Graph Coloring
Proiect Analiza Algoritmilor

Mircea-Andrei Beznea, Ștefan Ghenescu, Robert-Fabian Tudor

Facultatea de Automatică și Calculatoare
Universitatea Politehnica din București

1 Introducere

1.1 Descrierea și istoria problemei

Colorarea grafurilor (*graph coloring* în engleză) este o problemă ce presupune colorarea vârfurilor unui graf astfel încât să nu existe două noduri adiacente cu aceeași culoare. În acest context, definim numărul minim de culori necesare pentru acolora nodurile unui graf ca fiind *numărul cromatic* al grafului.

Problema poate fi formulată atât ca o problemă de decizie, cât și ca o problemă de optimizare:

- O **problemă de decizie** este formulată astfel: „Dat fiind un graf G și K culori, este posibil să existe o schemă de colorare validă?”
- O **problema de optimizare** este formulată astfel: „Dat fiind un graf G, găsiți numărul minim de culori necesare pentru colorarea grafului.”

Deși ambele tipuri se încadrează în același context, ele diferă ca nivel de complexitate: problema de decidabilitate este NP-complete, în timp ce găsirea numărului cromatic este NP-hard.

Graph Coloring Problem a apărut pentru prima dată în contextul colorării hărților, când Francis Guthrie a postulat în 1852 că patru culori sunt suficiente pentru a colora orice hartă în așa fel încât regiunile adiacente să aibă culori diferite. Aceasta este cunoscută drept Problema celor Patru Culori și a fost transmisă de fratele său profesorului Augustus de Morgan, care a popularizat întrebarea în comunitatea matematică. Problema a rămas deschisă timp de peste un secol, fiind demonstrată abia în 1976 de Kenneth Appel și Wolfgang Haken, cu ajutorul calculatoarelor.

1.2 Aplicații practice

Programarea ședințelor din cadrul unei companii. Să presupunem că o companie are mai multe echipe și dorim să programăm ședințele acestora. Dacă o persoană face parte din mai multe echipe, atunci ședințele respective trebuie să fie programate în intervale de timp diferite.

Această problemă poate fi modelată printr-un graf *G*, în care fiecare echipă este reprezentată de un nod, iar două noduri sunt conectate printr-o muchie dacă echipele corespunzătoare au cel puțin un membru comun. Colorarea nodurilor grafului cu *K* culori, astfel încât nodurile conectate să primească culori diferite, reprezintă o alocare validă a intervalelor de timp pentru ședințe.

Alocarea frecvențelor radio. În cazul posturilor de radio, fiecare post trebuie să opereze pe o anumită frecvență fără a interfera cu altele din apropiere. Dacă două posturi transmit pe aceeași frecvență și au zone de acoperire care se suprapun, semnalele lor pot interfera, afectând calitatea transmisiunii.

Problema poate fi modelată printr-un graf *G*, în care nodurile reprezintă posturile de radio, iar muchiile sugerează că 2 posturi au zone de acoperire care se suprapun și pot interfera. O colorare corectă a grafului cu *K* culori echivalează cu alocarea a *K* frecvențe astfel încât posturile apropiate să folosească frecvențe diferite, eliminând riscul de interferență și minimizând numărul de frecvențe folosite.

Alocarea variabilelor în regiștrii procesorului. În arhitectura calculatoarelor, compilatoarele trebuie să aloce variabilele unui program în regiștrii procesorului. Datorită numărului de regiștri finit și a numărului mare de variabile, este necesar să asociem mai multe variabile unui singur registru, cu condiția ca acestea să nu fie utilizate simultan.

Problema poate fi modelată printr-un graf *G*, unde nodurile reprezintă variabilele, iar muchiile conectează variabile care sunt active în același timp (și, prin urmare, nu pot ocupa același registru). O colorare corectă a grafului cu *K* culori indică alocarea regiștrilor astfel încât să evităm conflictele, optimizând utilizarea acestora.

2 Demonstrație NP-Hard

Pentru a demonstra apartenența problemei K-Colorare la clasa NP-Hard vom demonstra întâi că problema 3-SAT (despre care știm de la curs că este NP-Hard) se reduce polinomial la problema 3-Colorare. Ulterior, demonstrăms că problema 3-Colorare se reduce polinomial la K-Colorare și astfel demonstrația va fi completă.

Observație: Pe tot parcursul demonstrației considerăm că V reprezintă numărul de noduri din graf, iar E numărul de muchii din graf.

2.1 3-Colorare ∈ NP-Hard

Pentru demonstrație arătăm că 3-SAT $<_p$ 3-Coloare. Astfel construim un graf convenabil astfel încât dacă acesta va fi colorat cu 3 culori atunci si problema 3-SAT va fi adevărată pe intrarea respectivă și viceversa.

Construirea grafului:

- Adaugăm 3 noduri legate între ele (formând un triunghi) colorate in "T", "F" și "B"[1] (ne trebuie 3 culori pentru a respecta regula de la 3-Colorare)
- Pentru fiecare dintre cei 3 literali construim 2 noduri asociate stărilor x_i, $!x_i$ și formăm triunghiurile x_i $!x_i$ B (Fig. 1)

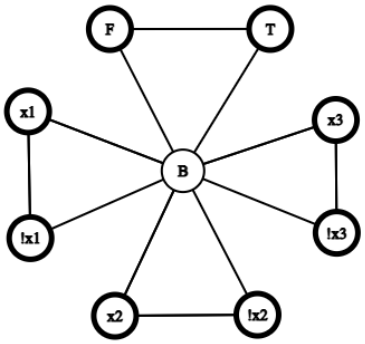


Fig. 1.
Observăm că unul dintre x_i, $!x_i$ trebuie sa fie neapărat true, fiind astfel colorat cu T. Negatul va fi colorat cu F, problema 3-Colorare fiind adevărată pe acest graf. Astfel, ne asigurăm că vor avea mereu culori (valori) diferite.

- Pentru a demonstra implicația de la dreapta la stânga trebuie să extindem acest graf. Intervine următoarea constructie, numită OR-Gadget (Fig. 2):
  - Pentru fiecare cluster (ex. c = (x_1 V x_2 V x_3)) adăugăm 5 noduri formandu-se următoarea configurație

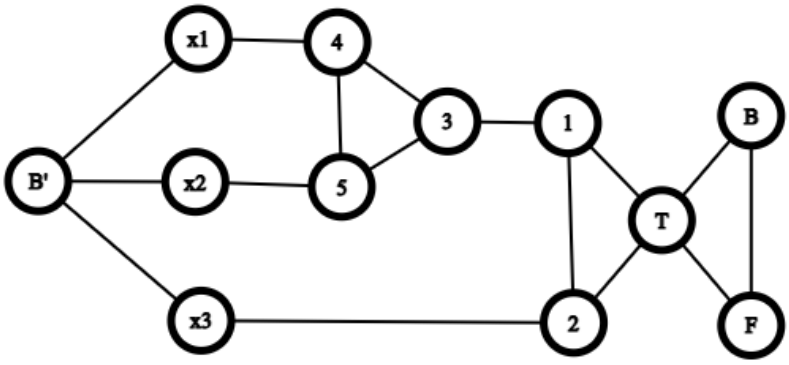


Fig. 2. B' și B reprezintă același nod

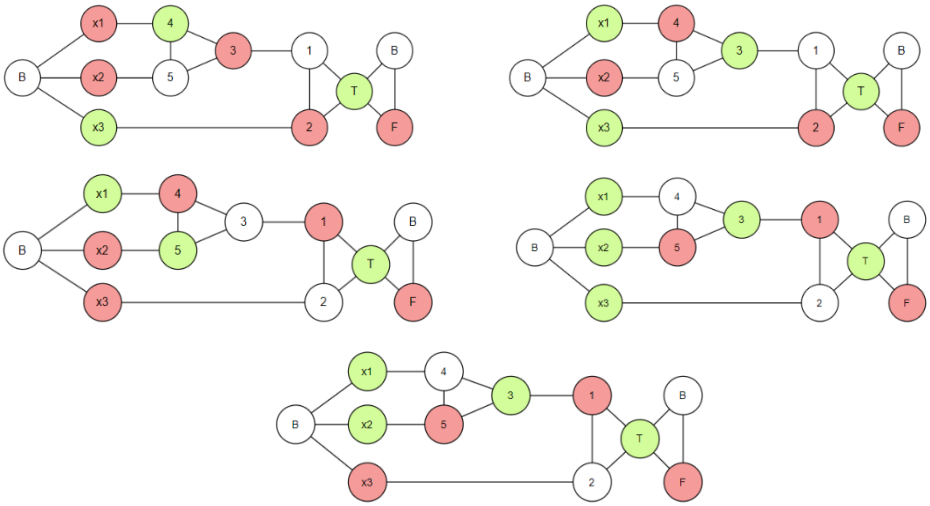
  - Nodurile B, T, F sunt aceleasi noduri ca cele din graful de bază (Fig. 1)

[1] T va fi culoarea asociată stării True, F – False, B – Base (referința grafului)

3-SAT → 3-Colorare. Cum 3-SAT este adevărată înseamnă că în fiecare cluster cel puțin unul dintre literali este adevărat (colorat cu T).

