C Clasificarea problemelor din perspectiva complexității procesului de rezolvare

Cursul caracterizează problemele din punctul de vedere al naturii procesului de rezolvare şi, implicit, al dificultății rezolvării.

Definiția C.12 Spunem că un algoritm este determinist dacă fiecare operație conținută, de control sau de prelucrare a datelor, are un rezultat unic determinat.

Caracteristica esențială este serialitatea algoritmului. Execuția unui algoritm determinist, pentru o instanță dată a problemei rezolvate, conduce la o secvență a operațiilor efectuate, astfel încât pentru orice moment de timp `t` din cursul execuției există o singură operație derulată la momentul `t`. Mulțimea operațiilor efectuate în cursul rezolvării este echipotentă cu mulțimea momentelor de timp la care sunt executate aceste operații. Prin urmare, comportarea algoritmului poate fi descrisă printr-o funcție `Timp→Operații` pentru fiecare instanță a problemei rezolvate.

Definiția C.13 Un algoritm este nedeterminist dacă are operații al căror rezultat nu este unic definit ci este o valoare dintr-o mulțime finită de posibilități.

Caracteristica esențială este paralelismul nerestricționat al algoritmului. Execuția unui algoritm nedeterminist, pentru o instanță dată a problemei rezolvate, conduce la o structură arborescentă de operații. Operațiile de pe o cale din arbore sunt efectuate serial, în timp ce operațiile de pe căi diferite sunt efectuate în paralel. Execuția algoritmului pentru o instanță a problemei rezolvate nu mai poate fi descrisă printr-o funcție `Timp→Operații`, ci printr-o o relație peste mulțimea `Timp` × `Operații`. Pentru a construi algoritmi nedeterminişti introducem următoarele operații:

• `choice(A)`, unde `A` este o mulțime finită de valori, construieşte, pentru fiecare valoare `x` din `A`, o copie a algoritmului executat, copie ce conține copii ale tuturor variabilelor algoritmului cu valorile avute în momentul execuției `choice(A)`. Copiile algoritmului sunt executate în paralel şi independent una față de celelalte. Complexitatea operației `choice` este $\Theta(1)$, iar rezultatul reîntors pentru copia algoritmului corespunzătoare valorii `x` din `A` este chiar `x`.

• `fail` termină cu insucces execuția copiei algoritmului în care se află. Celelalte copii aflate în execuție ale algoritmului nu sunt afectate. Complexitatea operației este $\Theta(1)$.
`success` termină cu succes execuția copiei algoritmului în care se află şi a tuturor celorlalte copii. Execuția întregului algoritm este terminată. Complexitatea operației este $\Theta(1)$.

Un algoritm nedeterminist se termină cu succes doar dacă există o operație `success` efectuată în cursul execuției algoritmului. Algoritmul se termină cu insucces atunci când toate copiile algoritmului se termină prin `fail`. Individual, `success` şi `fail` nu au rezultat. Prin convenție, un algoritm nedeterminist terminat cu succes reîntoarce valoarea `1`; dacă algoritmul se termină cu insucces valoarea reîntoarsă este `0`.

Un algoritm nedeterminist **Alg** asociat unei probleme $Q:I\to\{0,1\}$ emulează procesul de ghicire a căii cu complexitate minimă corespunzătoare rezolvării problemei **Q**. Pentru date de intrare $i\in I$ fixate, astfel încât **Alg(i)=1**, complexitatea algoritmului se defineşte ca sumă a complexităţii operaţiilor din calea cu complexitate minimă care termină algoritmul cu **success**. Astfel măsurată, complexitatea se numeşte *angelică*. Spunem că un algoritm nedeterminist are complexitate angelică limitată de o funcţie $f:N\to R_+$, dacă pentru orice date de intrare $i\in I$ de dimensiune **n**, astfel încât **Alg(i)=1**, complexitatea angelică a algoritmului pentru datele **i** este **O(f(n))**.

Exemplul C.5 Fie ecuaţia **Ec(x) = 0**. Care dintre următoarele soluţii este corectă: (1) $x = val_1$; (2) $x = val_2$; (3) $x = val_3$ ... (n) $x = val_n$.

```
// Algoritm determinist
ecuație(val,n) {
   x = Rezolvare(Ec);
   for(i=1; i≤n; i++)
     if(x = vali) return i;
}
```

```
// Algoritm nedeterminist
N_ecuație(val,n) {
   i = choice(1..n);
   if(Ec(vali)≠0){print i; success;}
   fail;
}
```

Rezolvarea deterministă impune cunoaşterea rezolvării ecuaţiei **Ec(x)=0**. Rezolvarea nedeterministă impune doar cunoaşterea operaţiilor folosite în ecuaţie, iar complexitatea este **Θ(4)+Timp_verificare(Ec)**. Dacă algoritmul **N_ecuație** este rescris ca un algoritm determinist, complexitatea devine **Θ(n)*Timp_verificare(Ec)** faţă de **Θ(n)+Timp_rezolvare(Ec)**, cât cere algoritmul **ecuație**.

