vedere al memoriei, ambele au aceeasi complexitate  $O(n)$  data de stocarea aceluiasi  $n$  numar de noduri.

*Memorie Backtracking in functie de noduri*



*Memorie Greedy in functie de noduri*



## 4 Concluzii

### 4.1 Abordarea problemei in practica

In concluzie, variantele prezentate nu sunt perfecte, ci complementare din punct de vedere al folosirii in viata reala.

Varianta backtracking ne aduce rezultatul corect intr-un mod foarte costisitor din punct de vedere al timpului, devenind util doar pe grafuri mici, banale, ceea ce nu ne este folositor pe probleme reale.

Varianta Greedy, desi ne da un rezultat aproximativ performanta ei in timp polinomial o face varianta mai competitenta in fata situatiilor reale. Algoritmul, desi nu da mereu colorarea minima, da intotdeauna o colorare corecta a grafului, ceea ce in majoritatea problemelor reale, este mai mult decat suficient.

## References

1. <https://muhaz.org/algoritmi-tehnici-si-limbaje-de-programare.html?page=12>
2. Introducere în analiza algoritmilor. Giumale, Cristian A., Editura Polirom, Bucuresti, 2004.
3. <https://play.rust-lang.org/version=stablemode=debugedition=2021gist=9ef6c1c7a947fddfce496f6303b687c7>
4. <https://www.qreferat.com/referate/informatica/Backtracking-in-grafuri114.php>
5. [https://nvlpubs.nist.gov/nistpubs/jres/84/jresv84n6p489\\_a1b.pdf](https://nvlpubs.nist.gov/nistpubs/jres/84/jresv84n6p489_a1b.pdf)
6. [https://nvlpubs.nist.gov/nistpubs/jres/84/jresv84n6p489\\_a1b.pdf](https://nvlpubs.nist.gov/nistpubs/jres/84/jresv84n6p489_a1b.pdf)