Vertex Cover Challenge

Stanescu Matei-Octavian
Nitu Eriko-Laurentiu
Burca Paul
Grupa 322CA

Universitatea Politehnica Bucuresti,
Facultatea de Automatica si Calculatoare

1 Introducere

Problema *Vertex Cover* este o problemă clasică din teoria grafurilor și optimizarea combinatorială. Fie un graf neorientat $G(V, E)$, se cere determinarea unui *set minim de noduri* $C \subseteq V$ astfel încât pentru fiecare muchie $(u, v) \in E$, cel puțin unul dintre capete, u sau v, să aparțină lui C. Cardinalul minim al unui astfel de set se numește *numărul de acoperire în noduri* (minimum vertex cover) al grafului.

Problema apare natural în numeroase contexte practice, precum securizarea rețelelor (plasarea minimă de noduri de control pentru monitorizarea tuturor legăturilor), alocarea resurselor, analiza circuitelor, probleme de scheduling și dependențe sau bioinformatică, unde grafurile sunt folosite pentru a modela interacțiuni complexe.

Din punct de vedere al complexității computaționale, problema *Minimum Vertex Cover* este *NP-hard* în cazul general. Mai mult, versiunea decizională a problemei este *NP-completă*, fapt ce implică inexistența, cel puțin la momentul actual, a unui algoritm polinomial care să garanteze soluția optimă pentru toate instanțele problemei.

Pentru grafuri de dimensiuni reduse, este posibilă utilizarea unor metode exacte, precum backtracking-ul sau explorarea completă a spațiului de soluții, pentru obținerea rezultatului corect într-un timp rezonabil. Aceste soluții exacte sunt însă rapid impracticabile pe măsură ce dimensiunea grafului crește.

În aplicațiile reale, se recurge frecvent la **algoritmi aproximați sau euristici**, care sacrifică optimalitatea în favoarea unui timp de execuție redus și a unui consum rezonabil de resurse. Studierea acestor compromisuri între corectitudine, performanță și simplitate de implementare reprezintă un aspect central al analizei algoritmilor pentru această problemă.

1.1 Corectitudine

O soluție pentru problema *Vertex Cover* este considerată **validă** dacă pentru fiecare muchie $(u, v) \in E$, cel puțin unul dintre nodurile u sau v aparține mulțimii de noduri selectate C. Această condiție poate fi verificată în timp liniar față de numărul de muchii, parcurgând fiecare muchie și testând dacă este acoperită de cel puțin un nod din soluție.

Atât soluțiile exacte, cât și cele euristice trebuie să respecte această proprietate de corectitudine structurală. Diferența dintre ele constă nu în validitatea soluției, ci în *dimensiunea* mulțimii obținute. O soluție este **optimă** dacă numărul de noduri selectate este minim posibil.

Determinarea unei soluții optime este însă computațional dificilă, fiind o problemă NP-hard. Din acest motiv, algoritmii euristici nu garantează întotdeauna obținerea rezultatului optim. Pentru evaluarea acestora, se compară dimensiunea soluției produse cu dimensiunea soluției optime, obținută printr-un algoritm exact.

O metrică naturală pentru această comparație este **raportul de aproximare**, definit ca raportul dintre dimensiunea soluției euristice și dimensiunea soluției optime. În plus, se poate analiza frecvența cu care o euristică reușește să producă soluția optimă pe un set de teste, precum și deviația medie față de optim.

Astfel, corectitudinea este o condiție necesară pentru orice soluție, însă criteriul principal de diferențiere între algoritmi rămâne apropierea față de soluția optimă, în raport cu resursele de timp și spațiu consumate.

1.2 Eficiența

Din punctul de vedere al eficienței, există diferențe semnificative între soluțiile exacte și cele euristice pentru problema *Vertex Cover*. Algoritmii exacți au o complexitate exponențială în numărul de noduri și sunt fezabili doar pentru grafuri mici, însă oferă soluția optimă.

În schimb, euristicile și algoritmii aproximați au complexități polinomiale și pot fi rulați rapid pe orice instanță a problemei, cu prețul pierderii garanției de optimalitate. Evaluarea eficienței se realizează prin raportarea timpului de execuție la dimensiunea grafului și prin compararea performanței relative a euristicilor.

2 Soluții abordate

2.1 Soluție exactă: brute-force

Spre deosebire de algoritmii euristici, care pot produce soluții sub-optimale, metoda *brute-force* garantează întotdeauna obținerea soluției optime pentru problema *Vertex Cover*. Această abordare explorează exhaustiv toate submulțimile posibile de noduri și selectează soluția validă cu cardinal minim.