Exemplul C.6 Fie **A** un vector cu **n** întregi strict pozitivi. Să se sorteze elementele vectorului **A**. Algoritmul nedeterminist de sortare construieşte permutările elementelor din **A**, plasând fiecare element într-un vector auxiliar **B**. Restricţia $A_i>0$, **i=1,n**, ajută la determinarea uşoară a poziţiilor deja ocupate din **B**. La fiecare operaţie **choice**, algoritmul şi vectorul **B** sunt duplicate. În final, rezultatul sortării este tipărit.

```
N_sort(A,n) {
   for(i = 1; i ≤ n; i++) B[i] = 0;

   // Generare soluție potențială
   for(i = 1; i ≤ n; i++) {
       // Poziționare Ai în vectorul B. Ignorăm construcția
       // mulțimii 1..n care nu măreşte complexitatea N_sort
       j = choice(1..n);
       if(Bj ≠ 0) fail;
       Bj = Ai;
   }
   // Test final sortare
   for(i= 1; i < n; i++) if(Bi > Bi+1) fail;
   print(B,n);
   success;
}
```

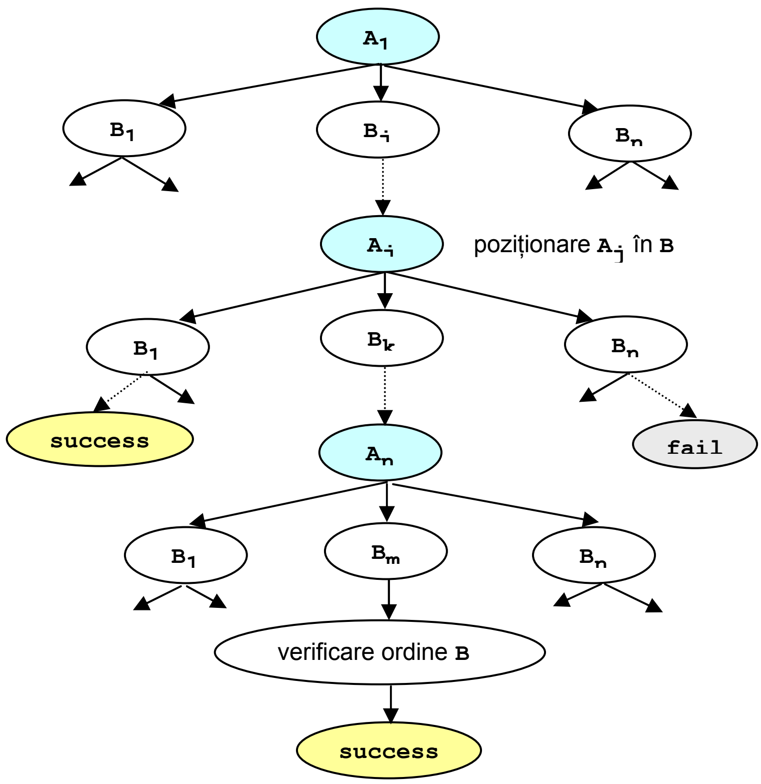


Figura C.3 Execuția algoritmului nedeterminist `N_sort`

Deşi complexitatea temporală a algoritmului este $\Theta(n)$, spațiul total consumat pare a fi proporțional cu `n!`. Totuşi, considerând execuția unui algoritm nedeterminist ca proces de ghicire şi execuție a unei căi cu complexitate minimă din arborele execuției algoritmului, complexitatea spațială a algoritmului se limitează la complexitatea spațială a acelei căi. De exemplu, pentru algoritmul de sortare nedeterministă complexitatea spațială este $\Theta(n\ lg(n))$.

Exemplele date sugerează că rezolvarea unei probleme cu un algoritm nedeterminst are două faze:

1. Construcția concurentă a soluțiilor potențiale. O soluție potențială nu este în mod necesar o soluție a problemei. Construcția este efectuată fără a ține seama de proprietățile impuse soluțiilor şi, de aceea, poate fi asemănată cu un proces de ghicire.

2. Verificarea serială a fiecărei soluții potențiale construite în faza (1). Se testează dacă soluția potențială satisface proprietățile cerute de problemă, deci dacă, într-adevăr, este o soluție a problemei. Complexitatea verificării influențează esențial complexitatea întregului algoritm nedeterminist.

Similar cu caracterizarea algoritmilor din punctul de vedere al execuției, putem clasifica problemele în raport cu natura algoritmilor de rezolvare. Prin convenție, spunem că natura unei probleme este:

- Deterministă poate fi rezolvată printr-un algoritm determinist cu complexitate polinomială (rezolvarea este tractabilă).

