Capitolul 7
NUMĂRUL DE STABILITATE ŞI DENSITATEA UNUI GRAF

7.1. Mulţimi stabile şi clici

Definiţie. Numim mulţime stabilă în graful G o mulţime de vârfuri $S \subseteq X$ pentru care

$$[S]_G = [S, \emptyset],$$

adică, $x, y \in S \Rightarrow [x, y] \notin U$. Notăm cu **P** familia mulţimilor stabile în graful G. O mulţime stabilă S care este maximală în raport cu relaţia de incluziune între mulţimi se numeşte mulţime stabilă maximală, adică $S \subset S' \Rightarrow S' \notin \mathbf{P}$ $(S' \subseteq X)$.

Numărul

$$\alpha(G) = \max_{S \in \mathbf{P}} |S|$$

se numeşte numărul de stabilitate al grafului G, iar o mulţime $S \in \mathbf{P}$ cu $|S| = \alpha(G)$ va fi numită sistem de stabilitate al grafului G.

Definiţie. Se numeşte clică în graful G o mulţime de vârfuri $C \subseteq X$ care generează un subgraf complet în graful G, adică $x, y \in C \Rightarrow [x, y] \in U$.

Se numeşte densitatea grafului G, numărul

$$\omega(G) = \max_{C \in \mathbf{C}} |C|,$$

unde **C** este familia clicilor grafului G.

Observaţii.

a) Pentru orice mulţime stabilă $S \in \mathbf{P}$ şi orice clică $C \in \mathbf{C}$ are loc relaţia:

$$|S \cap C| \leq 1.$$

b) Orice mulţime stabilă în graful G este o clică în graful său complementar $\overline{G}$ şi invers. Avem evident:

$$\alpha(G) = \omega(\overline{G}) \text{ şi } \omega(G) = \alpha(\overline{G}).$$

Un graf G cu n vârfuri şi m muchii va fi notat cu G(n,m). Una dintre cele mai importante probleme privind densitatea unui graf este următoarea:

Fie n şi k < n două numere naturale. Care este cel mai mic număr $z_{k+1}(n)$ cu proprietatea:

$$m \geq z_{k+1}(n) \Rightarrow \omega(G(n,m)) \geq k+1?$$

Pentru a răspunde la această întrebare să considerăm:

$$n = t \cdot k + r,$$

unde $t = \left[\frac{n}{k}\right]$, $0 \leq r \leq k-1$ şi să construim graful T_n^k în felul următor:

Graful T_n^k are n vârfuri repartizate în clasele $S_1, S_2, ..., S_k$ cu

$$|S_i| = \begin{cases} t+1, & i = 1,2,...,r \\ t, & i = r+1,...,k, \end{cases}$$

iar două vârfuri diferite vor fi unite printr-o muchie dacă şi numai dacă aparţin la clase diferite.

Se constată uşor că numărul muchiilor grafului T_n^k este:

$$f_k(n) = \frac{k(k-1)t^2}{2} + r(k-1)t + \frac{r(r-1)}{2}$$

$$= \frac{k-1}{2k}(n^2 - r^2) + \binom{r}{2}.$$

7.2. Problema mulţimii independente

În matematică, *problema mulţimii independente* (*independent set problem* **(IS)**) este recunoscută ca o problemă de teoria grafurilor sau/şi combinatorică. Problema mulţimii independente este de tipul NP – completă.

Descriere

Fiind dat un graf *G*, o *mulţime independentă* reprezintă o submulţime a nodurilor grafului considerat, noduri neadiacente. Cu alte cuvinte, subgraful indus de aceste noduri nu are muchii, ci doar vârfuri izolate. În aceste condiţii, problema mulţimii independente cere: Fiind dat un graf *G* şi un întreg *k*, are *G* o mulţime independentă de mărime cel puţin *k*?

Problema de optimizare corespunzătoare este cunoscută sub numele de *problema mulţimii independente maxime*, care încearcă să găsească cea mai extinsă mulţime independentă dintr-un graf. Odată găsită soluţia problemei decizionale, se poate face uz de căutarea binară pentru a rezolva problema iniţială, apelarea soluţiei fiind de ordinul $O(\log|V|)$. Se cunoaşte faptul că pentru această problemă nu avem un algoritm de aproximare al factorului constant dacă $P \neq NP$.