Demonstrăm că OR-Gadget se poate colora cu 3 culori plecând de la implicația dată. Este suficient să considerăm 2 cazuri având în vedere că literalii x1, x2 sunt așezați simetric.

1. x3 – True => cazurile: $(x1, x2) \in \{(F, F), (T, F), (T, T)\}$
2. x1 – True => cazurile: $(x2, x3) \in \{(F, F), (T, F)\}$



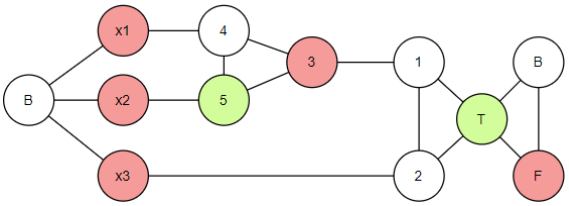
Grafurile din figură reprezintă structura unui OR-Gadget. Fiecare nod este colorat astfel încât niciun nod adiacent nu are aceeași culoare, ceea ce respectă condițiile problemei 3-Colorare. Aceste soluții arată că există o colorare validă folosind doar 3 culori.

Literalii din OR-Gadget-uri sunt de fapt cei din graful de bază. Cu alte cuvinte, dacă în mai multe clauze apare același literal, acesta reprezintă un singur nod în graf, asigurându-se astfel colorarea (valoarea) consecventă a literalului în toate clauzele.

Așadar, conform observațiilor făcute referitor la construcția de la Fig. 1 și cum grafurile OR-Gadget reprezintă subgrafuri ce pot fi colorate cu 3 culori, atunci graful extins poate fi colorat cu 3 culori, problema 3-Colorare fiind adevarată la rândul ei.

3-Colorare → 3-SAT. Graful de bază (Fig. 1) ne asigura colorarea true si false a lui x_i și $!x_i$. Pentru a demonstra că 3-SAT este adevărată este suficient să arătăm că in fiecare graf OR-Gadget cel putin unul dintre nodurile asociate literalilor este colorat true.

Presupunem prin absurd că toți literalii dintr-o clauza sunt falși.



Se observă faptul că nu putem colora cu 3 culori construcția OR-Gadget (nodul 1 si 2 au aceeasi culoare).

Contradicție cu faptul că putem colora graful cu 3 culori. Astfel am demonstrat că în orice clauză, cel puțin unul dintre literali trebuie să fie adevărat, deci automat problema 3-SAT o sa fie adevărată.

De asemenea, construcția grafului se poate realiza în timp polinomial.
În concluzie, conform celor două implicații 3-SAT $<_p$ 3-Coloare.

2.2 3-Colorare $<_p$ K-Colorare

Construim un program care să ne ia intrarea de la 3-Colorare și să întoarcă o intrare convenabilă pentru K-Colorare.

$$F : G \rightarrow G$$

La trecerea de la 3-Colorare la K-Colorare, adaugăm k - 3 noduri noi, unite cu toate nodurile.

```
def F(G: tuple):
    V, E = G
    # V cont, ține noduri numerotate de la 0 la len(V) - 1
    # E cont, ține muchiile, de tip (nod1, nod2)
    offset = len(V)
    for i in range(k - 3):
        V.append(i + offset)
        E = E + [(v, i + offset) for v in V] # unim nodul cu toate celelalte noduri din graf
    return (V, E)
```

Complexitatea temporală a lui F este polinomială.

3-Colorare → K-Colorare. *3-Colorare(G) = 1 ⇒ K-Colorare(F(G)) = 1*

Pentru orice situație în care graful se poate colora cu 3 culori, nodurile noi vor trebui colorate cu o singură altă culoare, deci va respecta K-Colorare.

3-Colorare ← K-Colorare. *3-Colorare(G) = 1 ⇐ K-Colorare(F(G)) = 1*

Avem nodurile noi (numerotate de la V pana la V + k - 4) care sunt conectate la toate celelalte noduri, deci trebuie sa aibă o culoare diferită față de toate. Pentru că problema este K-Colorabilă, înseamnă că, dacă scoatem aceste noduri, graful care rămâne (cel original) este 3-Colorabil.

3 Prezentare Algoritmi

3.1 Descrierea algoritmilor aleși

Pentru toți algoritmii considerăm știut că V reprezintă numărul de noduri din graf, iar E numărul de muchii din graf.

Algoritmul Greedy. Colorarea greedy (numită și colorare secvențială) este o metodă de atribuire a culorilor nodurilor unui graf folosind un algoritm ce rulează în timp liniar.

Acesta procesează nodurile grafului într-o anumită ordine și atribuie fiecărui nod prima culoare disponibilă, astfel încât nodurile adiacente să aibă culori diferite. Deși acest algoritm poate fi executat în timp liniar, el nu garantează utilizarea unui număr minim de culori.

De asemenea, ordinea în care sunt procesate nodurile influențează puternic eficiența. Există întotdeauna o ordine care produce o colorare optimă, dar identificarea acestei ordini este dificilă. Așadar, algoritmul Greedy este eficient în din punct de vedere al complexității temporale, însă soluția produsă nu este garantat optimă datorită modului în care sunt parcurse nodurile (la Greedy nodurile sunt procesate în ordine crescătoare). Pentru soluții mai rapide, trebuie folosite metode mai avansate de alegere a ordinii în care sunt parcurse nodurile cum ar fi:

- procesarea nodurilor de grad mai mare înaintea celor de grad mai mic
- alegerea nodurilor mai constrânse (cu mai puțini vecini deja colorați) în detrimentul celor mai puțin constrânse

Problema cu aceste euristici, deși sunt mai rapide, nu garantează o soluție optimă, putând genera o colorare care să folosească mai multe culori decat numărul minim cu care poate fi colorat graful (numărul cromatic va fi greșit).

Algoritmul începe cu un graf necolorat, unde fiecare nod este asociat cu o valoare de culoare care poate fi inițializat la „-1” (pentru a arăta că nu a fost colorat). Apoi, programul selectează un nod pentru a-l colora. De obicei, acest nod este ales într-o ordine specificată (în exemplul nostru am ales să parcurgem nodurile în ordine crescătoare). După ce un nod este ales, algoritmul parcurge toți vecinii acestuia și „marchează” culorile deja atribuite acestora. Astfel, culoarea folosită de fiecare vecin este marcată ca fiind „indisponibilă” pentru nodul curent. După aceea, algoritmul caută prima culoare disponibilă pentru nodul curent și i-o atribuie. La final, culorile „marcate” anterior sunt resetate pentru a deveni disponibile pentru restul iterațiilor.

Algoritmul Welsh Powell. Soluția propusă de Powell reprezintă o îmbunătățire a algoritmului de colorare greedy, utilizând o strategie mai eficientă de procesare a nodurilor. În acest sens, nodurile sunt sortate în ordine descrescătoare după gradul lor (numărul de vecini), astfel încât cele cu mai mulți vecini să fie colorate primele.

Această abordare are rolul de a reduce numărul de culori necesare, deoarece procesarea inițială a nodurilor cu grad ridicat minimizează riscul de a aloca culori suplimentare atunci când sunt disponibile mai multe opțiuni pentru nodurile cu grade

mai mici. În plus, algoritmul poate fi adaptat cu ușurință pentru a determina numărul cromatic al unui graf.

Complexitatea acestuia este influențată de doi pași principali: sortarea nodurilor și atribuirea culorilor. Sortarea nodurilor după grad se realizează în $O(V \log V)$. Atribuirea culorilor presupune parcurgerea listei de vecini pentru fiecare nod, ceea ce implică o complexitate totală de $O(E)$.

Astfel, complexitatea finală a algoritmului este $O(V \log V + E)$, ceea ce îl face eficient atât pentru grafuri sparse, cât și pentru grafuri dense.

Algoritmul începe prin inițializarea unei liste în care fiecare nod este asociat cu culoarea „-1”, indicând că nu a fost colorat. Gradul fiecărui nod este calculat, reprezentând numărul de vecini, iar nodurile sunt sortate în ordine descrescătoare după grad. Nodurile sunt procesate unul câte unul, începând cu cel mai mare grad. Nodului de la pasul curent i se atribuie prima culoare care nu a fost utilizată la un pas anterior, dacă nu a fost deja colorat. Apoi, cu aceeași nuanță colorăm restul nodurilor care respectă următoarele condiții:

- nodul nu a fost deja colorat
- nodul nu are vecini colorați în nuanța curentă (aici se verifică implicit și că nodul nu este vecin cu nodul curent)

Soluția Brute Force (Backtracking). Spre deosebire de algoritmii prezentați anterior, soluția brute force pentru colorarea unui graf constă în generarea tuturor combinațiilor posibile de culori pentru nodurile grafului și verificarea fiecăreia pentru a determina dacă respectă cerința problemei: nicio pereche de noduri adiacente nu trebuie să aibă aceeași culoare.