- Nedeterministă dacă poate fi rezolvată printr-un algoritm nedeterminist cu complexitate polinomială şi nu există un algoritm de rezolvare determinist cu aceeaşi proprietate.

Deoarece algoritmii determinişti sunt cazuri particulare ale celor nedeterminişti, clasa problemelor nedeterministe include pe cea a problemelor deterministe.

O perspectivă alternativă a problemelor deterministe şi nedeterministe se bazează pe legătura intuitivă ce există între natura algoritmilor de rezolvare şi modul de calcul al predicatului de navigare în spațiul stărilor problemei rezolvate.

Definiția C.14 Fie $G_Q=\{S,E\}$ spațiul stărilor unei instanțe oarecare a unei probleme decizionale Q: G_Q este un graf orientat cu nodurile (stările S) şi arcele (tranzițiile valide) $E \subseteq S \times S$. Fie $T: E \to \{0,1\}$ un predicat de navigare în spațiul stărilor, iar $M_T = \{x \in E \mid x$ duce spre o soluție a problemei$\}$ mulțimea de adevăr a lui T.

- Problema Q este deterministă dacă valoarea $T(x)$ poate fi calculată fără a enumera elementele M_T. Algoritmul care calculează $T(x)$ nu foloseşte un generator al mulțimii M_T pentru a decide dacă x este în M_T. M_T este tratată ca mulțime recursivă.

- Problema Q este nedeterministă dacă valoarea $T(x)$ este calculabilă prin enumerarea elementelor din M_T, deci pe baza unui generator al lui M_T. Pentru fiecare soluție generată a problemei se testează dacă x face parte din soluție. M_T este tratată ca mulțime recursiv-numărabilă.

Imposibilitatea sau dificultatea calculului apriori (fără a investiga spațiul stărilor problemei) al mulțimii M_T conduce la varianta, potențial mai eficientă, a străbaterii în paralel a diverselor căi din spațiul stărilor problemei (sau a ghicirii drumului celui mai scurt până la soluție), deci la un algoritm nedeterminist.

Pentru o problemă rezolvabilă mecanic, definiția (C.14) caracterizează, implicit, complexitatea rezolvării problemei. Pe o maşină de calcul serială, care execută o singură operație la un moment dat, navigarea în spațiul stărilor unei probleme deterministe este mai rapidă decât navigarea în spațiul stărilor unei probleme nedeterministe. Pentru o problemă deterministă arcul curent ales pentru avans conduce cert la soluție. Nu sunt necesare reveniri şi apoi noi avansuri. Pentru o problemă nedeterministă, determinarea arcelor care pleacă dintr-un nod şi duc spre soluție se efectuează prin încercarea de a străbate arcele, într-o ordine oarecare sau o ordine aleasă (în funcție de euristici de rezolvare). Eşecul conduce la revenirea în nod şi continuarea procesului de rezolvare cu alt arc.

De exemplu, să considerăm spațiul stărilor din figura C.4, unde soluția este un drum de la starea inițială s_0 la o stare finală s_f. Dacă problema este nedeterministă decizia "*arcul* (s_i,s_j) *face parte din soluție?*" impune explorarea subgrafului G_j. Dacă soluția este găsită, arcul (s_i,s_j) este implicit marcat ca fiind în soluție. Altfel, are loc revenirea la s_i, urmată de parcurgerea altui arc. În cazul unei probleme deterministe, avansul pe arcul (s_i,s_j) are loc doar dacă el face parte din soluție.

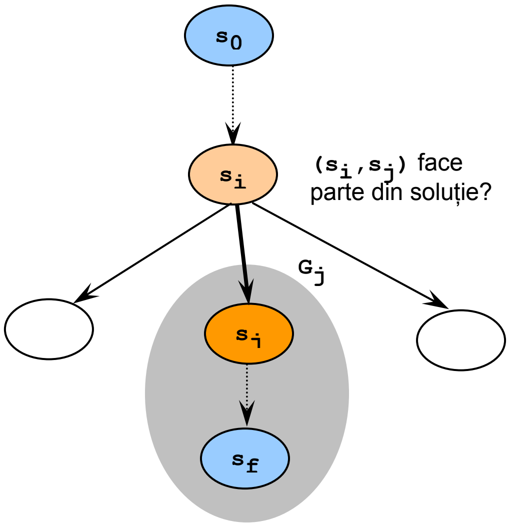


Figura C.4 Spațiul stărilor unei probleme

Dacă spațiul stărilor este un arbore de ordin m cu înălțimea h, iar problema este deterministă, rezolvarea cere un timp $O(h)$. Dacă problema este nedeterministă, atunci rezolvarea cu un algoritm serial (determinist) cere un timp $O(m^h)$.