Algoritmi

Cel mai simplu algoritm pentru mulţimile independente examinează fiecare submulţime de vârfuri de mărime cel puţin *k*, verificând dacă este independentă sau nu. Acest lucru implică un timp de ordin polinomial dacă

acest *k* egalează numărul vârfurilor, sau dacă este mai mic cu o unitate ca acesta, dar nu dacă reprezintă jumătate din numărul vârfurilor.

O problemă mult mai uşor de rezolvat este aceea a găsirii unei mulțimi independente maximale, care să nu fie conținută în nici o altă astfel de mulțime independentă. Pornim de la un singur vârf. Găsim un vârf neadiacent celui considerat inițial şi-l adăugăm mulțimii respective, apoi găsim un alt nod neadiacent niciunui vârf menționat anterior ș.a.m.d. până când nu mai găsim astfel de noduri. La acel moment mulțimea este maximal independentă. Se cunosc algoritmi mult mai complecşi, de listare a tuturor mulțimilor independente maximale, dar, în general, numărul unor astfel de mulțimi poate fi foarte mare.

Demonstrația NP – complet

Se poate observa uşor că problema este de tipul NP (aparține clasei NP), ținând cont de faptul că, dacă avem o submulțime de vârfuri, putem verifica dacă există muchii între oricare din două vârfuri într-un timp polinomial. Pentru a arăta că problema este de tipul *dificultate – NP* (*NP – dificil*), vom folosi o „reducere" a unei alte probleme de tipul NP – complet.

Presupunem că se cunoaşte deja rezultatul lui Cook, conform căruia problema satisfacerii booleene este NP – completă. Orice formulă booleană poate fi redusă, în mod eficient, la forma sa normală conjunctivă (*conjunctive normal form CNF*). În forma normală conjunctivă:

- Formula este o conjuncție (*and (şi)*) de propoziții.
- Fiecare propoziție reprezintă o disjuncție (*or(sau)*) de literali.
- Fiecare literal reprezintă fie o variabilă fie negația acesteia.

Spre exemplu, următoarea formulă constituie o formă CNF, unde ~ denotă negația:

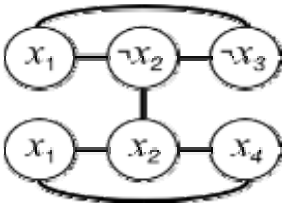
$$(x_1 \text{ sau} \sim x_2 \text{ sau} \sim x_3) \text{ şi } (x_1 \text{ sau } x_2 \text{ sau } x_4)$$

O astfel de formulă este *satisfăcătoare* dacă putem atribui valori *adevărat / fals* fiecărei variabile în parte astfel încât cel puțin un literal din fiecare propoziție să aibă valoarea de adevăr *adevărat*. Spre exemplu, orice atribuire a lui x_2 cu valoarea de adevăr *fals,* respectiv a lui x_4 cu valoarea de adevăr *adevărat* satisface formula mai sus amintită.

În cele ce urmează, se va prezenta o reducere de tipul *mai mulți - la unul* (timp - polinomial), de la CNF- la problema mulțimii independente. Mai întâi, se ataşează câte unu nod pentru fiecare literal din formulă; se includ duplicate ale vârfurilor pentru apariții multiple. Se trasează o muchie între:

1. Oricare doi literali, fiecare reprezentând negația celuilalt.
2. Oricare doi literali, ce se află în aceeaşi propoziție.

Astfel, în exemplul de mai sus, x_2 ar fi adiacent cu $\sim x_2$, primul x_1 ar fi adiacent cu $\sim x_2$, iar cel de-al doilea x_1 ar fi adiacent cu x_4.



Graful rezultat în urma reducerii, pentru exemplul de mai sus

Rămâne de văzut dacă acest graf are o mulțime independentă de mărime cel puțin *k*, unde *k* reprezintă numărul propozițiilor, dacă şi numai dacă formula rămâne satisfăcătoare.

