

Colorarea grafurilor

Căramidă Iustina-Andreea - 322CA

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București
`iustina.caramida@stud.acs.upb.ro`

Abstract. Problema colorării nodurilor unui graf (*graph coloring* - *GCP* sau *colorarea grafurilor*) este una dintre cele mai importante probleme, fapt datorat numărului mare de situații din viața reală care se pot rezuma la o problemă de colorare a unui graf. În această temă, voi prezenta câteva comparații între diferiți algoritmi dedicați acestei probleme, precum Greedy, Dsatur și RLF din punct de vedere al timpului de execuție, al memoriei folosite și al performanței în raport cu numărul de noduri și muchii.

Keywords: Brute-Force · Greedy · DSatur · RLF · Colorarea grafurilor.

1 Introducere

1.1 Descrierea problemei

Colorarea grafurilor este o problemă centrală în teoria grafurilor [1]. Ea constă în alegerea unui set de culori pentru nodurile unui graf, astfel încât niciun nod adiacent să nu primească aceeași culoare.

Problema colorării graficului a început cu încercarea lui Francis Guthries de a colora toate țările pe harta Angliei. Acesta a presupus inițial că patru culori sunt suficiente pentru a procesa orice hartă, astfel încât să nu fie asociate două țări vecine cu aceeași culoare. Această sarcină este doar una dintre cele peste 200 de probleme [2], legate de aria analizei grafice cromatice, iar această situație poate fi tradusă prin colorarea fiecărui vârf dintr-un graf în care marginile sale ar reprezenta vecinătatea dintre două regiuni [3].

Colorarea grafurilor este o problemă de tip NP-hard bine studiată cu aplicații importante în optimizarea combinatorie și într-un domeniu de cercetare activ, cu multe aplicații practice [4] în inginerie, cum ar fi *alocarea registrelor*, *atribuirea frecvenței*, *potrivirea șablonului* și *programări*. În consecință, colorarea grafurilor a fost subiectul unor cercetări intense [5] [2].

Un exemplu de o problemă de colorare a grafurilor este planificarea examenelor, unde fiecare examen este reprezentat de un nod și fiecare legătură dintre noduri reprezintă o interdicție de a planifica examenele. Astfel, colorarea grafului reprezintă planificarea examenelor, astfel încât două examene care nu pot fi planificate în același timp să fie colorate diferit. În acest caz, numărul minim de culori necesare pentru a colora graful este egal cu numărul minim de zile necesare pentru a planifica examenele. [6]

Colorarea grafurilor este asociată cu două tipuri de colorare: colorarea vârfurilor și colorarea marginilor. Scopul ambelor tipuri de colorare este de a colora întregul graf fără conflicte. Prin urmare, vârfurile adiacente sau muchiile adiacente trebuie să fie în culori diferite. Numărul celor mai mici posibile culori care pot fi utilizate pentru colorarea grafului se numește **număr cromatic**. Pe măsură ce numărul de vârfuri sau muchii dintr-un graf crește, complexitatea problemei crește și ea. Din această cauză, fiecare algoritmul nu poate găsi numărul cromatic exact și pot fi, de asemenea, diferențe în timpul lor de execuție [7]. Pentru a obține o soluție mai bună pentru colorarea grafurilor, mulți algoritmi euristici și meta euristici au fost inventați [8].

1.2 Specificarea algoritmilor aleși

Cerință: Fie un graf neorientat G cu N noduri și M muchii. Problema cere să asociem o culoare fiecărui nod, astfel încât oricare două noduri adiacente (conectate printr-o muchie directă) să aibă culori diferite. Care este numărul minim de culori necesare pentru a colora toate nodurile conform restricției menționate anterior? [9]

Brute Force: Când încercăm să oferim o soluție la această problemă, primul instinct este de a folosi o abordare "Brute Force". Acest lucru ar duce la o soluție care ar fi din punct de vedere al timpului de execuție exponențială, făcând această soluție inutilă pentru cazuri mari. A spune că nu putem găsi un algoritm eficient deoarece acesta nu există ar fi la fel ca și când am spune că problema nu are nicio soluție eficientă [10]. În prezent, există algoritmi care se ocupă să rezolve problema colorării grafurilor, deși obțin un număr cromatic apropiat de al grafului în schimbul unui timp rezonabil sau rezultate rapide care sunt suficient de utile [11].

Greedy Algorithm: Logica algoritmului ia vârfurile grafului unul câte unul, urmând o ordine (care poate fi aleatorie) și atribuie prima culoare disponibilă fiecărui vârf [12]. Deoarece este un algoritm euristic, soluția oferită de acesta poate să nu fie optimă. Cu toate acestea, o alegere corectă a ordinii vârfurilor pentru colorarea lor poate oferi o soluție optimă pentru orice graf. În practică, algoritmul Greedy produce soluții rapid practicabile, deși aceste soluții pot fi "sărace" pe baza numărului de culori pe care algoritmul le cere, în comparație cu numărul cromatic al grafului.