În general, dacă o problemă este rezolvabilă în timp t_1 cu un algoritm nedeterminist `Alg` şi în timp t_2 cu un algoritm determinist rezultat din rescrierea lui `Alg`, atunci $t_1 \le t_2$. Într-adevăr, algoritmul determinist, simulează operația `choice` prin avans şi revenire (backtrack) pentru a străbate toate drumurile parcurse în paralel de algoritmul nedeterminist, deci lungimea totală a drumului parcurs în spațiul stărilor va fi, în cazul cel mai defavorabil, suma lungimilor drumurilor parcurse de algoritmul nedeterminist.

Definiția (C.14) nu impune în mod necesar terminarea procesului de rezolvare, deci decidabilitatea problemei, ci doar terminarea cât mai rapidă a algoritmului în cazul în care problema acceptă soluție pentru datele de intrare. Conceptele de determinism şi nedeterminism nu trebuie confundate cu cele de decidabilitate şi semi-decidabilitate. Decidabilitatea şi semi-decidabilitatea sunt proprietăți intrinseci ale problemelor şi caracterizează rezolvabilitatea lor mecanică. Determinismul şi nedeterminismul caracterizează tehnica de rezolvare şi, implicit, complexitatea problemelor rezolvabile mecanic. Un algoritm nedeterminist priveşte tranzițiile (arcele) din spațiul stărilor problemei rezolvate ca şi cum ar fi valori ale unei mulțimi recursivnumărabile, valori ce pot fi obținute prin generare (concurentă). Considerând spațiul stărilor problemei finit local (aşa cum cere operația `choice`) şi fără drumuri de lungime infinită, atunci mulțimea tranzițiilor posibile este recursivă, deci problema, rezolvată nedeterminist pentru o dimensiune finită a datelor, este decidabilă. Cu ce efort de calcul, este o altă problemă discutată în următoarea secțiune.

NP-completitudine

Clasificarea problemelor efectuată conform definițiilor de mai sus este calitativă. Ne interesează în ce măsură o problemă poate fi rezolvată cu efort acceptabil, cuantificabil cantitativ. Evident, ne interesează acest aspect pentru problemele care pot fi rezolvate efectiv cu calculatorul, deci din punctul de vedere al problemelor decidabile. În cele ce urmează, efortul de rezolvare este privit ca timp de rezolvare.

Definiția C.15 O problemă este:

- *Tractabilă (determinist)* dacă poate fi rezolvată folosind un algoritm determinist tractabil. Un algoritm tractabil are complexitate polinomială: dacă n este dimensiunea problemei, timpul de rezolvare este $O(n^k)$, unde k este o constantă. Similar, spunem că problema este *tractabilă nedeteriminist* dacă poate fi rezolvată folosind un algoritm nedeterminist tractabil (cu complexitate angelică polinomială).

- *Intractabilă* dacă toți algoritmii determinişti de rezolvare au complexitate supra-polinomială (dacă n este dimensiunea problemei, timpul de rezolvare este $\Omega(k^n)$, unde k este o constantă).

Noțiunea de tractabilitate este discutabilă. Astfel complexitatea n^{100} este uriaşă, chiar pentru supercalculatoarele contemporane. Pentru o maşină cu 10^{10} operații pe secundă şi pentru $n=100$ sunt necesare 10^{190} secunde. Tractabilitatea, aşa cum este definită mai sus, este acceptabilă comparativ şi mai ales atunci când gradul polinomului este modest. Chiar şi aşa, există probleme rezolvabile teoretic în timp supra-polinomial, dar care pentru aplicațiile practice uzuale acceptă soluții cu performanțe acceptabile. Un exemplu este problema sintezei de tip în limbajele funcționale, a cărei complexitate este exponențială. Totuşi, în practică, algoritmii de sinteză de tip se comportă suficient de bine, cazurile "dificile" de sinteză fiind rare în programarea funcțională de rutină.

Definiția C.16 În funcție de natura algoritmilor de rezolvare şi complexitatea acestora, *problemele de decizie* (notate `Dec`) pot fi divizate în două mari clase:

- $P = \{Q \in Dec \mid$ există algoritmi determinişti tractabili care rezolvă $Q\}$.

- $NP = \{Q \in Dec \mid$ există algoritmi nedeterminişti tractabili care rezolvă $Q\}$.

Intuitiv, clasa `P` (sau `PTIME`) corespunde problemelor deterministe tractabile, în timp ce clasa `NP` (sau `NPTIME`) include în plus problemele nedeterministe tractabile, rezolvabile determinist în timp supra-polinomial. Orice problemă deterministă poate fi privită şi din perspectivă nedeterministă, de unde rezultă următoarea teoremă.

Teorema C.2 $P \subseteq NP$.