Pentru un graf cu n noduri, fiecare submulțime de noduri poate fi reprezentată printr-o mască de biți de lungime n. O mască este considerată validă dacă pentru fiecare muchie $(u, v) \in E$, cel puțin unul dintre nodurile u sau v este selectat. Verificarea acestei condiții se realizează în timp liniar față de numărul de muchii.

Algoritmul parcurge toate cele 2^n măști posibile și calculează dimensiunea fiecărei soluții valide, păstrând-o pe cea cu numărul minim de noduri. Deși corectitudinea este

garantată, complexitatea de timp a acestei metode este exponențială, de ordinul $O(2^n \cdot |E|)$, ceea ce o face impracticabilă pentru grafuri mari.

În contextul restricției de maxim 20 de noduri, această soluție este însă fezabilă și joacă un rol esențial ca reper de comparație pentru evaluarea calității soluțiilor euristice.

2.2 Euristică greedy bazată pe grad

Prima euristică utilizată pentru problema *Vertex Cover* este o metodă greedy, care construiește incremental o soluție validă, fără a garanta însă optimalitatea. Ideea de bază este de a selecta noduri astfel încât muchiile să fie acoperite cât mai rapid, folosind informații locale despre structura grafului.

Algoritmul pornește prin alegerea unui nod inițial cu grad maxim, pe baza observației că nodurile cu grad ridicat sunt incidente la mai multe muchii și pot contribui semnificativ la acoperirea acestora. Din acest nod se realizează o parcurgere de tip BFS a grafului. Pentru fiecare nod procesat, se examinează muchiile incidente care nu au fost încă acoperite. Dacă niciunul dintre capetele unei muchii nu este selectat, algoritmul alege greedy unul dintre ele pentru a fi adăugat în soluție.

După terminarea parcurgerii BFS, este posibil ca anumite muchii să rămână neacoperite, în special în cazul grafurilor neconexe. Pentru aceste muchii, algoritmul aplică o regulă suplimentară: dintre cele două capete ale muchiei, este ales nodul cu grad mai mare, presupunând că acesta va acoperi un număr mai mare de muchii rămase.

Complexitatea de timp a acestei euristici este polinomială. Parcurgerea BFS are complexitate $O(|V|+|E|)$, iar verificarea și acoperirea muchiilor este realizată în timp $O(|E|)$, rezultând o complexitate totală de ordinul $O(|V|+|E|)$. Din punct de vedere al spațiului, algoritmul utilizează structuri auxiliare liniare în numărul de noduri și muchii.

Deși această metodă produce întotdeauna o soluție validă pentru problema *Vertex Cover*, nu există garanții privind apropierea de soluția optimă. Calitatea rezultatului depinde de structura grafului și de nodul de start ales, însă în practică euristica oferă soluții rezonabile într-un timp de execuție foarte redus.

2.3 Euristică greedy cu grad maxim și departajare după vecini

A doua euristică urmărește o strategie greedy clasică pentru *Vertex Cover*: se selectează repetat un nod „important" din graf și se elimină toate muchiile incidente acestuia, până când nu mai rămân muchii neacoperite. Intuiția este că alegerea unui nod cu grad mare acoperă dintr-o singură selecție cât mai multe muchii.

Algoritmul menține listele de adiacență într-o structură de tip `set`, pentru a permite ștergerea eficientă a muchiilor pe măsură ce noduri sunt introduse în acoperire. La fiecare iterație, se caută nodul `bestNode` cu grad maxim (numărul de vecini rămași). În cazul în care există mai multe noduri cu același grad maxim, se aplică o regulă de departajare: se preferă nodul care are un vecin cu gradul minim cât mai mic. Această departajare tinde să selecteze noduri care "protejează" vecinii slabi (cu puține opțiuni), reducând riscul de a ajunge ulterior în situații unde rămân muchii care pot fi acoperite doar prin alegeri costisitoare.

După alegerea nodului `bestNode`, toate muchiile incidente lui sunt considerate acoperite: nodul este eliminat din mulțimile de vecini ale tuturor vecinilor săi, iar lista sa de adiacență este golită. Contorul soluției crește cu 1. Procesul se repetă până când numărul de muchii rămase ajunge la 0, moment în care acoperirea este completă.

