k-Clique

Calin Marinescu 323CD Stefan Nemeti 323CD

Facultatea de Automatică si Calculatoare
Universitatea Politehnica din Bucuresti
https://github.com/NemetiStefan/Analiza-k-Clique

Abstract. Evaluarea eficientei algoritmilor ce rezolva problema "Clique"

1 Introducere

În numeroase contexte practice, se pune întrebarea cum putem identifica submulțimi de entități care sunt toate conexe între ele, respectiv cum putem determina dacă există un grup de obiecte (noduri) care formează o rețea completă (fiecare obiect fiind în legătură directă cu toate celelalte). În termeni de teorie a grafurilor, acest lucru se traduce prin studierea k-clique-urilor (clanurilor de mărime k) și, mai general, a max-clique-ului (clanul de mărime maximă).

O clique într-un graf neorientat G=(V,E) este un subgraf în care oricare două noduri distincte sunt conectate printr-o muchie. Astfel, o k-clique este un astfel de subgraf cu exact k noduri, iar problema max-clique constă în determinarea celei mai mari clique dintr-un graf, deci a subgrafului complet cu numărul maxim de vârfuri.

Identificarea unor astfel de structuri este esențială în studiul relațiilor dintre entități: submulțimile dense (clique-urile) reflectă, de pildă, grupuri de prieteni sau comunități cu interese comune în rețelele sociale, proteine care interacționează strâns în cadrul unei rețele biologice sau seturi de documente cu tematică foarte asemănătoare în cadrul analizei de text.

Exemplul clasic este cel al rețelelor sociale, unde o clique reprezintă un grup de persoane care se cunosc reciproc. Studiul clique-urilor ajută la înțelegerea mecanismelor de formare a comunităților, la îmbunătățirea sugestiilor de conexiuni (recomandarea „prieteni ai prietenilor") și la identificarea influencerilor.

Alte aplicații practice ale conceptului de k-clique și max-clique includ:

- **Bioinformatică:** identificarea subrețelelor dense de gene sau proteine care interacționează intens, pentru a descoperi funcții biologice comune sau căi de semnalizare în genom.

- **Optimizare și planificare:** determinarea celor mai bine conectate echipe sau resurse într-o rețea, pentru a coordona mai eficient sarcinile și a reduce costurile.

- **Analiza documentelor:** gruparea fișierelor cu conținut similar într-o rețea de co-apariție a termenilor, pentru a găsi subteme sau subiecte de interes în analiza textelor mari (de ex. articole științifice).

- **Securitate informatică:** detectarea grupurilor de atacatori care colaborează într-o rețea, pentru a identifica amenințările cibernetice emergente.

2 Demonstratie NP-Hard

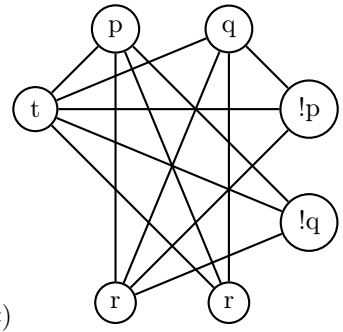
O demonstrație că problema k-Clique este NP-hard implică reducerea unei probleme cunoscute ca fiind NP-completă la k-Clique într-un timp polinomial. În acest caz, se poate utiliza problema SAT (o problemă NP-completă bine cunoscută) pentru a demonstra această reducere.

1. **Problema MAX Clique:**
   - Dat un graf neorientat G=(V,E), găsește cea mai mare mulțime de noduri care formează o clique (adică toate perechile de noduri din această submulțime sunt conectate prin muchii).
2. **Problema SAT:**
   - Dată o formulă booleană în formă normală conjunctivă (CNF) cu exact 3 literali per clauză, determină dacă formula este satisfiabilă.