Această metodă utilizează o strategie de tip *„trial and error”* bazată pe backtracking pentru a găsi o soluție validă, fiind astfel potrivită pentru determinarea numărului cromatic al unui graf. Deși soluția cu backtracking garantează găsirea unei soluții, complexitatea sa este exponențială, în general $O(K^V)$, unde K este numărul de culori, iar V reprezintă numărul de noduri. Din acest motiv, algoritmul este eficient pentru grafuri mici, însă devine ineficient pentru grafuri mari sau dense.

Algoritmul pornește prin inițializarea unei liste în care fiecare nod este asociat cu valoarea „-1”, indicând că nu a fost colorat. Nodurile sunt procesate recursiv, pornind de la primul, iar pentru fiecare nod algoritmul încearcă să-i atribuie o culoare dintr-un set de culori disponibile.

Pentru un nod dat, algoritmul verifică dacă o anumită culoare poate fi atribuită, fără a intra în conflict cu vecinii săi. Această verificare constă în parcurgerea listei de vecini ai nodului curent și compararea culorilor acestora cu cea pe care dorim să o atribuim. Dacă nu există conflicte, culoarea este atribuită temporar nodului. După atribuirea unei culori unui nod, algoritmul trece la următorul nod și reia procesul. Dacă algoritmul ajunge într-o situație în care un nod nu poate fi colorat respectând regula, acesta revine la nodul anterior (backtracking) și încearcă să-i atribuie o altă culoare. Acest proces continuă până când fie toate nodurile sunt colorate cu succes, fie algoritmul trece la a încerca să coloreze nodurile cu mai multe culori până când găsește numărul de culori necesare colorării grafului. Această soluție asigură găsirea corectă a numărului cromatic.

3.2 Analiza complexitații soluțiilor

Amintesc următoarele notații: V reprezintă numărul de noduri din graf, iar E numărul de muchii din graf.
Complexitatea algoritmilor va fi analizată prin studiul cazurilor:

- worst-case
- best-case
- mid-case.

Algoritmul Greedy. Această soluție este de tip euristică, iar complexitatea este polinomială. Intuiția este că pe măsură ce numărul de muchii crește (graf dens) va crește si complexitatea de calcul a algoritmului. În esență pentru a colora un nod trebuie să trec prin toți vecinii lui (să trec prin toate muchiile), cum graful este neorientat fiecare muchie o să fie vizitată de 2 ori (o dată de la v1 la v2 și o dată de la v2 la v1). Astfel, trecând de 2 ori prin muchii voi putea găsi soluția problemei rezultând la general o complexitate de *O(E)*. De multe ori însă este util să ne referim la complexitate funcție de număr de vârfuri, deoarece nu știm nimic despre poziționarea muchiilor.

Worst-case. Acest caz se obține pentru un graf complet (secțiune robert). Așa cum am spus mai sus este suficient și necesar să trecem de două ori prin toate muchiile care, pentru un graf complet, sunt în număr de *V(V-1) / 2*. Astfel vom executa *c * V(V-1)* operații, rezultând o complexitate de $O(V^2)$.

Best-case. Acest caz se obține pentru un graf nul (fără muchii). În funcție de implementare, acest caz poate fi tratat separat, rezultatul fiind astfel obținut în *O(1)*.

Mid-case. Vom considera acum că numărul de muchii este jumătate din numărul posibil de muchii rezultând *V(V-1) / 4* muchii, deci în total vom executa *c * V(V-1) / 2* operații, rezultând similar ca la Worst-case, o complexitate pătratică.

Avantaje / dezavantaje Greedy. Acest algoritm este performant în contextul în care vrem o soluție rapidă, nu neapărat optimă. Ne dorim ca soluția să fie cât mai apropiată de cea optimă, însă această euristică simplă produce un rezultat nesatisfăcător în momentul în care exista noduri care au mult mai multi vecini față de celelalte noduri din graf, acestea necesitând de cele mai multe ori culori suplimentare. Pentru a depăși acest neajuns este introdus următorul algoritm, tot de tip euristic.

Algoritmul Welsh-Powell. Această soluție este de tip euristică, iar complexitatea este polinomială. Intuiția este asemănătoare cu cea de la greedy, însă de această dată vom ține cont de gradul nodului.

Sortarea vine la pachet cu un efort de calcul suplimentar, dar care de multe ori poate genera o soluție corectă față de un greedy obișnuit care deși este mai rapid, soluțiile eronate sunt mai întâlnite, referința fiind soluția dată de backtracking. Calcularea gradelor nodurilor este O(V), pentru fiecare nod operația fiind executată în *O(1)*. Mai

apoi, sortarea nodurilor după gradul lor are o complexitate de *O(V logV)* indiferent de tipul de graf, ceea ce conduce până acum la o complexitate de *O(V logV);* luând în considerare calcularea gradelor. Colorarea nodurilor depinde de numărul de muchii, ceea ce conduce la o analiză în funcție de tipul de graf. Pentru colorarea unui nod suntem nevoiți să trecem prin vecinii acestuia și astfel contează gradul nodului. Astfel, trecând de 2 ori prin muchii putem găsi soluția problemei rezultând la general o complexitate de *O(E).* Cu cât gradul nodurilor este mai mare cu atât crește complexitatea algoritmului.

Worst-case. În acest caz, grafurile complete conduc la cea mai mare complexitate. Numarul de muchii este *V(V-1)/2,* iar fiecare nod are gradul *V-1*. Din această cauză complexitatea este $O(V \log V + V^2) = O(V^2)$.

Best-case. Acest caz se obține pentru un graf nul (fără muchii), rezultatul putând fi obținut în *O(1).* Alte cazuri rapide sunt cele în care graful este cât mai rar (am puțini vecini de vizitat pentru fiecare nod), complexitatea fiind $O(V \log V + E) \approx O(V \log V)$, $E << V \log V$.

Mid-case. . La general, pentru un graf mediu (semi-spars) numărul mediu de muchii este *V(V-1)/4*. Coloarea nodurilor este *O(V(V-1)/2)* iar sortarea *O(V logV).* Complexitatea algoritmului este prin urmare $O(V^2)$.

Avantaje / dezavantaje Welsh-Powell. Complexitatea este aceeași pentru mid-case ca la Greedy, oferind în plus un procentaj mai mare de soluții optime. Este util atunci când avem grafuri mari și dorim un timp de execuție scăzut și nu vrem să periclităm soluția optimă la fel de des ca la Greedy.

Pe de altă parte, nici acest algoritm nu oferă o siguranță în găsirea numărului cromatic. Apare un efort de calcul suplimentar din cauza sortării, care nu întotdeauna vine cu o îmbunătățire în găsirea soluției optime (când gradele nodurilor este același sau foarte apropiate).

Algoritmul backtracking (brute-force). Se pot implementa mai multe soluții de tip backtracking. Forma clasică a lui este aceea în care se generează toate colorările posibile selectându-se dintre acestea cea optimă (cu cele mai puține culori folosite).

Implementarea pe care o propunem noi este însă una puțin schimbată. Pe parcursul algoritmului vom considera cunoscut numărul cromatic, apelând incremental pentru această valoare algoritmul BKT. Când algoritmul reușește colorarea tuturor nodurilor înseamnă că nu mai este nevoie generarea soluțiile care folosesc un număr de culori mai mare deoarece am găsit numărul cromatic. Astfel, algoritmul se oprește fără a genera toate soluțiile posibile. Soluția prezentată mai devreme vine cu o îmbunătățire: pentru un număr cromatic mic, acest backtracking modificat va rula mult mai repede decât un backtracking normal. Cum numărul cromatic este strâns legat de numărul de muchii, propoziția anterioară este echivalentă cu a spune că pentru un graf rar, backtracking-ul va genera un răspuns garantat optim într-un timp mult mai scurt față

de cum ne-am aștepta în mod normal pentru astfel de algoritmi. Complexitatea unui apel de backtracking este cu atât mai mare cu cât numărul permis de culori (C) este mai mare, dar mai mic strict decât numărul cromatic (algoritmul va ajunge în punctul în care realizează că soluția este gresită mult mai greu).

Worst-case. Dacă graful este complet va trebui să executăm backtracking de V ori. Astfel complexitatea algoritmului pentru un graf va fi suma tuturor apelurilor ce consideră numărul cromatic de la 1 la V, rezultând astfel o complexitate de $O(V^{V+1})$.

Best-case. Cazurile bune sunt cele în care graful este cât mai rar, adică în care numărul cromatic este cât mai mic. Vom vedea în cele ce urmează că pentru un graf bipartit, complexitatea ajunge să fie chiar liniară, $O(V)$.