Complexitatea acestei euristici este polinomială, însă mai mare decât a primei euristici. La fiecare pas se scanează toate nodurile pentru a determina gradul maxim, iar pentru fiecare nod se calculează și minimul gradelor vecinilor. În cel mai defavorabil caz, acest lucru conduce la un timp aproximativ de ordinul $O(|V| \cdot |E|)$, la care se adaugă costurile de ștergere în `set` (logaritmice). În practică, pentru dimensiuni mici (maxim 20 de noduri), acest cost rămâne rezonabil, iar calitatea soluției este în general mai bună decât a unei alegeri greedy simple.

Ca orice euristică, metoda nu garantează optimalitatea, dar produce întotdeauna o soluție validă, deoarece la final nu rămâne nicio muchie neacoperită. Calitatea depinde de structura grafului și de regulile de departajare folosite la egalitate de grad.

3 Reducere K-Clique $\leftrightarrow$ Vertex Cover

Dacă avem un graf $G(V, E)$, găsirea unui K-Clique poate fi redusă la găsirea unui Vertex Cover de mărime $|V| - K$ în graful său complementar $\overline{G}$.

Theorem 1. *Există o funcție polinomială f astfel încât:*

$$f : (G, K) \to (\overline{G}, |V| - K)$$

unde G are un K-Clique dacă și numai dacă $\overline{G}$ are un Vertex Cover de mărime $|V| - K$.

3.1 Direcția 1: Clique $\to$ Vertex Cover

Claim 1. *Dacă $V' \subseteq V$ formează un K-clique în G, atunci $V \setminus V'$ formează un vertex cover de mărime $|V| - K$ în $\overline{G}$.*

Demonstraţie. Fie V' un K-clique în G.

- Prin definiția clique: $\forall u, v \in V'$, $(u, v) \in E$ (toate perechile sunt conectate în G)
- În complementul $\overline{G}$: $\forall u, v \in V'$, $(u, v) \notin E(\overline{G})$ (nu există muchii între nodurile clique-ului)
- Considerăm o muchie arbitrară $(x, y) \in E(\overline{G})$:
  - Deoarece $(x, y) \in E(\overline{G})$, rezultă că $(x, y) \notin E$
  - Deoarece V' este un clique în G, cel puțin unul dintre $\{x, y\}$ trebuie să fie în afara lui V'
  - Prin urmare, cel puțin unul dintre $\{x, y\}$ aparține lui $V \setminus V'$
- Deci muchiile din $\overline{G}$ au cel puțin un capăt printre nodurile $V \setminus V'$
- Astfel, $V \setminus V'$ formează un vertex cover în $\overline{G}$

□

3.2 Direcția 2: Vertex Cover $\to$ Clique

Claim 2. *Dacă S formează un vertex cover de mărime $|V| - K$ în $\overline{G}$, atunci $V \setminus S$ formează un K-clique în G.*

Demonstraţie. Fie S un vertex cover de mărime $|V| - K$ în $\overline{G}$.

- Fie $V' = V \setminus S$, deci $|V'| = K$
- Presupunem prin reducere la absurd că $\exists u, v \in V'$ astfel încât $(u, v) \notin E$
- Atunci $(u, v) \in E(\overline{G})$ (prin definiția grafului complement)
- Dar $u \notin S$ și $v \notin S$ (ambele sunt în $V \setminus S$)
- Aceasta înseamnă că muchia (u, v) din $\overline{G}$ nu este acoperită de $S \to$ **contradicție!**
- Prin urmare, $\forall u, v \in V'$, $(u, v) \in E$
- Astfel, nodurile rămase $V' = V \setminus S$ reprezintă un K-Clique în G, căci nu există nicio muchie între ele în $\overline{G}$

□

3.3 Complexitate Polinomială

Reducerea este polinomială deoarece:

1. Calcularea lui $\overline{G}$ din G: $O(|V|^2)$ timp (verificăm fiecare pereche posibilă de noduri)
2. Transformarea parametrului: $K \to |V| - K$ se face în $O(1)$

Theorem 2 (Concluzie)**.** *Problema K-Clique este NP-Completă, fiind dată reducerea polinomială de la problema Vertex Cover (care este NP-Completă).*

Observație cheie: Nodurile din clique în G nu au muchii între ele în $\overline{G}$, prin urmare nodurile rămase trebuie să acopere toate muchiile din $\overline{G}$.

4 Metodologia de Evaluare

Pentru a evalua calitatea soluțiilor euristice, am dezvoltat o infrastructură completă de testare care generează grafuri diverse, calculează soluțiile optime prin metoda exactă (brute-force) și compară rezultatele euristicilor cu acestea.