DSatur Algorithm: Algoritmul DSatur (abreviere din engleză pentru "Degree Saturation"), propus de Brelaz (1979), se comportă foarte asemănător cu algoritmul Greedy, cu excepția că, în acest caz, ordonarea vârfurilor este generată de algoritmul însuși. La fel ca în algoritmul Greedy, ordonarea a fost decisă înainte ca orice vârf să fie colorat, ordinea vârfurilor fiind decisă euristic pe baza caracteristicilor colorării parțiale a grafului la momentul în care se selectează fiecare

dintre vârfuri [13]. În cel mai rău caz, complexitatea sa are aceeași situație ca și în Algoritmul Greedy, deși în practică poate fi luat în considerare și faptul că monitorizarea saturației vârfurilor necolorate produce o complexitate puțin mai mare. Este de reținut că Algoritmul DSatur este **exact** pentru grafurile bipartite [14].

RLF Algorithm: Algoritmul RLF (abreviere din engleză pentru “Recursive Largest First”), propus de Leighton (1979), lucrează prin colorarea unui graf cu o singură culoare pentru fiecare iterație a algoritmului, în loc de un vârf per repetare. În fiecare iterație, algoritmul caută seturi de vârfuri independente din graf, care vor fi asociate cu aceeași culoare. Acel set independent a fost eliminat din graf, iar subgraful rămas va continua în același mod, până când subgraful menționat este gol, caz în care toate vârfurile vor fi atribuite unei culori, producând astfel o soluție ce satisface toate cerințele [12].

1.3 Evaluarea soluțiilor

Sursele vor fi testate pe grafuri de dimensiuni diferite, de la 10 la 2 000 de noduri, cu un număr de muchii de la 10 la 1 000 000. Pentru fiecare graf se va genera o configurație aleatoare de noduri și muchii, iar apoi se va testa performanța algoritmilor pe aceste grafuri. Pentru fiecare algoritm, se va calcula timpul de execuție și numărul de culori folosite pentru a colora graful. De asemenea, se va calcula și numărul de noduri colorate în fiecare iterație a algoritmului, pentru a vedea cât de eficient este algoritmul în ceea ce privește numărul de iterații necesare colorării grafului.

2 Prezentarea soluțiilor

2.1 Algoritmul Brute Force

Algoritmul Brute Force este un algoritm care se bazează pe forță brută, adică parcurge toate posibilitățile de colorare a grafului și alege cea mai bună soluție. Complexitatea acestui algoritm este de $O(n^m)$, unde n este numărul de noduri și m este numărul de culori. Pentru grafuri cu un număr mare de noduri, complexitatea acestui algoritm este foarte mare, de aceea nu este folosit în practică. Pe de altă parte, algoritmul este exact, adică oferă întotdeauna o soluție optimă. Un pseudo-cod al algoritmului este prezentat în Pseudo codul 1.

Algorithm 1 Brute Force Algorithm

1: procedure BRUTEFORCE(G) return $List\langle int \rangle$	
2: $List\langle int \rangle$ colors = newList(int)	▷ List of colors
3: int maxColors = 0	▷ Number of colors
4: int maxColorsIndex = 0	▷ Index of the best solution

Algorithm 1 Brute Force Algorithm

```

5:  int  $i = 0$                                 ▷ Index of the current solution
6:  int  $n = G.GetNumberOfNodes()$                 ▷ Number of nodes
7:  int  $m = G.GetNumberOfColors()$                 ▷ Number of colors
8:  int  $max = pow(m, n)$                             ▷ Number of possible solutions
9:  while  $i < max$  do
10:      $colors = G.GetColors()$ 
11:     int  $currentColors = colors.Count()$ 
12:     if  $currentColors > maxColors$  then
13:          $maxColors = currentColors$ 
14:          $maxColorsIndex = i$ 
15:      $i++$ 
16:      $colors = G.GetColors(maxColorsIndex)$ 
17: return  $colors$ 

```

2.2 Algoritmul Greedy

Algoritmul Greedy este un algoritm care alege mereu cea mai bună soluție la momentul curent, fără a ține cont de soluțiile viitoare. Un pseudo-cod al algoritmului este prezentat în Pseudo codul 2.