Să presupunem că avem o atribuire ce satisface formula inițială. În aceste condiții putem alege un literal din fiecare propoziție, care devine adevărată prin atribuirea acestei valori. Această mulțime este independentă, deoarece include un literal din fiecare propoziție (nu există muchii de tipul 2), şi deoarece nici o atribuire nu transformă atât literalul cât şi negația acesteia propoziții adevărate(nu există muchii de tipul 1). Pe de altă parte, însă, presupunând că avem o mulțime independentă de mărime *k*, sau mai mare; nu poate conține oricare doi literali în aceeaşi propoziție, odată ce acestea constituie perechi adiacente. Dar, ținând cont de faptul că există cel puțin *k* noduri şi *k* propoziții, trebuie să avem cel puțin unul în fiecare propoziție (de fapt exact 1). De asemenea nu se poate întâmpla să coexiste un literal şi negația sa, deoarece există muchii între acestea. Această înseamnă că este uşor să alegem o atribuire astfel încât toate cele *k* propozițiile să devină adevărate, această atribuire satisfăcând formula inițială. Ceea ce face ca reducerea la mulțimea independentă să fie atât de simplă este capacitatea muchiilor de a exprima, în graf, *constrângerile*, dar şi necesitatea de a nu alege simultan un literal şi negația sa. Problema colorării grafurilor „beneficiază", de asemenea, de această proprietate.

7.3. Problema clicii

În teoria complexității computaționale, **problema clicii** este o problemă teoretică în grafuri, de complexitate NP. Problema se numără printre problemele inițiale, de complexitate NP, prezentate de către Richard Karp în cadrul seminarului din 1972 intitulat „Reductibility Among Combinatorial Problems". Această problemă a fost, de asemenea, menționată şi în articolul lui Cook, o introducere în teoria problemelor NP - complete.

Graf conținând clică de mărime 3

Clica într-un graf reprezintă o mulțime de vârfuri adiacente perechi, adică subgraful indus, care este un graf complet.

În cele din urmă, problema clicii constă în determinarea existenței unei clici într-un graf, a cărei mărime să fie cel puțin k (ordin predefinit). Odată identificate k sau mai multe vârfuri ce formează o clică, este trivial să demonstrăm acest lucru, fapt ce-i justifică complexitatea NP. Problema corespunzătoare de optimizare, **problema clicii maxime,** constă în găsirea celei mai mari clici din graf.

Complexitatea NP a problemei clicii derivă din complexitatea NP a problemei mulțimii independente, deoarece există o clică de mărime k (cel puțin) dacă şi numai dacă există o mulțime independentă de mărime k (cel puțin) în graful complementar. Acest lucru este uşor de observat, ținând cont de faptul că dacă un subgraf este complet, atunci subgraful complementar nu are muchii.

Algoritmi

Un algoritm „primitiv" de găsire a clicilor într-un graf constă în examinarea fiecărui subgraf ce conține cel puțin k vârfuri, şi îndeplinirea condiției de formare a unei clici. Algoritmul este polinomial (complexitate polinomială) dacă acel k coincide cu numărul vârfurilor, sau cu o constantă mai puțin, dar nu dacă k este, să spunem, jumătate din numărul total al vârfurilor. Numărul total al clicilor de mărime k a unui graf de mărime $|V|$ este egal cu

$$\binom{|V|}{k} = \frac{|V|!}{k!(|V|-k)!}.$$

În mod euristic am începe prin considerarea fiecărui nod ca fiind o clică, iar mai apoi să se unească aceste clici în altele mai mari până când acest lucru nu mai este posibil.

Două clici A şi B pot fuziona dacă fiecare vârf din clica A este adiacent fiecărui vârf din clica B. Aceasta presupune un cost, în ceea ce priveşte timpul, liniar (liniar în numărul muchiilor), dar poate eşua în găsirea unei clici mari, întrucât două sau mai multe părți a clicii mari vor fi fost fuzionat anterior, prin intermediul unor vârfuri care nu aparțin clicii.

Se găseşte, totuşi, cel puțin o clică maximală, care nu este conținută în nici o altă clică mai mare.

Unele cazuri mai speciale, pot fi rezolvate într-un timp mai scurt decât cel exponențial. Pentru $k = 3$, algoritmul are o complexitate $O(n^{1,41})$, unde n reprezintă numărul muchiilor.