Fie $Q \in P$, o problemă rezolvabilă printr-un algoritm determinist cu complexitate polinomială. Înseamnă că algoritmul străbate un drum de lungime polinomială în spațiul stărilor problemei. Evident, se poate construi un algoritm nedeterminist care la execuție urmează, printre alte căi, şi calea de lungime polinomială parcursă de algoritmul determinist. Deci algoritmul nedeterminist are complexitate polinomială. ■

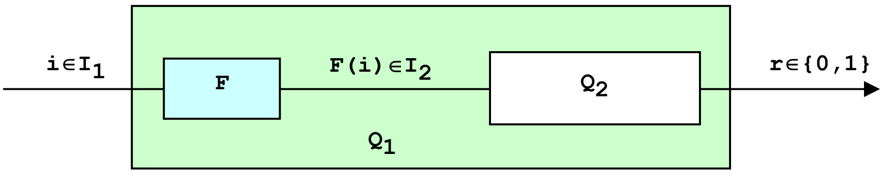


Figura C.5 Reductibilitatea problemelor

Definiția C.17 Fie Q_1 şi Q_2 două probleme abstracte de decizie. Se spune că problema Q_1 este reductibilă în timp polinomial la problema Q_2 (relație notată $Q_1 \leq_P Q_2$) dacă există un algoritm F, determinist şi cu complexitate polinomială, care transformă problema Q_1 în Q_2, aşa cum se sugerează în figura C.5. În fapt, F transformă datele $i \in I_1$ ale lui Q_1 în date $F(i) \in I_2$ pentru Q_2 astfel încât:

$$\forall i \in I_1 \bullet (Q_1(i)=1 \Leftrightarrow Q_2(F(i))=1)$$

Lema C.1 Dacă $Q_2 \in P$ şi $Q_1 \leq_P Q_2$ atunci $Q_1 \in P$.

Dacă $Q_2 \in P$, iar m este dimensiunea datelor lui Q_2, atunci există un algoritm determinist A_2 cu complexitate $O(m^k)$ care rezolvă Q_2. Să presupunem că n este dimensiunea datelor lui Q_1, iar $O(n^r)$ este complexitatea lui F, k şi r fiind constante. Se poate construi algoritmul $A_1(i)\{return\ A_2(F(i))\}$ a cărui complexitate este $O(n^r)+O(m^k) = O(n^r+n^{rk})$. Într-adevăr, $F(i)$ nu poate produce date cu o dimensiune mai mare decât $O(n^r)$ într-un timp $O(n^r)$. Cu alte cuvinte, dimensiunea datelor primite de algoritmul A_2 este $m = O(n^r)$ ■

Lema C.2 Relația de reducere polinomială $\leq_P$ este tranzitivă.

Fie Q_1, Q_2 şi Q_3 probleme cu date de tip I_1, I_2 şi, respectiv, I_3 astfel încât să avem $Q_1 \leq_P Q_2$ şi $Q_2 \leq_P Q_C$.

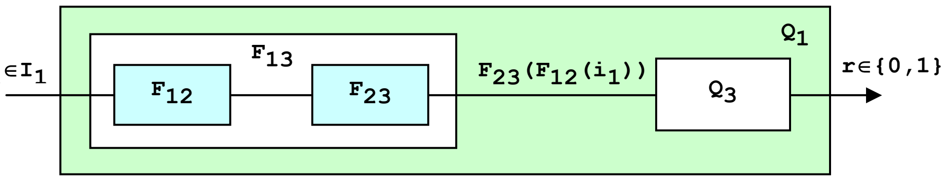


Figura C.6 Tranzitivitatea relației $\leq_P$

a) Conform definiției (C.17) există un algoritm F_{12} care transformă $i_1 \in I_1$ în $i_2 \in I_2$ în timpul polinomial $O(n^r)$, unde n este dimensiunea datelor i_1 ale lui Q_1. De asemenea, există un algoritm F_{23} care transformă i_2 în $i_3 \in I_3$ în timpul polinomial $O(m^k)$, unde m este dimensiunea datelor i_2 ale lui Q_2. Fiind determinist, algoritmul $F_{12}(i_1)$ nu poate produce date i_2 cu o dimensiune m mai mare decât $O(n^r)$ într-un timp $O(n^r)$. Rezultă că algoritmul $F_{13}(i_1) = F_{23}(F_{12}(i_1))$ transformă i_1 în i_3 în timpul $O(n^r) + O(m^k) = O(n^r + n^{kr})$. Deci algoritmul F_{13} este determinist şi tractabil.

b) Din relațiile de reductibilitate $Q_1 \leq_p Q_2$ şi $Q_2 \leq_p Q_3$ rezultă proprietatea: $\forall i \in I_1 \bullet (Q_1(i_1)=1 \Leftrightarrow Q_3(F_{13}(i_1))=1)$.

În final, conform definiției (C.17), din (a) şi (b) rezultă $Q_1 \leq_p Q_3$ ■

Definiția C.18 O problemă Q este:

- NP-dură dacă pentru orice problemă $Q' \in NP$ avem $Q' \leq_p Q$.
- NP-completă dacă Q este NP-dură şi $Q \in NP$.