Algorithm 2 Greedy Algorithm

```

1: procedure GREEDY( $G$ )
Require:  $S =$  Class Set,  $V =$  Colorless vertices in random order
Ensure:  $S = \emptyset$ 
2:  for  $v \in V$  do
3:     for  $i$  to  $S.length$  do
4:         if  $NonConfliativeEdges(v \cup S_i)$  then
5:              $AssignClass(v, S_i)$ 
6:              $NextVertex$ 
7:         if  $NotColored(v)$  then
8:              $S_{i+1} = NewClass$ 
9:              $AssignClass(v, S_{i+1})$ 
10:          $Next\ Vertex$ 

```

În algoritmul Greedy, așa cum se poate vedea în pseudo cod, se folosește o permutare inițială care este generată aleatoriu și în care este colorat fiecare vârf, comparând în fiecare caz dacă acesta poate fi inclus într-o culoare fără a provoca conflicte. Pentru a analiza algoritmul în cel mai rău caz, vom presupune că fiecare vârf este verificat cu fiecare culoare înainte de a fi colorat, ce ne dă un polinom care este

$$P(x) = n(n+1)/2. \quad (1)$$

Polinomul se obține din faptul că fiecare vârf, înainte de a fi colorat, trebuie să se compare cu toate culorile actuale:

$$P = 0 + 1 + 2 + \dots + n = n(n + 1)/2 \quad (2)$$

În cele din urmă, vom avea o complexitate de $O(n^2)$, unde n este numărul de noduri. Este necesar să afirmăm că algoritmul Greedy, deși calculează numărul cromatic aproximativ, acest lucru va depinde direct de permutarea inițială pe care o avem. Cu algoritmul următor vom vedea tehnici care ne permit să îmbunătățim modul de a alege vârfurile pentru a obține soluții mai bune și mai apropiate de numărul cromatic.

2.3 Algoritmul DSatur

Algoritmul DSatur este un algoritm care alege mereu nodul cu cea mai mare valoare de saturație, adică nodul care are cel mai mare număr de vecini cu aceeași culoare. Un pseudo-cod al algoritmului este prezentat în Pseudo codul 3.

Algorithm 3 DSatur Algorithm

```

1: procedure DSATUR( $G$ )
Require:  $S$  = Class Set,  $V$  = Non-colored vertices
Ensure:  $S = \emptyset$ 
2:   while  $V \neq \emptyset$  do
3:      $v = \text{MaxSaturation}(V)$ 
4:     for  $i$  to  $S.\text{length}$  do
5:       if  $\text{NonConfliativeEdges}(v \cup S_i)$  then
6:          $\text{AssignClass}(v, S_i)$ 
7:         Next Vertex (break)
8:       if  $\text{NotColored}(v)$  then
9:          $S_{i+1} = \text{NewClass}$ 
10:         $\text{AssignClass}(v, S_{i+1})$ 
11:         $\text{Remove}(v, V)$ 
12:        Next Vertex

```

Se poate observa din pseudocod că acest algoritm este foarte asemănător cu algoritmul Greedy, însă puterea algoritmului DSatur constă în prioritatea dată nodurilor cu cea mai mare saturație. Astfel, aceste noduri mai restrânse sunt colorate înainte de restul nodurilor, care nu au aceste “restricții”. Complexitatea algoritmului este similară cu cea a algoritmului Greedy, deoarece este foarte asemănător în cod, însă are și calcule suplimentare pentru a obține saturația nodurilor, însă acestea nu afectează direct complexitatea totală a algoritmului, care este totuși $O(n^2)$.

Trebuie punctat faptul că, deși algoritmul DSatur are un performanță mai bună în general pentru a da soluții mai apropiate de numărul cromatic, acesta are

și cazuri în care obține o soluție mai proastă decât algoritmul Greedy. Aceasta se poate observa în [15] unde se arată că o distribuția neadecvată a vârfurilor este dificilă de colorat cu algoritmul DSatur.

2.4 Algoritmul RLF

Algoritmul RLF caută să coloreze toate vârfurile disponibile în momentul în care acestea nu afectează niciun conflict al aceleiași culori. De asemenea, acordă prioritate vârfurilor care au un grad mai mare, acordând astfel prioritate tipului nostru de restricție prevăzut pentru această euristică. Algoritmul RLF are o complexitate în cel mai rău caz mai mare decât cea a algoritmului Greedy sau DSatur, aceasta fiind o complexitate dovedită și de Leighton (1979) - $O(n^3)$. Cu toate acestea acest algoritm are și o îmbunătățire a calității soluțiilor pe care le oferă, deși la un cost de calcul mai mare. Mai jos se poate observa pseudo codul ??alg4) algoritmului RLF.