Mid-case. La general, pentru un graf mediu (semi-spars) soluția va fi descoperită după un efort de calcul exponențial.

Avantaje / dezavantaje BKT. Backtracking vine cu o garanție pe care niciun alt algoritm nu o aduce. În urma efortului de calcul vom ști că soluția gasită este cea optimă. De asemenea, pentru grafuri rare acest algoritm nu este neapărat depășit de euristicile prezentate anterior. Cu toate acestea, la general, pentru un graf mai dens, cu un număr mai mare de noduri se poate întâmpla să nu reușim niciodată să calculăm soluția (puterea de calcul nu este în prezent suficient de mare pentru a rula backtracking pe un număr mare de intrări)

4 Evaluare

4.1 Generarea testelor

Amintim următoarele notații: V reprezintă numărul de noduri din graf, iar E numărul de muchii din graf.

Criteriile de construire a setului de date. Pentru a valida implementările algoritmilor de colorare a grafurilor (Backtracking, Greedy, Welsh-Powell) am construit un set de date care evidențiază atât performanța, cât și corectitudinea acestora.

Ele sunt generate folosind un script scris în Python pentru a automatiza procesul de testare. Setul de teste a fost creat ținând cont de următoarele categorii de grafuri:

- **Grafuri rare**
  - **Definiție:** Grafuri cu un număr redus de muchii comparativ cu maximul posibil (V * (V - 1) / 2).
  - **Motivație pentru alegerea făcută:** Grafurile rare sunt folosite pentru a studia comportamentul algoritmilor în contexte cu densitate scăzută de muchii. Acest tip de graf reflectă rețele simple și structuri care apar frecvent în probleme practice, cum ar fi structurile ierarhice. Performanța algoritmilor pe aceste grafuri este relevantă deoarece numerele cromatice sunt, în general, ușor de

obținut, iar eficiența poate fi analizată fără complexitatea suplimentară introdusă de numărul mare de muchii. (arborii, lațurile, spre exemplu, necesită doar 2 culori).

- **Grafuri dense**
  - **Definiție:** Grafuri cu un număr de muchii apropiat de maximul posibil (V * (V - 1) / 2).
  - **Motivație pentru alegerea făcută:** Alegerea lor este importantă deoarece numărul cromatic al grafurilor dense tinde să fie mai mare, ceea ce face ca algoritmii să fie puși în fața unor provocări mai mari, scoțând în evidență diferențele de eficiență și corectitudine între metode prin complexitatea suplimentară introdusă de numărul mare de muchii.

- **Grafuri complete**
  - **Definiție:** Grafuri în care fiecare nod este conectat cu toate celelalte (numărul cromatic este egal cu numărul de noduri).
  - **Motivație pentru alegerea făcută:** Grafurile complete sunt esențiale pentru a testa comportamentul algoritmilor de colorare în scenarii extreme. În cazul algoritmului Welsh-Powell, aceste grafuri pun în evidență ineficiența introdusă de sortarea nodurilor cu grade egale. Deși soluția optimă este evidentă (fiecare nod necesită o culoare diferită), sortarea inutilă a nodurilor crește semnificativ timpul de execuție, ceea ce face ca acest tip de graf să fie un test relevant pentru a evalua performanța reală a algoritmului.

- **Grafuri ciclice**
  - **Definiție:** Grafuri închise în care fiecare nod este conectat la exact două alte noduri.
  - **Motivație pentru alegerea făcută:** Grafurile ciclice sunt o alegere interesantă în studiul permormației algoritmilor deoarece numărul cromatic depinde de paritatea numărului de noduri. Astfel, ciclurile de lungime pară au numărul cromatic 2, pe când cele de lungime impară au soluția egală cu 3**.** De asemenea, acest aspect influențează timpii de execuție ai algoritmului BKT.

- **Grafuri bipartite**
  - **Definiție:** Grafuri ale căror noduri pot fi împărțite în două mulțimi disjuncte astfel încât muchiile să existe doar între nodurile din mulțimi diferite.
  - **Motivație pentru alegerea făcută:** Grafurile bipartite reprezintă un caz interesant de studiu deoarece numărul cromatic este întotdeauna 2 datorită structurii specifice a acestora, în care nodurile pot fi împărțite în două seturi disjuncte, iar muchiile conectează doar noduri din seturi diferite.

- **Grafuri de tip drum (Path)**
  - **Definiție:** Grafuri simple în care nodurile sunt conectate linear.

  - **Motivație pentru alegerea făcută:** Grafurile de tip drum au fost alese pentru a evalua performanța algoritmilor de colorare în cazul structurilor simple și liniare, unde interacțiunea dintre noduri este minimă.

- **Grafuri grid 2D**
  - **Definiție:** Grafuri în care nodurile sunt aranjate într-o rețea bidimensională și sunt conectate la nodurile adiacente pe orizontală și verticală. (Ne putem imagina pătratele de o foie din caietul de matematică, în care colțurile fiecărui pătrat reprezintă nodurile, iar laturile muchiile).
  - **Motivație pentru alegerea făcută:** Grafurile grid 2D sunt alese pentru a testa algoritmii de colorare în contextul unei structuri unde fiecare nod are cel mult patru vecini. Ele sunt un exemplu interesant de grafuri planare ce au numărul cromatic maxim 2. Putem considera aceste grafuri ca fiind rare numarul maxim de muchii find < 4*V.

Numărul și modul de generare al testelor. Pentru validarea algoritmilor de colorare a grafurilor, am generat un număr diferit de teste pentru fiecare categorie, ținând cont de complexitatea structurii grafurilor și de dimensiunile lor**.**
În total, setul de teste cuprinde 361 de teste distribuite astfel:

- **Grafuri rare (sparse)** – 55 de teste
  - Sunt generate utilizând modelul Erdős–Rényi cu probabilități de muchie între 0.1 și 0.4.
  - Numărul de noduri variază între 1 și 11, iar pentru fiecare număr de noduri am generat mai multe teste cu probabilități diferite.

- **Grafuri dense** – 55 de teste
  - Sunt, de asemenea, generate folosind modelul Erdős–Rényi, dar cu probabilități de muchie între 0.7 și 1.
  - Numărul de noduri variază între 1 și 11, iar pentru fiecare dimensiune am creat mai multe grafuri dense cu probabilități diferite.

- **Grafuri complete** – 11 teste
  - Sunt generate pentru dimensiuni între 1 și 11 noduri.

- **Grafuri ciclice** – 80 de teste
  - Sunt construite cu dimensiuni care variază între 10 și 80 de noduri, pentru a acoperi atât cazuri simple, cât și complexe.

- **Grafuri bipartite** – 50 de teste
  - Sunt construite cu dimesiuni ale grafurilor care variază între 1 și 50 de noduri, iar împărțirea între mulțimi este diferită între teste.

- **Grafuri de tip drum (path)** – 80 de teste

- Sunt generate cu dimensiuni dimensiuni între 1 și 80 de noduri.

- **Grafuri grid** 2D – 30 de teste
  - Sunt realizate astfel încât dimensiunea grilei variază între 1×1 și 30×30, rezultând până la 900 de noduri pentru cele mai mari grafuri.

Grafurile au fost generate utilizând modulul NetworkX din Python, recunoscut pentru funcționalitățile sale avansate în modelarea și manipularea structurilor de grafuri. Pentru fiecare tip de graf în parte, au fost utilizate funcții specifice, adaptate pentru a răspunde cerințelor fiecărui scenariu de testare:

- **Grafuri rare și dense**
  - *nx.erdos_renyi_graph(V, p)* – Creează un graf cu V noduri și probabilitatea p pentru fiecare muchie.

- **Grafuri complete**
  - *nx.complete_graph(V)* – Creează un graf complet cu V noduri.

- **Grafuri ciclice**
  - *nx.cycle_graph(V)* – Creează un graf ciclic cu V noduri.

- **Grafuri bipartite**
  - *nx.complete_bipartite_graph(V1, V2)* – Creează un graf bipartit complet cu V1 noduri în prima mulțime și V2 în a doua.

- **Grafuri de tip drum**
  - *nx.path_graph(V)* – Creează un graf lanț cu n noduri.

- **Grafuri grid 2D**
  - Acest tip de grafuri au fost generate printr-un algoritm personalizat care conectează nodurile orizontal și vertical pe baza poziției lor.

Limitările algoritmilor Greedy și Welsh-Powell. În afară de setul de date menționat anterior, pentru a demonstra limitările algoritmilor de colorare a grafurilor, am construit trei cazuri simple care evidențiază situații în care algoritmii Greedy și Welsh-Powell pot produce soluții incorecte în raport cu numărul cromatic al grafului.