Proprietatea de completitudine asociată unei probleme Q dintr-o clasă de complexitate C arată că problema respectivă este dintre "cele mai complicate" probleme din C, astfel încât, dacă Q ar face parte dintr-o clasă de complexitate C' inferioară lui C, atunci toate problemele din C ar fi în C'. În particular, dacă o problemă NP-completă ar putea fi rezolvată folosind un algoritm determinist cu complexitate polinomială atunci $P = NP$.

Teorema C.3 O problemă Q_2 este NP-dură dacă şi numai dacă există o problemă NP-completă Q_1 astfel încât $Q_1 \leq_p Q_2$.

a) Fie Q_1 NP-completă astfel încât $Q_1 \leq_p Q_2$. $Q_1 \in NP$ şi $Q' \leq_p Q_1$ pentru orice $Q' \in NP$. Relația $\leq_p$ este tranzitivă, deci $Q' \leq_p Q_2$. Q_2 este NP-dură.

b) Fie Q_2 NP-dură. Deoarece pentru orice $Q_1 \in NP$ avem $Q_1 \leq_p Q_2$ putem alege Q_1 NP-completă. ■

Teorema C.4 Problemele NP-complete formează o clasă de echivalență în raport cu relația $\leq_p$.

Fie Q_1 şi Q_2 probleme NP-complete. Conform definiției (C.18), avem:

a) $Q' \in NP \Rightarrow Q' \leq_p Q_1$. Dar $Q_2 \in NP$, deci $Q_2 \leq_p Q_1$
b) $Q' \in NP \Rightarrow Q' \leq_p Q_2$. Dar $Q_1 \in NP$, deci $Q_1 \leq_p Q_2$ ■

Teorema C.5 Dacă există o problemă NP-completă Q, $Q \in P$, atunci $P = NP$.

Conform definiției (C.18), $Q \in$ NP-complete impune ca pentru orice $Q' \in NP$ să avem $Q' \leq_p Q$. Dacă $Q \in P$ atunci, prin lema (C.1), rezultă $Q' \in P$. Obținem $NP \subseteq P$ şi, prin teorema (C.2), $P \subseteq NP$, deci $P = NP$. ■

Corolarul C.1 Dacă există o problemă **NP-completă** Q, astfel încât $Q \in P$ atunci toate problemele **NP-complete** sunt în **P**.

Cu alte cuvinte, dacă există o problemă **NP-completă** rezolvabilă printr-un algoritm determinist tractabil, atunci toate problemele **NP-complete** pot fi rezolvate determinist în timp polinomial. Corolarul rezultă direct din teoremele (C.4) şi (C.5) ■

Teorema C.6 Dacă există o problemă Q **NP-dură** şi $Q \in P$, atunci $P = NP$.

Conform teoremei (C.3), există Q' **NP-completă** astfel încât $Q' \leq_P Q$, iar prin lema (C.1) dacă $Q \in P$ atunci $Q' \in P$. Deci, conform teoremei (C.5) $NP = P$. ■

Corolarul C.2 Dacă există o problemă **NP-dură** Q, astfel încât $Q \in P$ atunci toate problemele **NP-complete** sunt în **P**.

Dacă există o problemă **NP-dură** $Q \in P$, din teorema (C.6) avem $P=NP$. Conform definiției (C.18), problemele **NP-complete** sunt în clasa **NP** şi, în acest caz, sunt în **P**. ■

Se observă că problemele **NP-dure** nu formează o clasă de echivalență, aşa cum se întâmplă în cazul problemelor **NP-complete**. Totuşi, o problemă **NP-dură** poate fi utilă pentru stabilirea clasei de complexitate a altei probleme.

Pentru a demonstra că o problemă Q este **NP-completă** (sau doar **NP-dură**) este necesar să se arate că:

a) Pentru orice problemă $Q' \in NP$ există relația $Q' \leq_P Q$. Un mod mai simplu de a verifica această proprietate constă în construcția unui algoritm determinist şi tractabil F care reduce o problemă Q", deja cunoscută ca **NP-dură** sau **NP-completă**, la Q. Din relația $Q'' \leq_P Q$ şi din $Q'' \in NPC$, adică $\forall Q' \in NP \bullet Q' \leq_P Q''$, prin tranzitivitatea relației $\leq_P$ rezultă că toate problemele din **NP** sunt reductibile în timp polinomial la Q.

b) Există un algoritm nedeterminist de rezolvare în timp polinomial a problemei Q, deci $Q \in NP$. Se construieşte efectiv algoritmul nedeterminist.

Etapa (a) stabileşte **NP**-duritatea problemei Q, conform teoremei (C.3), şi impune cunoaşterea unei probleme **NP-dure** sau **NP-complete**. Totodată, este cea mai complicată fază a demonstrației. Etapa (b), în conjuncție cu etapa (a), stabileşte **NP**-completitudinea problemei Q.