Algorithm 4 RLF Algorithm

```

1: procedure RLF( $G$ )
Require:  $S$  = Class Set,  $V$  = Competent vertices,  $W$  = Non-Competent vertices
Ensure:  $S = \emptyset$ ,  $V$  = Vertices of a given graph,  $W = \emptyset$ 
2:   while  $V \neq \emptyset$  do
3:      $S_i = NewClass$ 
4:     while  $V \neq \emptyset$  do
5:        $v = MaxSaturation(V)$ 
6:        $AssignClass(v, S_i)$ 
7:        $Add(Adjacent\ Vertices(v), W)$ 
8:        $Remove(Adjacent\ Vertices(v), W)$ 
9:      $V = W$ 
10:     $W = \emptyset$ 

```

3 Evaluare

3.1 Grafuri de test și execuția algoritmilor

Pentru a evalua performanța algoritmilor de colorare, am folosit grafuri de test generate cu ajutorul unui generator de teste de pe Github ce îl puteți accesa aici [16]. Am generat 30 de teste prezentate în tabelul 1 pentru toți algoritmii prezentați mai sus.

numărul testului	tipul grafului	numărul de noduri (N)	numărul de muchii (M)
1	graf gol	887	0
2		1384	0
3	graf complet	10	45
4		50	1225
5		100	4950
6		500	124750
7		1000	499500
8		2000	1999000
9	graf bipartit complet	10	21
10		50	504
11		100	1771
12		500	35275
13		1000	164151
14		2000	557775
15	arbore binar	10	9
16		50	49
17		100	99
18		500	499
19		1000	999
20		2000	1999
21	graf planar	10	20
22		50	100
23		100	200
24		500	1000
25		1000	2000
26		2000	4000
27	graf cordal	10	
28		50	
29		100	
30		500	
31		1000	
32		2000	
33	graf cu număr de muchii fixat	1383	10
34		886	100
35		777	500
36		915	1000
37		1793	10000
38		1386	100000
39		1421	500000
40		1763	1000000

Table 1: Grafuri de test

3.2 Specificațiile sistemului de calcul

Codul a fost builduit în C++, iar fiecare dintre algoritmii prezentați mai sus au fost evaluați folosind un Laptop cu următoarele specificații:

- Procesor: Intel(R) Core(TM) i7-10750H CPU @2.60GHz 2.59 GHz
- Memorie RAM: 16 GB
- Sistem de operare: Windows 10 Pro 64-bit
- Versiunea compilatorului: GCC 9.4.0
- Versiunea C++: C++17
- Versiunea IDE: Visual Studio 2019
- Versiunea CMake: 3.19.2

References

1. J. Bondy and U. Murty, Graph Theory - Graduate Texts in Mathematic. Springer, 2008.
2. Z. Ádám Mann and A. Szajkó, “Average-case complexity of backtrack search for coloring sparse random graphs,” Journal of Computer and System Sciences, vol. 79, no. 8, pp. 1287–1301, 2013.
3. “Backtrack: An $o(1)$ expected time algorithm for the graph coloring problem,” Information Processing Letters, vol. 18, no. 3, pp. 119–121, 1984.
4. N. Barnier and P. Brisset, “Graph coloring for air traffic flow management,” Annals of Operations Research, vol. 130, 03 2002.
5. “The application of a graph coloring method to an examination scheduling problem,” Institute for Operations Research and the Management Sciences (INFORMS), vol. 11, no. 5.
6. University Exam Scheduling System Using Graph Coloring Algorithm and RFID Technology
7. A. Murat and B. Nurdan, “A performance comparison of graph coloring algorithms,” International Conference on Advanced Technology Sciences (ICAT’16), vol. 4, pp. 1–19, 12 2016
8. Z. Mann, “Complexity of coloring random graphs: An experimental study of the hardest region,” Journal of Experimental Algorithmics, vol. 23, pp. 1–19, 03 2018
9. Enunțul problemei
10. M. Garey and D. Johnson, Computer and Intractability: A Guide to the Theory of NP-Completeness, 01 1979.
11. D. Porumbel, J.-K. Hao, and P. Kuntz, “An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring,” Computers Operations Research, vol. 37, pp. 1822–1832, 10 2010.
12. L. Ouerfelli and H. Bouziri, “Greedy algorithms for dynamic graph coloring,” 2011 International Conference on Communications, Computing and Control Applications, CCCA 2011, 03 2011.
13. Á. E. Eiben, J. K. Van Der Hauw, and J. I. van Hemert, “Graph coloring with adaptive evolutionary algorithms,” Journal of Heuristics, vol. 4, no. 1, pp. 25–46, 1998.
14. D. Brèláz, “New methods to color the vertices of a graph,” Commun. ACM, vol. 22, pp. 251–256, 04 1979.

15. R. Janczewski, K. Manuszewski, and K. Piwakowski, "The smallest hard-to-color graph for algorithm dsatur," *Discrete Mathematics*, vol. 236, pp. 151–165, 06 2001.
16. GitHub - Jngen: preparing tests made simpler
17. Graph coloring - Wikipedia
18. A Comparison of Parallel Graph Coloring Algorithms
19. A Performance Comparison of Graph Coloring Algorithms
20. Moodle - Analiza algoritmulor