2.1 Pasii Demonstratiei

1. **Alegerea problemei de reducere:**
   - Alegem problema SAT, care este cunoscută a fi NP-completă.
2. **Reducerea problemei SAT la MAX Clique:**
   - Fie o formulă booleană ϕ constând din clauzele C1, C2, C3 ... Cn, unde fiecare clauză conține exact 3 literali (de exemplu C1 = (p, !q, r))
3. **Construim un graf G=(V,E) astfel**
   - **Noduri:** Pentru fiecare literal din fiecare clauză, adăugăm un nod înG
   - **Muchii:** Adăugăm muchii între noduri dacă:
     - Cele două noduri aparțin unor clauze diferite, și
     - Literalul corespunzător nu este contradictoriu (adică nu există muchie între x si !x
4. **Legătura între satisfiabilitatea formulei și clique-ul din graf:**
   - Daca ϕ este satisfiabilă, atunci există o atribuire a valorilor booleene care face ca fiecare clauză să fie adevărată. Acest lucru corespunde selectării unui nod din fiecare clauză, formând o clique în G.
   - Dacă există o clique de dimensiune k în G, atunci aceasta corespunde unui set de literali care satisfac formula ϕ
5. **Reducerea este polinomială:**
   - Construirea nodurilor și muchiilor din graf se face în timp polinomial în raport cu numărul de clauze și literali din formula ϕ

6. **Exemplu de graf**



F:(p$\cup q$) $\cap$ (!$p\cup$!$q \cup r$) $\cap$ ($r \cup t$)

2.2 Concluzie

Am redus problema SAT la k-Clique într-un timp polinomial. Deoarece SAT este NP-completă, iar k-Clique este cel puțin la fel de dificilă ca aceasta, concluzionăm că problema k-Clique este NP-hard.

3 Prezentarea Algoritmilor

3.1 Backtracking Simplu

Descrierea modului de functionare:

- **Principiu:** Parcurge recursiv toate subseturile posibile de noduri (sau toate extensiile posibile) pentru a verifica dacă formează o clică.

- **Operații:**
  1. Începe de la un subset (inițial gol).
  2. La fiecare pas, încearcă adăugarea unui nod v dacă păstrează proprietatea de clică (adică v este adiacent cu toate nodurile deja în subset).
  3. Continuă recursiv cu următoarele noduri, actualizând „cea mai mare clică" găsită.

- **Rezultate:** Găsește soluția exactă (dimensiunea maximă și compoziția clicei).

Complexitate:

- În cel mai rău caz: $O(2^N)$. Practic, pe măsură ce N crește, numărul de subseturi crește exponențial.
- Optimizări: Se pot face „pruning" (de exemplu, dacă nu mai sunt suficiente noduri rămase pentru a depăși bestSize), însă tot exponențial rămâne în cel mai rău caz.

Avantaje

– Este exact și garantează găsirea clicei de dimensiune maximă.
– Implementare relativ simplă (dacă nu se urmăresc optimizări majore).

Dezavantaje

– Foarte lent pentru N mare (peste 20–25 de noduri devine impracticabil).
– Nu profită la maximum de structura grafului.

3.2 Bron–Kerbosch

Descrierea modului de functionare:

– **Principiu:** Algoritm clasic pentru găsirea tuturor clicilor maximale dintr-un graf neorientat.

– **Pasi:**
  1. **Pornește cu 3 seturi:** R (submulțimea actuală, care se extinde spre o clică), P (nodurile ce pot fi adăugate în R), X (nodurile excluse).
  2. Alege un pivot și, pe rând, mută nodurile candidate din P în X, după ce le extinde în R, pentru a nu repeta configurații.
  3. Când P și X devin goale, R este o clică maximală.

– **Adaptare:** Pentru găsirea clicei maxime, putem stoca cea mai mare clică maximală găsită, sau folosim pivotare pentru prune mai agresiv.

Complexitate:

– Este exponențial în cel mai rău caz, dar pe grafuri sparse poate fi mult mai rapid decât backtrackingul simplu.
– Poate genera toate clicile maximale, ceea ce costă în grafuri dense, dar în practică este adesea mai rapid decât un backtracking naiv.

Avantaje

– Exact și poate produce clicile de dimensiune maximă cu overhead mai mic decât backtrackingul simplu.
– Poate fi îmbunătățit prin pivot, ordonări inteligente etc.

Dezavantaje

– Poate genera foarte multe clici în grafuri dense.
– Tot exponențial rămâne în cel mai rău caz pentru N mare.

3.3 Hill Climbing (Local Search)

Descrierea modului de functionare:

- **Principiu:**Algoritm euristic local, pornește de la o soluție (de exemplu, random) și face îmbunătățiri pas cu pas.

- **Operații:**
  1. Există un subset care este o clică validă (inițial mic).
  2. Se încearcă adăugarea unui nod compatibil (dacă se găsește, se adaugă și se reia).
  3. Dacă nu se poate adăuga, se scoate un nod și se încearcă introducerea altora.
  4. Se oprește când nu mai există nicio mutare care să crească (sau să mențină) dimensiunea clicei.