7.4. Determinarea mulțimilor stabile maximale

Fie G = [X,U] un graf simplu cu mulțimea de vârfuri $X = \{x_1, x_2, ..., x_n\}$. Notăm cu $\mathbf{S}_n \subset \mathbf{S}$ familia mulțimilor stabile maximale ale grafului G. Există mai mulți algoritmi pentru determinarea familiei $\mathbf{S}_n$. În continuare vom prezenta algoritmul lui Bednarek şi Taulbee. Pentru acesta să punem pentru $1 \le k \le n$:

$$X_k = \{x_1, x_2, ..., x_k\},$$

$$G_k = [X_k]_G,$$

$$X_{k+1} = \{y \mid y \in X_k, [y, x_{k+1}] \notin U\},$$

şi fie $\mathbf{S}_k$ familia mulțimilor stabile maximale ale grafului G_k.

Algoritmul B – T. (Bednarek şi Taulbee)

1. Se consideră $X_1 = \{x_1\}$, $\mathbf{S}_1 = \{x_1\}$ (k = 1).
2. Se determină familia $I_k = \{T \mid T = S \cap Y_{k+1}, S \in \mathbf{S}_k\}$ şi familia I'_k a mulțimilor maximale față de incluziune din I_k.
3. Se determină familia $\mathbf{S}^*_{k+1}$ după cum urmează:

   Pentru fiecare $S \in \mathbf{S}_k$ punem

   $$S \cup \{x_{k+1}\} \in \mathbf{S}^*_{k+1}, \text{ dacă } S \subseteq Y_{k+1}$$

   sau dacă $S \not\subseteq Y_{k+1}$ punem $S \in \mathbf{S}^*_{k+1}$ şi $\{x_{k+1}\} \cup (S \cap Y_{k+1}) \in \mathbf{S}^*_{k+1}$ atunci şi numai atunci când $S \cap Y_{k+1} \in I'_k$.

   Familia $\mathbf{S}^*_{k+1}$ conține numai mulțimile specificate mai sus.
4. Se determină familia $\mathbf{S}_{k+1}$ a mulțimilor maximale din $\mathbf{S}^*_{k+1}$.
5. Algoritmul se termină când k = n – 1.

În ultima fază obținem familia $\mathbf{S}_n$ a mulțimilor stabile maximale în graful $G_n = G$. Pentru justificarea acestui algoritm demonstrăm următoarea afirmație:

Teoremă.

Pentru fiecare k = 1, 2, …, n – 1 familia $\mathbf{S}_{k+1}$ furnizată de algoritmul B – T., este familia tuturor mulțimilor stabile maximale ale grafului G_{k+1}.

Demonstrație.

Presupunem că $\mathbf{S}_k$ este familia mulțimilor stabile maximale ale grafului G_k. Este suficient să arătăm că orice element al familiei $\mathbf{S}^*_{k+1}$ este o

mulțime stabilă a grafului G_{k+1} şi că orice mulțime stabilă maximală a acestui graf aparține familiei $\mathbf{S}^*_{k+1}$.

Prima parte a afirmației rezultă din modul de construcție al familiei $\mathbf{S}^*_{k+1}$, deoarece $\mathbf{S}^*_{k+1}$ conține mulțimi sau submulțimi din $\mathbf{S}_k$ (care fiind stabile în G_k, sunt stabile şi în graful G_{k+1}) la care se adaugă elementul x_{k+1} dacă şi numai dacă x_{k+1} nu este adiacent în G_{k+1} cu nici unul dintre elementele mulțimii sau submulțimii considerate.

Fie acum S o mulțime stabilă maximală a grafului G_{k+1}. Dacă $x_{k+1} \notin S$ atunci $S \in \mathbf{S}_k$ şi $S \not\subseteq Y_{k+1}$. Rezultă, conform pasului 3 al algoritmului (alternativa a doua) că $S \in \mathbf{S}^*_{k+1}$.

Dacă $x_{k+1} \in S$ atunci $T = S - \{x_{k+1}\}$ este stabilă în graful G_k şi $T \subseteq Y_{k+1}$.