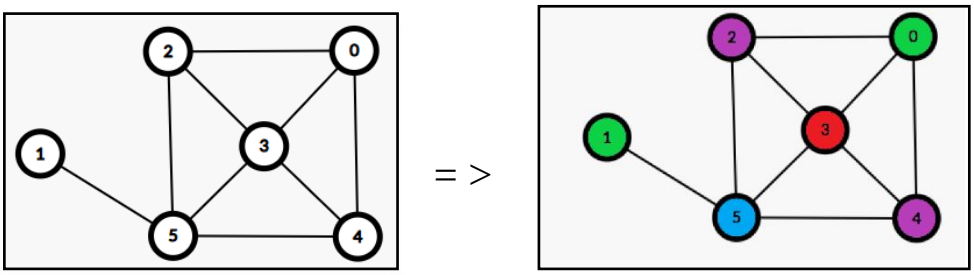
Aceste cazuri sunt relizate pentru a evidenția:

1. Situații în care algoritmul Greedy nu oferă o soluție corectă, utilizând mai multe culori decât numărul cromatic datorită ordinii în care sunt parcurse nodurile.
2. Situații în care algoritmul Welsh-Powell generează o soluție incorectă, datorită ordonării descrescătoare după grad.
3. Situații în care ambii algoritmi oferă soluții incorecte, nerespectând numărul cromatic al grafului.

Table 1. Tabel care să arate corectitudinea fiecărui algoritm.

Nume	Nr total teste	Nr teste esuate	% Succes
Greedy	100	7	93%
Welsh-Powell	100	1	99%
BKT	100	0	100%

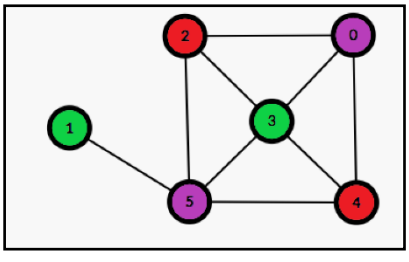
Cazul I.



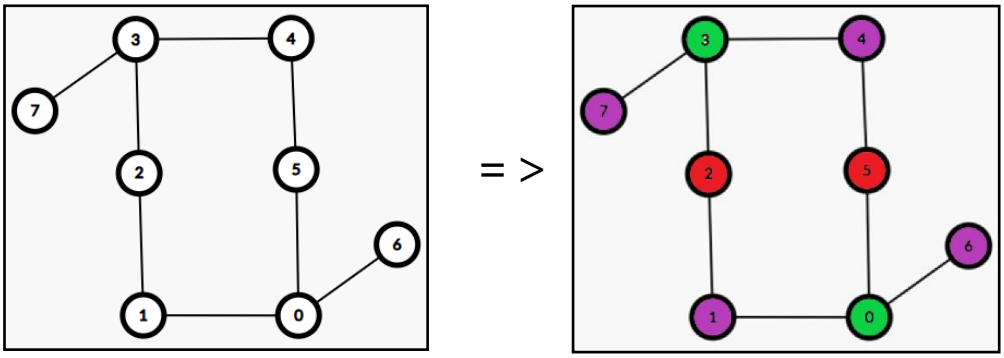
În acest caz, algoritmul Greedy se va comporta astfel:

- Parcurge nodurile în ordine crescătoare începând cu nodul 0 și verifică dacă există conflicte cu vecinii deja colorați:
  - Nodul 0 nu are niciun vecin colorat, deci va primi culoarea verde(0).
  - Același lucru se întâmplă cu nodul 1 care ia tot culoarea verde(0).
  - Când ajunge în nodul 2 deoarece nodul 1 a fost colorat anterior cu verde(0), acesta va lua culoarea mov(1).
  - Nodul 3 este următorul și va primi culoarea roșu(2) deoarece nodul 2 e colorat cu mov(1) și nodul 0 are culoarea verde(0).
  - În continuare, nodul 4 are vecini colorați cu verde(0) și roșu(2), deci va fi colorat cu prima culoare disponibilă adică mov(1).
  - În final, nodul 5 are vecini cu verde(0), mov(1) și roșu(2), deci îi atribuie o nouă culoare, albastru(3).
- Prin urmare, numărul cromatic obținut de Greedy este 4, deși cel corect este 3, acest caz concret demonstrând că ordinea de parcurgere din cadrul algoritmului Greedy poate duce la răspunsuri eronate.

Exemplu: Cu algoritmul Welsh-Powell, graful colorat ar arăta astfel:



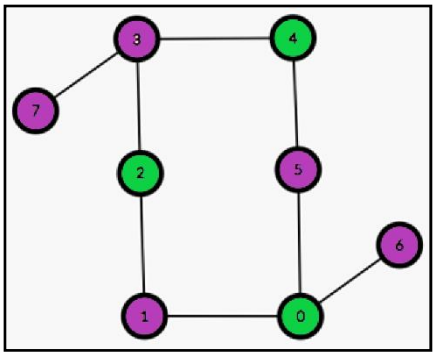
Cazul II.



În acest caz, algoritmul Welsh-Powell se va comporta astfel:

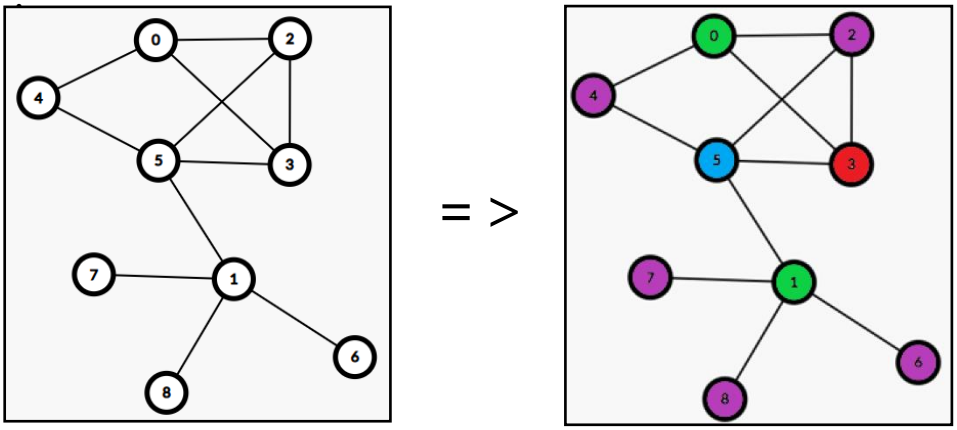
- La început, sortează nodurile în ordine descrescătoare după numărul de vecini:
  - 3 vecini – nodurile 0 și 3
  - 2 vecini – nodurile 1, 2, 4, 5
  - 1 vecin – nodurile 6 și 7
- Apoi, parcurge nodurile în această ordine începând cu nodul 0 și verifică dacă există conflicte cu vecinii deja colorați:
  - Nodul 0 nu are niciun vecin colorat, deci va primi culoarea verde(0).
  - Același lucru se întâmplă cu nodul 3 care ia tot culoarea verde(0).
  - Când ajunge în nodul 1 deoarece nodul 0 a fost colorat anterior cu verde(0), acesta va lua culoarea mov(1).
  - Nodul 2 este următorul și va primi culoarea roșu(2) deoarece nodul 1 e colorat cu mov(1) și nodul 3 are culoarea verde(0).
  - În continuare, nodul 4 are un vecin colorat cu verde(nodul 3), deci va fi colorat cu prima culoare disponibilă adică mov(1).
  - Nodul 5 are 2 vecini: nodul 0 colorat cu verde(0) și nodul 4 cu mov(1), așadar va fi colorat cu roșu(2).
  - În final, nodurile 6 și 7 au câte un singur vecin colorat cu verde(0), deci vor fi colorate cu mov(1).
- Prin urmare, numărul cromatic obținut de Welsh-Powell este 3, deși acest caz poate fi colorat și cu doar 2 culori, ceea ce demonstrează că în anumite situații ordinea de parcurge creată de sortare poate duce la un răspuns greșit în ceea ce privește numărul cromatic.