În urma discuției topologia teritoriului problemelor decizionale poate fi imaginată ca în figura C.7. Este doar o variantă posibilă. Se cunosc multe probleme **NP-complete** şi **NP-dure** (marea masă a celor cu aplicabilitate practică), dar - până în prezent - pentru nici o problemă **NP-completă** sau **NP-dură** nu s-a descoperit un algoritm de rezolvare determinist şi tractabil. Totuşi, nu s-a demonstrat că asemenea algoritmi nu pot exista. Deşi nu se poate afirma riguros dacă $P \neq NP$, se bănuieşte că relația este adevărată. Din acest punct de vedere, încadrarea unei probleme Q în clasa **NPC (NP-complete)** sau **NPD (NP-dure)** evită căutarea unei rezolvări tractabile care ar exista doar dacă $P=NP$. În asemenea cazuri, o cale spre tractabilitate constă în acceptarea unei soluții aproximative cu un factor de aproximare cert sau garantat cu o probabilitate acceptabilă.

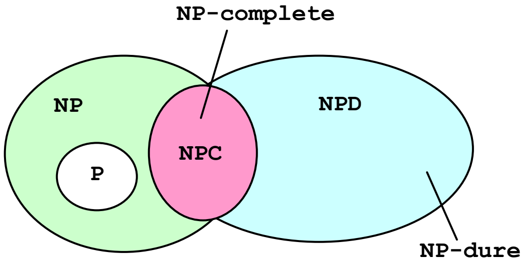


Figura C.7 O structură posibilă a spațiului problemelor de decizie

Clasificarea din această secțiune s-a concentrat asupra complexității temporale a algoritmilor, dar conceptele de compeltitudine şi duritate sunt valabile şi pentru complexitatea spațială şi, de asemenea, pentru alte clase de complexitate aşa cum sunt cele din ierarhia

$$\mathbf{LOGSPACE} \subseteq \mathbf{NLOGSPACE} \subseteq \mathbf{PTIME} \subseteq \mathbf{NPTIME} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE}$$

Verificarea durității unei probleme

Conform teoremei (C.3) pentru a verifica dacă o problemă **Q** este **NP-dură** este suficient să arătăm reductibilitatea polinomială $Q' \leq_p Q$, unde **Q'** este o problemă **NP-dură** sau **NP-completă**. În plus, dacă **Q** are un algoritm de rezolvare nedeterminist cu complexitate polinomială înseamnă că **Q** este **NP-completă**. Esențială este probarea **NP**-durității, care arată că problema nu are soluție deterministă şi tractabilă cunoscută.

Secțiunea conține exemple de verificare a apartenenței unor probleme la clasele **NP-complete** şi **NP-dure**. Primul exemplu tratează un caz teoretic extrem, cel al problemei terminării algoritmilor (programelor), problemă nedecidabilă. Celelalte exemple au importață practică, problemele analizate constituind ierarhia din figura C.8. În cadrul discuției este interesantă reducerea unei probleme la altă problemă, mai ales atunci când problemele par foarte diferite, aşa cum este cazul determinării unei clici de dimensiune dată într-un graf neorientat versus problema **SAT**.

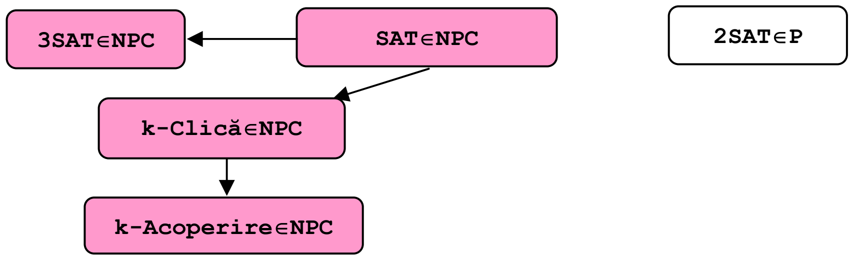


Figura C.8 Ierarhia problemelor analizate

Problema k-clicii unui graf

Definiția C.20 Fie un graf neorientat **G = (V,E)** cu nodurile **V** şi arcele **E**. O **k-clică** a lui **G** (o clică de dimensiune **k**) este un subgraf **G'=(V',E')** complet al lui **G** astfel încât **card(V')=k** (**G'** este complet dacă pentru orice **u∈V'** şi **v∈V'**, **(u,v)∈E'**).

Există o definiție echivalentă a **k-clicii**: un subgraf cu **k** noduri şi **k(k-1)/2** arce. Într-adevăr, să numerotăm cu $v_1, v_2, \ldots, v_k$ nodurile din **k-clică**. Nodul v_1 contribuie cu **k-1** arce către cele **k-1** noduri rămase; nodul v_2 contribuie cu **k-2** arce; în general, nodul v_j contribuie cu **k-j** arce. Deci numărul total de arce din **k-clică** este **1+2+...+k-1 = k(k-1)/2**.