- **:** O clică (nu neapărat maximă), dar mai bună decât un subset complet aleator.

Complexitate:

- De obicei polinomială, deoarece la fiecare iterație se fac verificări de adiacență.
- Numărul de iterații este un parametru (ex. 500, 1000). Timpul total este aproximativ O(itMax $\times$ $O(2^N)$), în funcție de implementare.

Avantaje

- Rapid pentru grafuri mari, unde metodele exacte nu mai sunt fezabile.
- Implementare relativ simplă.

Dezavantaje

- Nu garantează soluția optimă (clică maximă), se poate bloca într-un minim local.
- Depinde de soluția inițială, iar rezultatele pot varia de la o rulare la alta.

3.4 Simulated Annealing

Descrierea modului de functionare:

- **Principiu:** Inspirat din procesul de răcire (annealing). Permite, cu o anumită probabilitate care scade treptat, acceptarea mutărilor ce scad calitatea, evitând blocajele în minime locale.

- **Pasi:**

1. Se pornește cu o soluție (clică) inițială aleatoare.
2. Se generează un vecin (de exemplu, adăugarea sau scoaterea unui nod).
3. Dacă soluția se îmbunătățește, este acceptată; dacă nu, este acceptată cu o probabilitate exp(-delta / T).
4. Temperatura T scade treptat (rata alpha).
5. Se poate face și un pas final de îmbunătățire locală (post-processing).

Complexitate:

- Depinde de maxIter (numărul de iterații) și de felul în care se generează vecinii.
- Timpul total poate fi aproximat prin O(maxIter $\times$ $O(2^N)$), în funcție de implementare.

Avantaje

- Poate evita minimele locale acceptând ocazional soluții mai slabe la început.
- Parametrii (temperatura inițială, factorul de răcire) pot fi reglați pentru diferite tipuri de grafuri.

Dezavantaje

- Nu garantează soluția optimă, fiind o euristică.
- Parametrii nepotriviți pot duce la blocare rapidă sau la explorare ineficientă.

3.5 Tabu Search

Descrierea modului de functionare:

- **Principiu:** O metaeuristică ce menține o listă tabu de mutări recent folosite, pentru a evita revenirea la configurații deja explorate.

- **Pasi:**
  1. Se pornește de la o clică inițială (random).
  2. Se fac mutări (adăugare/eliminare nod) doar dacă nu sunt tabu.
  3. Mutarea făcută este plasată în lista tabu pentru un anumit număr de iterații (tabuTenure).
  4. Se reține cea mai bună soluție globală găsită.

- **Rezultat:** O soluție euristică, de obicei mai bună decât un simplu Hill Climbing, fiindcă nu se mai revine în aceleași bucle.

Complexitate:

- Parametrii: maxIter (număr de iterații), tabuTenure (câte iterații rămâne o mutare tabu).

– În fiecare iterație se caută nod de adăugat/eliminat (verificări de adiacență). Timpul total poate fi aproximat prin O(maxIter $\times$ $O(2^N)$).

Avantaje

– Evită să rămână blocat în aceleași minime locale, memorând mutările tabu.
– Poate obține rezultate bune pe instanțe mai mari, reglând parametrii.

Dezavantaje

– Nu garantează soluția optimă (este tot o euristică).
– Parametrii (tabuTenure, maxIter) influențează major calitatea și timpul de rulare.

4 Evaluare

4.1 Construirea Testelor

Testele au fost concepute astfel incat sa testeze atat gradul de corectitudine al algoritmilor euristici, cat si viteza, in medie, a algoritmilor. Astfel, am formulat 15 teste reprezentative:

– **Test 1:** Testeaza viteza in cazul cel mai favorabil, numar mic de noduri fara muchii intre ele
– **Test 2:** Testeaza viteza pe un graf complet cu 5 noduri
– **Test 3:** Testeaza viteza pe un graf complet cu 15 noduri
– **Test 4:** Testeaza corectitudinea medie: 12 noduri, 3 elemente conexe(fiecare formand un graf complet)
– **Test 5:** Testeaza viteza si corectitudinea pe un graf aleator cu 10 noduri si 12 muchii
– **Test 6:** Testeaza viteza si corectitudinea pe un graf aleator cu 12 noduri si 16 muchii
– **Test 7:** Testeaza viteza si corectitudinea pe un graf aleator cu 15 noduri si 19 muchii
– **Test 8:** Testeaza viteza si corectitudinea pe un graf cu o componenta conexa usor mai mare(18 noduri 24 muchii)
– **Test 9:** Testeaza viteza si corectitudinea pe un graf conex (20 noduri 28 muchii)
– **Test 10:** Testeaza viteza si corectitudinea pe un graf aleator cu 20 noduri si 20 muchii
– **Test 11:** Testeaza viteza si corectitudinea pe un graf cu 4 componente conexe de dimensiuni diferite
– **Test 12:** Testeaza viteza si corectitudinea pe un graf aleator cu 18 noduri si 3 componente
– **Test 13:** Testeaza viteza si corectitudinea pe un graf aleator cu 14 noduri si 2 componente

– **Test 14:** Testeaza viteza si corectitudinea pe un graf aleator
– **Test 15:** Testeaza viteza si corectitudinea pe un graf aleator

Testele evalueaza de mai multe ori cazul mediu pentru o aproximare buna a timpului de rulare, dar si cazurile cele mai favorabile, cat si cele mai nefavorabile, pentru o comapartie amanuntita a vitezelor de rulare si complexitatiolor. Fiecare test etse rulat de 500 de ori de catre fiecare algoritm, pentru a putea calcula timpul mediu de executie si pentru a putea calcula procentul de corectitudine al algoritmilor.

4.2 Specificatiile sistemului

Laptopul pe care am rulat testele are ca procesor un Intel i7-13620H si 16 GB de memorie RAM. Testele au fost rulate pe WSL2. in makefile nu au fost folosite flag-uri de optimizare.

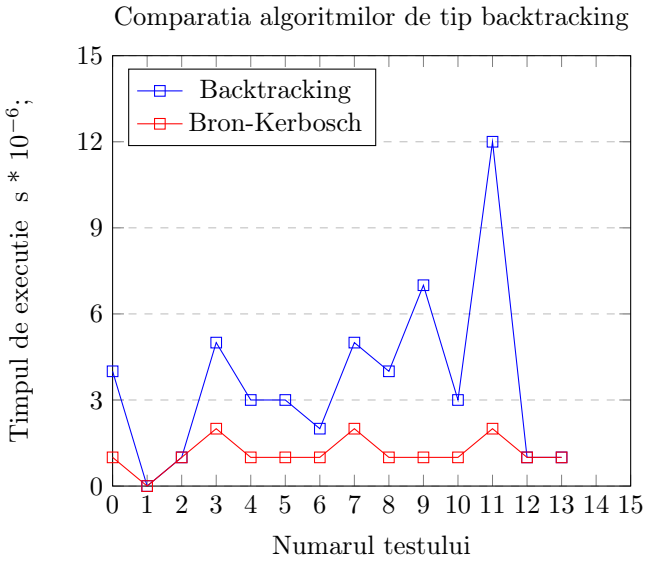
4.3 Rezultatele evaluarii

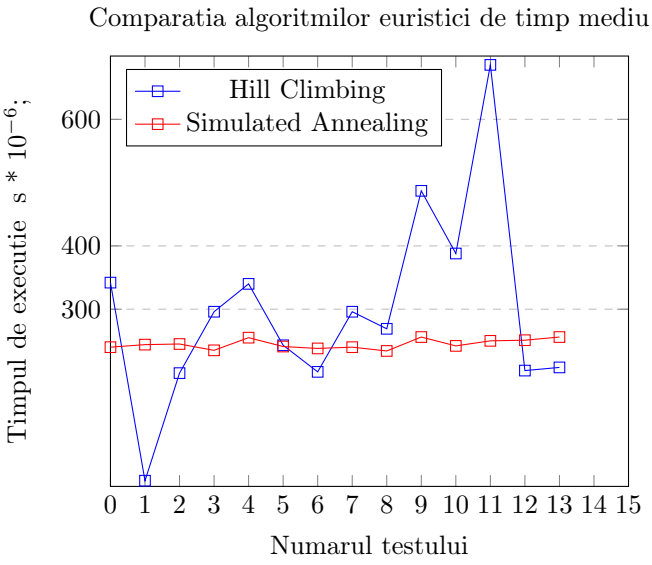
Table 1. Rezultatele testelor de gasire a k-Clique.