Exemplu: Cu algoritmul Greedy, graful colorat ar arăta astfel:



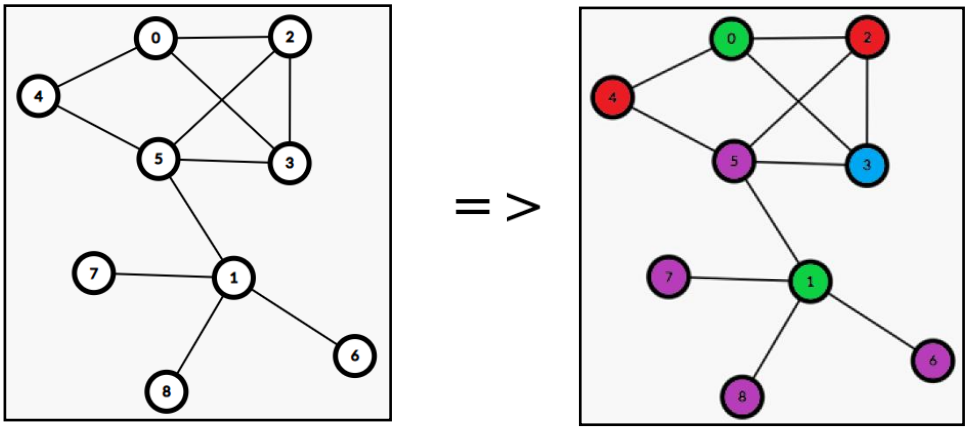
Vom analiza pe rând cum se comportă Greedy și Welsh-Powell pe un caz în care ambii algoritmii oferă o soluție greșită:

Cazul III. Greedy



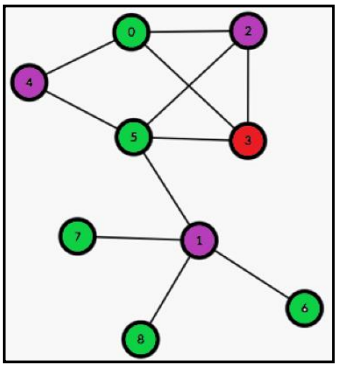
- Parcurge nodurile în ordine crescătoare începând cu nodul 0 și verifică dacă există conflicte cu vecinii deja colorați:
  - Nodul 0 nu are niciun vecin colorat, deci va primi culoarea verde(0).
  - Același lucru se întâmplă cu nodul 1 care ia tot culoarea verde(0).
  - Când ajunge în nodul 2 deoarece nodul 0 a fost colorat anterior cu verde(0), acesta va lua culoarea mov(1).
  - Nodul 3 este următorul și va primi culoarea roșu(2) deoarece nodul 0 e colorat cu verde(0) și nodul 2 cu mov(1).
  - În continuare, nodul 4 are un vecin colorat cu verde(0), deci va fi colorat cu prima culoare disponibilă adică mov(1).
  - Nodul 5 are vecini cu verde(0), mov(1) și roșu(2), deci îi atribuie o nouă culoare, albastru(3).
  - În final deoarece nodurile 6, 7 și 8 au un vecin culorat cu verde(nodul 1), aceste vor fi colorate cu mov(1).
- Prin urmare, numărul cromatic obținut de Greedy este 4.

Cazul III. Welsh-Powell.



- La început, sortează nodurile în ordine descrescătoare după numărul de vecini:
  - 4 vecini – nodurile 1 și 5
  - 3 vecini – nodurile 0, 2, 3
  - 2 vecini – nodul 4
  - 1 vecin – nodurile 6, 7, 8
- Apoi, parcurge nodurile în această ordine începând cu nodul 1 și verifică dacă există conflicte cu vecinii deja colorați:
  - Nodul 0 nu are niciun vecin colorat, deci va primi culoarea verde(0).
  - Apoi, nodul 5 îl are vecin pe nodul 1 deci primește a doua culoare, mov(1).
  - Când algoritmul ajunge în nodul 0 deoarece nu are vecini colorați, acesta ia culoarea verde(0).
  - Nodul 2 este următorul și va primi culoarea roșu(2) deoarece nodul 0 e colorat cu verde(0) și nodul 5 cu mov(1).
  - În continuare, nodul 3 are 3 vecini colorați:modul 0 cu verde(0), nodul 2 cu roșu(2) și nodul 5 cu mov(1), deci va fi colorat cu prima culoare disponibilă adică albastru(3).
  - Nodul 4 are 2 vecini: nodul 0 colorat cu verde(0) și nodul 5 cu mov(1), așadar va fi colorat cu roșu(2).
  - În final, nodurile 6, 7 și 8 au un singur vecin colorat cu verde(0), deci vor fi colorate cu mov(1).
- Așadar, număr cromatic obținut de Welsh-Powell este tot 4.

Cu toate că ambii algoritmi obțin numărul cromatic 4, soluția brute-force e mereu corectă și obține numărul cromatic egal cu 3:



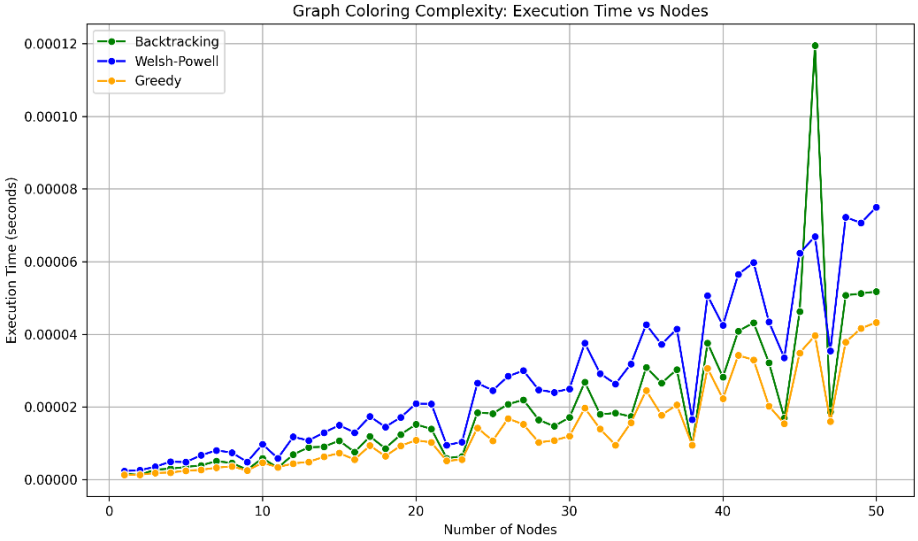
4.2 Specificatiile sistemului de calcul

- Procesor: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz 3.19 GHz
- RAM: 16 GB DDR3 @ 1600MHz
- Storage: 2TB ADATA SP900
- OS: Windows 10 Pro, version 22H2

4.3 Ilustrarea folosind grafice a rezultatelor algoritmilors

În următoarea secțiune vom prezenta grafic cum se comportă acești algoritmi pe diferite tipuri de grafuri.

Grafuri bipartite:

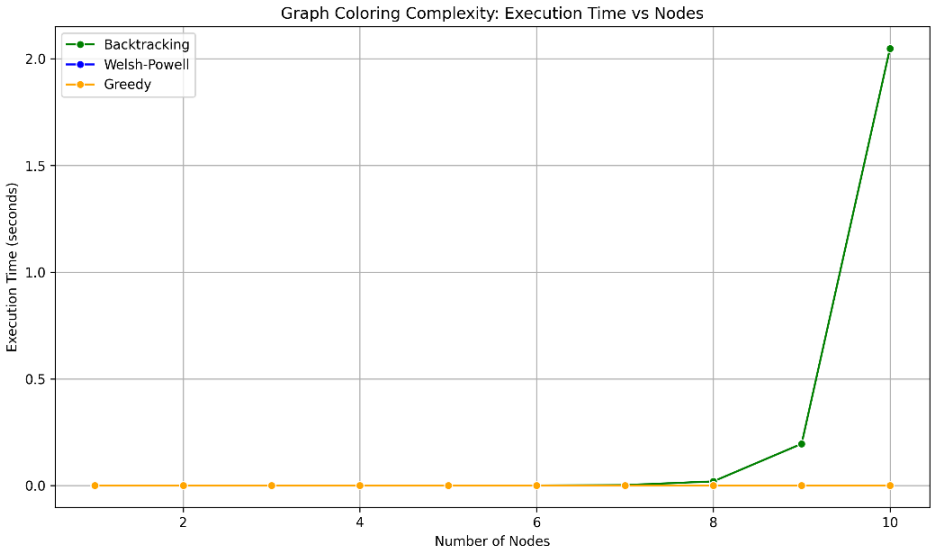


Se observă faptul că cei trei algoritmi produc rezultate similare. Șocant este faptul că algoritmul BKT produce rezultate mai bune decât euristica cu grad. Așa cum am specificat în secțiunea de analiză a complexităților, BKT poate fi eficient pe anumite intrări (grafuri cu număr cromatic mic). Știm că un graf bipartit are numărul cromatic doi, algoritmul BKT va reuși astfel să coloreze corect un nod după cel mult două încercări, făcându-l astfel foarte eficient. Efortul suplimentar de a sorta nodurile conduce în această situația la o performanță proastă a algoritmului Welsh-Powell (greedy cu grad).

Greedy este în această situație cel mai eficient deoarece reușește să coloreze corect un nod trecând o singură dată prin vecinii lui.