4.1 Generarea Testelor

Testele sunt organizate pe **clase**, fiecare clasă reprezentând un tip specific de graf cu proprietăți distincte. Am implementat trei generatoare de grafuri:

1. **Grafuri aleatoare** (`test-gen-random`): Generează grafuri cu $N = 20$ noduri, unde fiecare pereche de noduri are o probabilitate dată (densitatea) de a forma o muchie. Densitățile testate sunt 10%, 30% și 67%, acoperind grafuri sparse, medii și dense.

2. **Cicluri cu coarde** (`test-gen-cycle-with-cords`): Generează un ciclu hamiltonian de 20 noduri la care se adaugă câteva coarde aleatorii. Această structură testează comportamentul euristicilor pe grafuri cu structură regulată.

3. **Grafuri bipartite** (`test-gen-bipartite`): Generează grafuri cu două mulțimi disjuncte de noduri, unde muchiile există doar între cele două mulțimi. Grafurile bipartite au proprietăți speciale pentru vertex cover (teorema lui König).

4.2 Pipeline-ul de Evaluare

Evaluarea se realizează automat prin comanda `make pipeline`, care execută următorii pași:

1. **Generare teste**: Se generează câte 10 instanțe pentru fiecare clasă de grafuri (random-10%, random-30%, random-67%, cycle-with-cords, bipartite).

2. **Calculare referințe**: Pentru fiecare test, algoritmul brute-force calculează soluția optimă și o salvează în directorul `ref/`.

3. **Rulare euristici**: Fiecare algoritm euristic este rulat pe toate testele, iar rezultatele sunt salvate în `out/<algoritm>/`.

4. **Calculare acuratețe**: Se compară rezultatele euristicilor cu referințele optime.

4.3 Metrici de Evaluare

Pentru fiecare euristică și clasă de teste, se calculează:

- **Perfect matches**: Numărul de teste pentru care euristica a găsit soluția optimă.
- **Acuratețe medie**: Media raportului $\frac{\text{optim}}{\text{euristic}} \times 100\%$ pe toate testele. O valoare de 100% indică soluție optimă, valori mai mici indică soluții sub-optimale.
- **Excess**: Suma diferențelor (euristic$-$optim) pe toate testele din clasă, reprezentând numărul total de noduri "în plus" selectate față de optim.

Raportarea se face atât per clasă (pentru a identifica tipurile de grafuri pe care euristica performează mai slab), cât și agregat pe toate testele.

4.4 Rezultatele Testelor

Am rulat ambele euristici pe un set de 500 de teste (100 per clasă). Rezultatele sunt prezentate în tabelele următoare:

Tabela 1: Rezultate: Greedy cu BFS (greedy-highest-order)

Clasă	**Teste**	**Perfect (%)**	**Avg (%)**	**Excess**
Random 10%	100	48 (48.0%)	92.19%	+73
Random 30%	100	27 (27.0%)	91.19%	+126
Random 67%	100	19 (19.0%)	93.99%	+103
Bipartite	100	90 (90.0%)	98.18%	+20
Cycle-with-cords	100	48 (48.0%)	94.89%	+58
TOTAL	500	232 (46.4%)	94.09%	+380

Tabela 2: Rezultate: Greedy cu grad maxim și departajare (greedy-remove-edges)

Clasă	**Teste**	**Perfect (%)**	**Avg (%)**	**Excess**
Random 10%	100	95 (95.0%)	99.43%	+5
Random 30%	100	87 (87.0%)	99.03%	+13
Random 67%	100	83 (83.0%)	99.00%	+17
Bipartite	100	78 (78.0%)	97.51%	+27
Cycle-with-cords	100	79 (79.0%)	98.09%	+21
TOTAL	500	422 (84.4%)	98.61%	+83

Tabela 3: Comparație timp de execuție (500 teste, N=20)

Algoritm	**Timp total**	**Perfect (%)**	**Avg (%)**
Brute-Force	10.02s	100.0%	100.00%
Greedy-highest-order	3.66s	46.4%	94.09%
Greedy-remove-edges	3.52s	84.4%	98.61%

4.5 Analiza Rezultatelor

Euristica **greedy-remove-edges** (cu departajare după gradul minim al vecinilor) este semnificativ mai performantă:

- **Perfect matches**: 84.4% vs 46.4% — aproape dublu
- **Acurateţe medie**: 98.61% vs 94.09%
- **Excess total**: +83 vs +380 — de peste 4 ori mai puţine noduri în plus
- **Timp similar**: ambele euristici rulează în 3.5s, de 3x mai rapid decât brute-force

Observaţii pe clase:

- Pe grafuri *sparse* (densitate 10%), greedy-remove-edges atinge 95% perfect matches, în timp ce greedy-highest-order doar 48%.