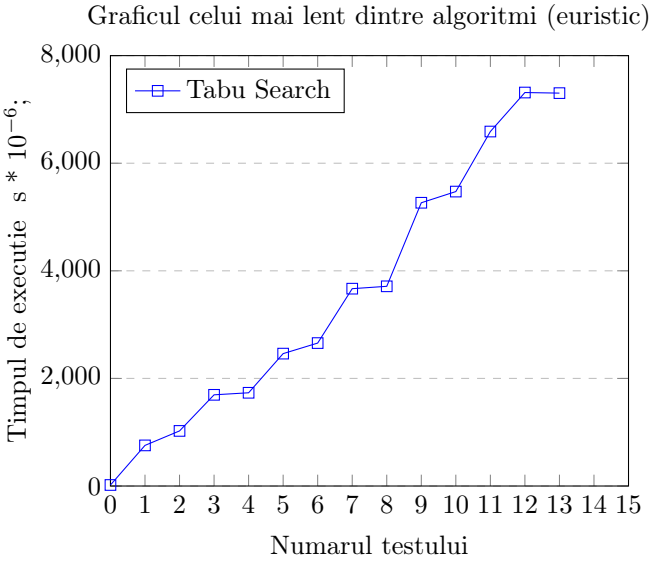
Nume	Numar de teste picate (din 7500)	% Succes	Timp mediu de executie
Backtracking(naiv)	0	100%	0.000112 s
Bron-Kerbosch	0	100%	0.000031 s
Hill Climbing	1399	82%	0.000349 s
Simulated Annealing	720	90.4%	0.000247 s
Tabu Search	1151	84.7%	0.004158 s

Asa cum era de asteptat, variantele bazate pe backtracking sunt cele mai precise.

4.4 Comparatie grafica







4.5 Interpretarea Datelor

Se poate observa din cele 3 grafice ca pe testele luate (maxim 20 de noduri) algoritmii recursivi sunt cei mai rapizi, cel mai rapid fiind Bron-Kerbosch si sunt si cei mai stabili, ambii avand o rata de succes de 100%. Pe de alta parte algoritmii euristici au timpuri mai mari de rulare de pana la 500 de ori, in cazul Tabu search. Dintre algoritmii euristici cel cu rata de succes cea mai mare este Simulated Annealing, iar cel cu cea mai mica este Hill Climbing. Simulated Annealing este si cel mai rapid pe testele rulate.

Se poate observa si faptul ca Simulated Annealing are un timp de executie relativ constant indiferent de dificultatea testelor.

In grafic testele sunt asezate in ordinea crescatoare a timpului mediu de executie (pe testul respectiv, compus din rularea celor 5 algoritmi. Acest lucru a fost facut pentru a putea observa posibila proportionalitate intre timpul de executie si dificultatea testelor, indiferent de algoritmul folosit. Astfel, putem observa cresterea si descresterea unitara a timpului de executie intre algoritmi, mai putin la Tabu Search, care se dovedeste a fi ineficient.

5 Concluzii

Dacă graful are puține noduri (ex. sub 20) și vrem neapărat soluția exactă, putem alege un algoritm exact (Backtracking sau Bron–Kerbosch). Pentru instanțe reale însă, de obicei mari și complexe, metodele exacte devin rapid impracticabile. În astfel de situații, folosirea unei euristici (Hill Climbing, Simulated Annealing, Tabu Search) este inevitabilă. Alegerea concretă depinde de cât de bună trebuie să fie soluția și de resursele disponibile.

Hill Climbing este foarte rapid, dar riscă să se blocheze în soluții slabe. Simulated Annealing și Tabu Search tind să ofere clici mai mari, însă consumă mai mult timp de rulare și necesită parametri bine reglați. Dacă graful aparține unei anumite clase (ex. foarte dispersat sau foarte dens), uneori există strategii specializate de care se poate profita. În general, însă, compromisul rămâne între rapiditatea metodelor euristice și garanția soluției oferită de metodele exacte, care, din păcate, nu mai fac față când numărul de noduri e ridicat.

6 Bibliografie

- https://en.wikipedia.org/wiki/Clique_problem $CliqueProblem$
- https://en.wikipedia.org/wiki/Bron–Kerbosch_algorithm $Bron\breve{}KerboschAlgorithm$
- https://en.wikipedia.org/wiki/Hill_climbing $HillClimbing$
- https://www.geeksforgeeks.org/what-is-tabu-search/ What is Tabu Search?
- https://www.geeksforgeeks.org/simulated-annealing/ Simulated Annealing