Dacă $T \in \mathbf{S}_k$, atunci

$$S = T \cup \{x_{k+1}\} \in \mathbf{S}^*_{k+1}.$$

În cazul când $T \notin \mathbf{S}_k$ există o mulțime $T' \in \mathbf{S}_k$ cu $T \subset T'$. Avem

$$T \subseteq T' \cap Y_{k+1}$$

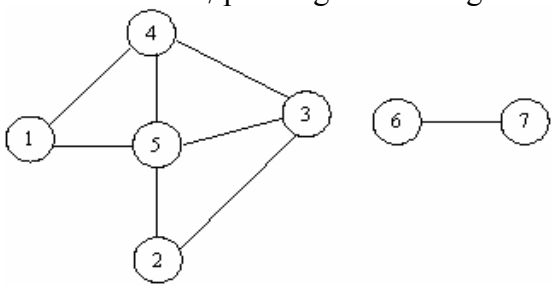
şi din cauza maximalității lui S rezultă că

$$T = T' \cap Y_{k+1},\ T' \not\subseteq Y_{k+1},\ T' \cap Y_{k+1} \in I'_k.$$

Deci $S = (T' \cap Y_{k+1}) \cup \{x_{k+1}\} \in \mathbf{S}^*_{k+1}$, şi astfel, teorema este demonstrată.

Exemplu.

Să se determine familia $\mathbf{S}_7$ pentru graful din figură:



1. $X_1 = \{1\}, \mathbf{S}_1 = \{1\}$ (k = 1)
   $Y_2 = \{1\}$
2. $I_1 = \{\{1\}\} = I'_1$
3. $\mathbf{S}^*_2 = \{\{1,2\}\}$

4. $\mathbf{S}_2 = \{\{1,2\}\}$
$X_2 = \{1,2\}$, $\mathbf{S}_2 = \{\{1,2\}\}$ $(k = 2)$
$Y_3 = \{1\}$

2. $I_2 = \{\{1\}\} = I'_2$

3. Avem $\{1,2\} \not\subseteq Y_3$ şi $\{1,2\} \cap Y_3 \in I'_2$. Deci $\mathbf{S}_3^* = \{\{1,2\},\{1,3\}\}$.

4. $\mathbf{S}_3 = \mathbf{S}_3^*$
$Y_4 = \{2\}$

2. $I_3 = \{\{2\}, \emptyset\}$
$I'_3 = \{\{2\}\}$

3. $\mathbf{S}_4^* = \{\{1,2\},\{2,4\},\{1,3\}\}$

4. $\mathbf{S}_4 = \mathbf{S}_4^*$
$Y_5 = \emptyset$

2. $I_4 = \{\emptyset, \emptyset, \emptyset, \emptyset\}$, $I'_4 = \{\emptyset\}$

3. $\mathbf{S}_5^* = \{\{1,2\},\{5\},\{2,4\},\{5\},\{1,3\},\{5\}\}$

4. $\mathbf{S}_5 = \{\{1,2\},\{2,4\},\{1,3\},\{5\}\}$
$Y_6 = \{1,2,3,4,5\}$

2. $I_5 = \{\{1,2\},\{2,4\},\{1,3\},\{5\}\} = I'_5$

3. $\mathbf{S}_6^* = \{\{1,2,6\},\{2,4,6\},\{1,3,6\},\{5,6\}\}$
$\mathbf{S}_6 = \{\{1,2,6\},\{2,4,6\},\{1,3,6\},\{5,6\}\}$
$Y_7 = \{1,2,3,4,5\}$

2. $I_6 = \{\{1,2\},\{2,4\},\{1,3\},\{5\}\} = I'_6$

3. $\mathbf{S}_7^* = \{\{1,2,6\},\{1,2,7\},\{2,4,6\},\{2,4,7\},\{1,3,6\},$
$\{1,3,7\},\{5,6\},\{5,7\}\} = \mathbf{S}_7$.

Deoarece $k = n - 1 = 6$, algoritmul se opreşte. Familia mulţimilor stabile maximale în graful G considerat este $\mathbf{S}_7$, iar numărul de stabilitate este $\alpha(G) = 3$.

Observaţii.

a) Determinare familiei $\mathbf{C_n}$ a clicilor maximale în graful G revine, pe baza observaţiei b) anterioare, la determinarea familiei mulţimilor stabile maximale în graful $\bar{G}$ (complementarul grafului G).

b) Determinarea familiei $\mathbf{S}_n$ şi a familiei $\mathbf{C}_n$ este necesară printre altele în problema găsirii unei acoperiri minimale a mulţimii vârfurilor unui graf cu mulţimi stabile (problema colorării vârfurilor) şi respectiv cu clici.