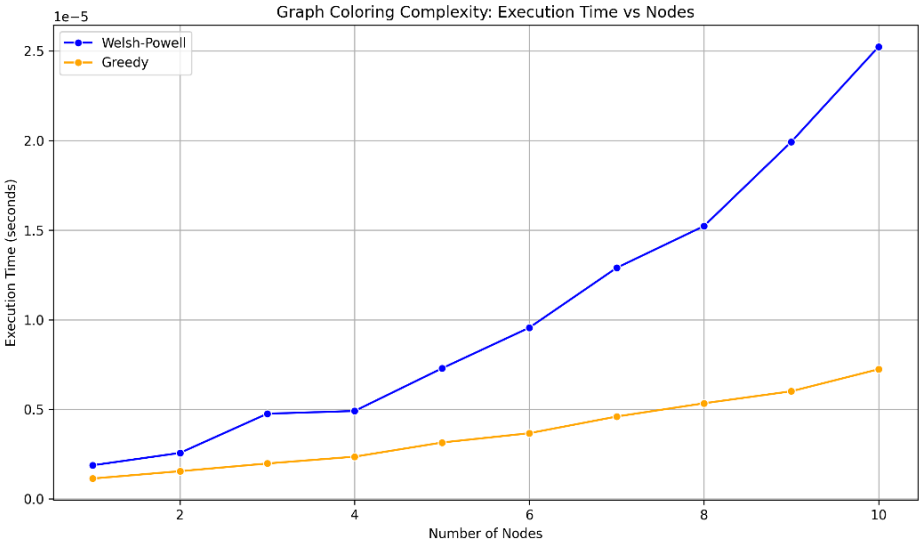
Nu există situații în care Greedy să eșueze în găsirea soluției optime pe un graf bipartit, fiind în această situație cel mai potrivit algoritm.

Grafuri complete

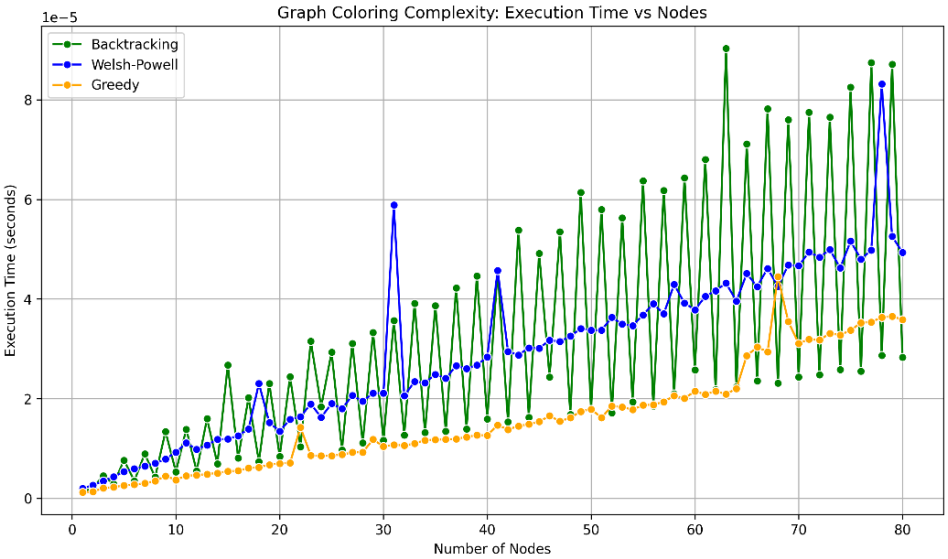


În cazul acestor grafuri se vede clar complexitatea exponențială a backtrackingului. De la 10 noduri soluția este deja generată într-un timp de 10^5 ori mai lung decât oricare dintre euristici.

În graficul de mai jos analizăm performanța dintre Welsh-Powell și Greedy pentru aceste tipuri de grafuri. Se observă că ambele grafice pot fi reprezentate asimptotic printr-o funcție pătratică. Greedy-ul pare aproape liniar datorită numărului foarte mic de operații executate pentru colorarea unui nod în raport cu Welsh-Powell.



Grafuri ciclice

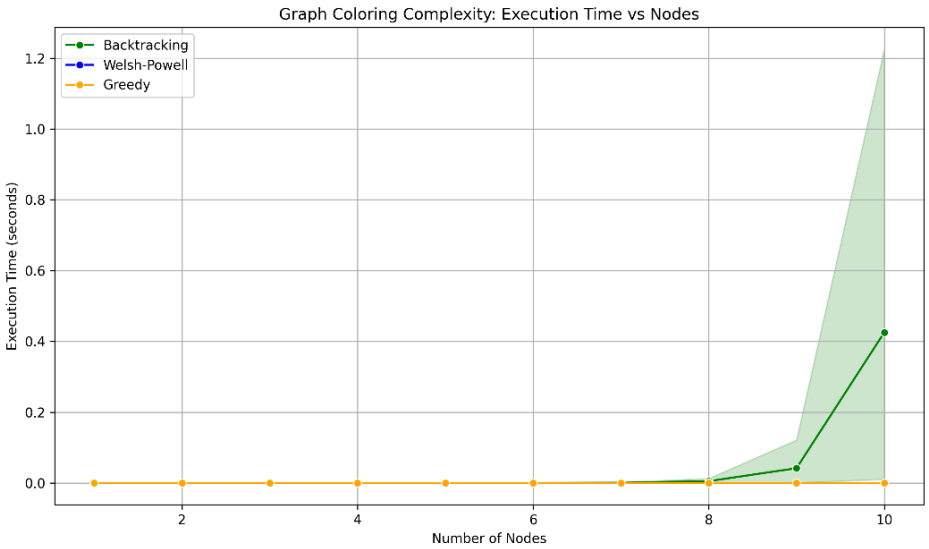


În primul rând graficul complexității pentru BKT pare să depinde de paritatea numărului de noduri din ciclu diferentiandu-se două cazuri:

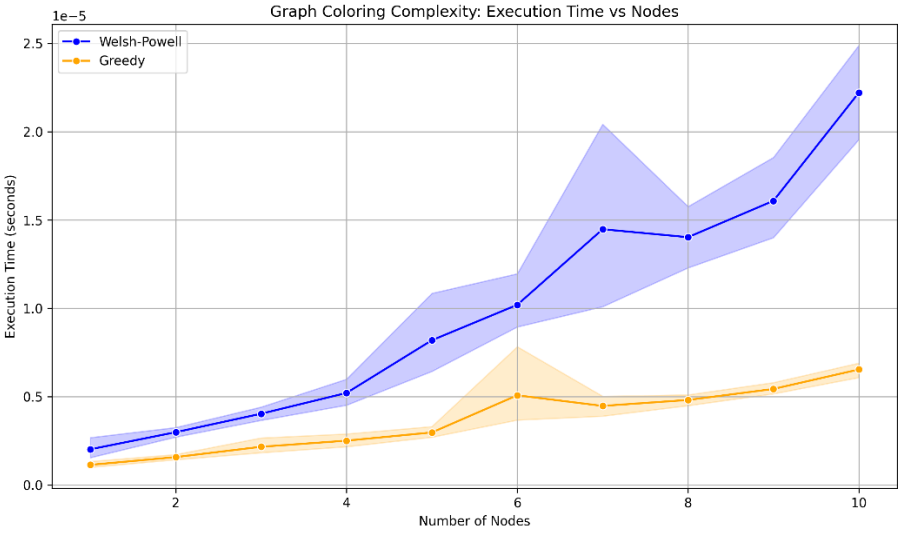
- Ciclu cu număr par de noduri: numărul cromatic va fi 2 și astfel apelurile de BKT vor avea o complexitate de calcul redusă.
- Ciclul cu număr impar de noduri: un astfel de ciclu va avea nevoie de 3 culori pentru a fi colorat. Deși algoritmul este în continuare destul de eficient, un apel de BKT suplimentar produce spikeurile din figura de mai sus.

Euristicile au pe aceste cazuri un comportament liniar de cele mai multe ori graful fiind unul destul de rar în testele alese (un simplu cerc). Spikeurile observate la acești algoritmi nu au legătură cu performanță acestora, ci cu modul de rulare (este posibil că sistemul să fi fost încărcat cu altceva în acele momente).

Grafuri dense



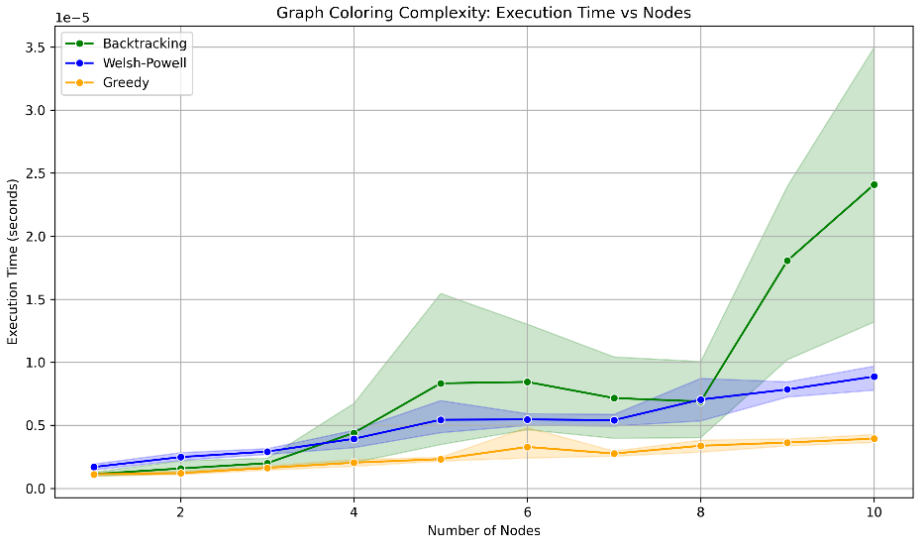
Similar ca la grafurile complete, se observă creșterea bruscă a complexității brute-forceului. Prin comparație cu acesta, putem spune că euristicile se execută instant. Mai jos este făcută comparația între euristici.