- Pe grafuri *bipartite*, ambele euristici performează bine (78-90% perfect).
- Greedy-remove-edges are performanță consistentă pe toate clasele (78-95% perfect), în timp ce greedy-highest-order variază mult (19-90%).

Observații pe timp de execuție:

- Brute-force (complexitate $O(2^n \cdot |E|)$) este de 3x mai lent, dar garantează soluția optimă.
- Euristicile au timp similar deoarece ambele au complexitate polinomială ($O(|V| \cdot |E|)$).
- Pentru $N = 20$, diferența nu este dramatică, dar pentru grafuri mari brute-force devine impracticabil.

Concluzia este că regula de departajare (alegerea nodului cu vecin de grad minim) îmbunătățește semnificativ calitatea soluției, reducând riscul de a "bloca" noduri care ulterior devin greu de acoperit.

4.6 Analiza Performanței în funcție de N

Pentru a înțelege mai bine comportamentul algoritmilor, am măsurat numărul de operații pe secundă (ops/s) în funcție de numărul de noduri N. Graficele următoare prezintă rezultatele obținute pe grafuri aleatorii cu densitate 30%.

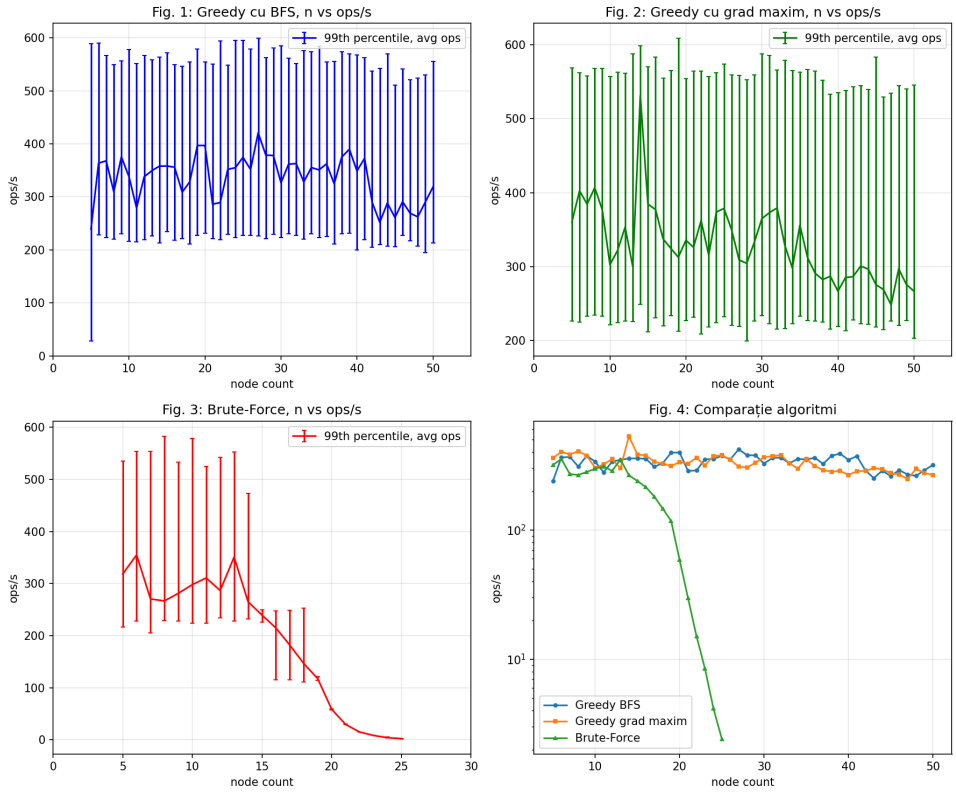

Figura 1: Comparație performanță: ops/s în funcție de numărul de noduri

Observații cheie:

- **Euristicile** (greedy-highest-order și greedy-remove-edges) mențin o performanță relativ constantă (300-400 ops/s) indiferent de N, confirmând complexitatea polinomială.
- **Brute-force** prezintă degradare exponențială: de la 350 ops/s la $N = 5$ la doar 2 ops/s la $N = 25$. La $N = 25$, o singură execuție durează 400ms.
- Pentru $N > 25$, brute-force devine impracticabil (timp de execuție > 1 secundă per graf), în timp ce euristicile rămân rapide.

Aceste rezultate confirmă necesitatea utilizării euristicilor pentru grafuri mari, unde soluția exactă nu poate fi calculată într-un timp rezonabil.