Din nou din cauza efortului de calcul suplimentar algoritmul Welsh-Powell este mai lent decât un Greedy obișnuit, totuși putem observa din rezultatele testelor că Welsh-Powell reușește să producă rezultatul optim mai bine decât Greedy. Din cele 50 de teste rulate, Welsh-Powell a avut o acuratețe de 100% spre deosebire de Greedy 94%.

Astfel, observăm că pentru grafuri dense, performanța Welsh-Powell este foarte apropiată de Greedy (ambele au complexități patratice), dar diferența de acuratețe face ca Welsh-Powell să fie mai potrivit în cazurile în care vrem o soluție optimă.

Grafuri rare

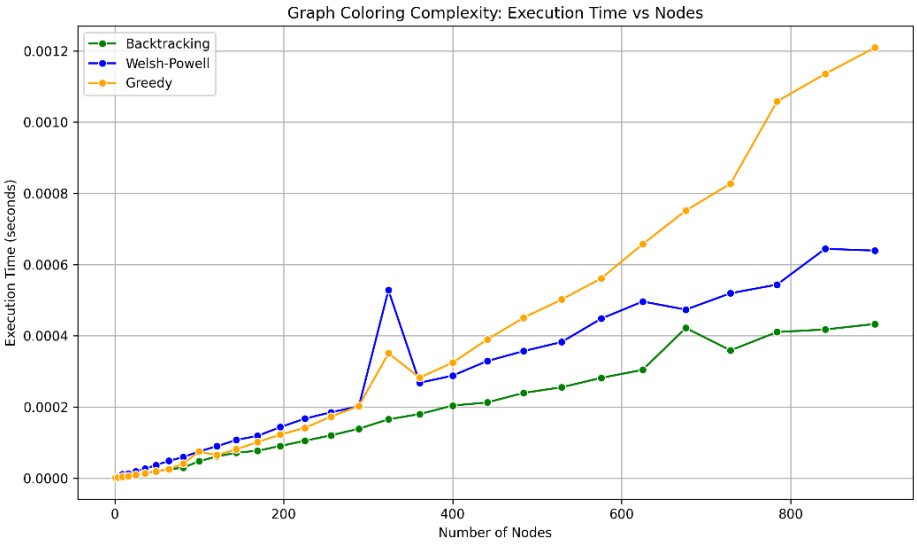


Graful fiind rar, cei trei algoritmi produc timpi de execuție similari. Euristicile produc o soluție în timp liniar, iar backtracking reușește într-un timp destul de mic să producă soluția optimă. La fel ca mai devreme, Welsh-Powell reușește să obțină același rezultat ca BKT în majoritatea cazurilor (98%), față de Greedy (92%).

Datorită proprietăților acestor grafuri, folosirea unui algoritm de tip backtracking nu este neapărat o soluție proastă. Din acest punct de vedere putem considera algoritmul Greedy cel mai slab dintre cei trei, performanța în timp nefiind suficient de mare pentru a compensa lipsei de acuratețe.

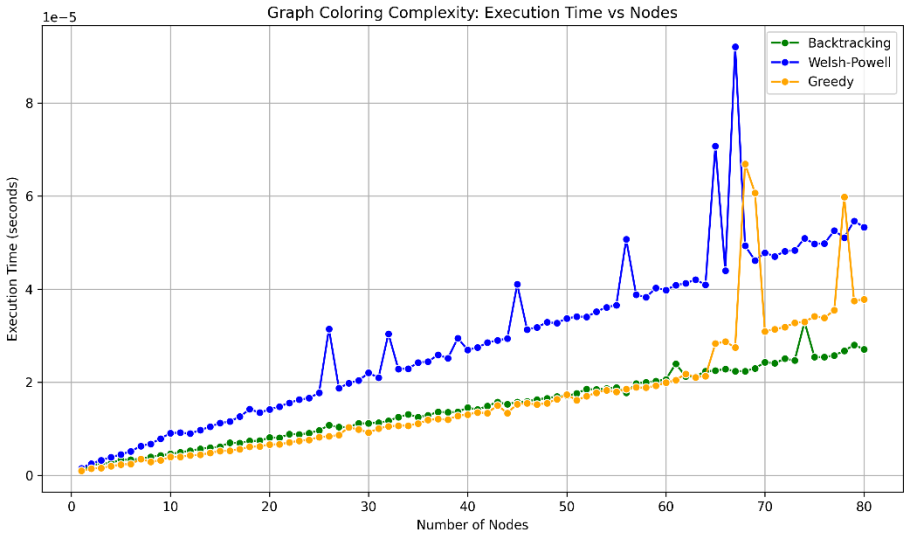
Grafuri grid

Pentru aceste grafuri numărul de muchii fiind < 4*V le putem considera grafuri rare, explicându-se astfel timpii mici de execuție. O proprietate interesantă este că pare că backtracking-ul produce răspunsul optim în cel mai scurt timp. Acest lucru poate fi explicat prin faptul că ceilalți algoritmi for face verificări suplimentare, backtracking-ul reușește să coloreze corect un nod după cel mult două încercări, acest tip de grafuri având numărul cromatic 2.



Astfel, pentru grafuri ce au la baza un grid și un număr mic de noduri suplimentar, BKT poate să aibă performanțe mult mai bune decât cele două euristici.

Grafuri lanț



Această situație este similară cu cea de la cicluri. Lipsa ultimei legături reușește să liniarizeze timpul de execuție pentru backtracking. Toți algoritmii for ajunge la soluția optimă într-un timp liniar, dar similar ca mai sus, backtracking-ul va necesita mai puține operații pentru asta, sortarea și verificarea culorilor vecinilor fiind operații redundante în acest caz de cele mai multe ori.

5 Concluzii

În concluzie, în urma analizei noastre, am constatat că problema colorării grafurilor, o provocare fundamentală în teoria grafurilor și clasificată drept NP-hard, nu permite existența unui algoritm care să garanteze soluții corecte într-un timp rezonabil pentru toate cazurile. Aceasta explică de ce, în practică, se apelează la algoritmi euristici, precum Greedy și Welsh-Powell, pentru a obține soluții aproximative într-un timp mai scurt. Totuși, studiul nostru a evidențiat că aceste metode pot eșua în moduri diferite, evidențiind limitele lor. Acest lucru arată că, deși utili în practică, acești algoritmi trebuie utilizați cu atenție, iar rezultatele lor trebuie interpretate în funcție de structura și complexitatea specifică a grafului.

În urma analizei algoritmilor specificați în acest raport exemplul legat de alocarea frecvențele radio considerăm că poate fi rezolvat prin rularea lui Welsh-Powell. Această decizie se datorează timpului scurt de rulare și returnării unei soluții apropiate de cea optimă. Chiar dacă algoritmul nu generează mereu numărul minim de frecvențe, acest lucru nu este o problemă majoră, în viața cotidiană toată lumea lovindu-se de această situație, interferența posturilor. Cu cât numărul de culori este mai mare, cu atât posturile de radio vor avea alocate mai multe frecvențe diferite, facilitând posibilele interferența.

Fiecare algoritm are particularitățile lui, important este să cunoaștem setul de date pe care rulăm pentru a obține rezultatele dorite.

6 Bibliografie

1. Jensen, Tommy R., and Bjarne Toft. *Graph coloring problems*. John Wiley & Sons, 2011.
2. Casselgren, Carl Johan. *On some graph coloring problems*. Diss. Umeå universitet, Institutionen för matematik och matematisk statistik, 2011.
3. https://www.geeksforgeeks.org/graph-coloring-applications/
4. https://www.geeksforgeeks.org/graph-coloring-set-2-greedy-algorithm/
5. https://www.tutorialspoint.com/welsh-powell-graph-colouring-algorithm
6. https://www.geeksforgeeks.org/welsh-powell-graph-colouring-algorithm/
7. Tannenbaum, A.S., 2014. NP-hard problems.
8. Formanowicz, P., & Tanaś, K. (2012). A survey of graph coloring-its types, methods and applications. *Foundations of Computing and Decision Sciences*, *37*(3), 223-238.
9. https://www.geeksforgeeks.org/3-coloring-is-np-complete/
10. Kubale, M. (2004). *Graph colorings* (Vol. 352). American Mathematical Soc..