De exemplu, graful din figura C.9 are **5** clici cu un nod (nodurile), **6** clici cu două noduri (arcele) şi **2** clici de dimensiune **3** cu nodurile **{a,b,c}** şi **{b,c,e}**.

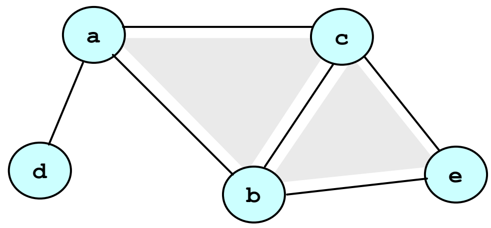


Figura C.9 `3-clici` într-un graf

Notăm `k-Clică` problema: să se determine dacă un graf `G` conține cel puțin o `k-clică`.

Teorema C.8 Problema `k-Clică` este `NP-completă`.

Demonstrația are două etape:

a) Arătăm că problema `k-Clică` este în `NP`, deci există un algoritm nedeterminist de rezolvare în timp polinomial;
b) Alegem o problemă `NP-completă`, fie ea `SAT`, şi demonstrăm reductibilitatea polinomială $\texttt{SAT} \leq_p \texttt{k-Clică}$.

a) `k-Clică` este în `NP`.

Construim algoritmul nedeterminist `N_k-Clică`. Complexitatea algoritmului este $\Theta(k^2)$. Într-adevăr, execuția algoritmului poate fi reprezentată ca un arbore în care secvența de verificare a `k-clicii` este pe nivelul `k`, complexitatea unei căii de la rădăcină la nivelul `k` fiind $\Theta(k^2)$[1]. Dacă graful este reprezentat printr-o matrice de adiacență, operația $(u,v)\notin E$ durează $\Theta(1)$, iar complexitatea testului `k-clicii` este, de asemenea, $\Theta(k^2)$, atunci când există soluție. Cum $k \leq n$, $n = card(V)$, complexitatea algoritmului este limitată la $O(n^2)$.

```
N_k-Clică(G,k){
    fie V nodurile, iar E arcele grafului G;
    V' = ∅; // Nodurile k-clicii
    for(i=1; i ≤ k; i++) {
       u = choice(V);
       if(u ∈ V') fail;
       V' = V' ∪ {u};
    }
    // V' formează o clică?
    for-each(u ∈ V')
       for-each(v ∈ V') if(u ≠ v ∧ (u,v)∉ E) fail;
    success;
}
```

[1] Se consideră că testul `u ∈ V'` este secvențial.

Observație. Folosind algoritmul decizional `N_k-Clică` putem rezolva problema `max-Clică` a clicii de dimensiune maximă dintr-un graf. Algoritmul nedeterminist `N_max-Clică` are complexitate $O(n^3)$.

```
N_max-Clică(G) {
    fie n numărul nodurilor din graful G;

    for(k=n; k > 0; k--)
       // Test existență k-clică în G
       if(N_k-Clică(G,k)) break;

    return k;
}
```

b) $SAT \leq_p$ `k-Clică`.

Fie $F = T_1 \wedge T_2 \wedge ... \wedge T_k$ o formulă CNF. Considerăm `Var` mulțimea variabilelor din `F`, iar `n = card(Var)`. Construim un graf neorientat $G_F=(V,E)$, echivalent formulei `F`, astfel:

- $V = \{\langle\alpha,i\rangle \mid \alpha\in T_i$, unde $\alpha=x$ sau $\alpha=\neg x$, cu $x\in Var$, este un literal din $T_i\}$

Nodurile grafului sunt dubleți $\langle\alpha,i\rangle$, unde α este un `<literal>` al formulei `F` (o variabilă sau o variabilă negată), iar `i` este indicele termenului din `F` în care apare α.

- $E = \{(\langle\alpha,i\rangle, \langle\beta,j\rangle) \mid \langle\alpha,i\rangle\in V \wedge \langle\beta,j\rangle\in V \wedge \alpha \neq \neg\beta \wedge i \neq j\}$

Prin convenție, notația $\alpha \neq \neg\beta$ înseamnă: literalii α şi β nu pot fi unul `x`, iar celălalt $\neg x$, pentru $x\in Var$. În schimb, α şi β pot fi identici sau pot conține variabile diferite. Aşadar, arcele grafului pot fi doar între noduri cu literali din termeni diferiți şi doar dacă literalii sunt identici sau corespund unor variabile diferite.

De exemplu, graful G_F ilustrat în figura C.10 (şi care are două `3-clici`: $\{\langle a,1\rangle,\langle b,2\rangle,\langle a,3\rangle\}$ şi $\{\langle a,3\rangle,\langle b,1\rangle,\langle b,2\rangle\}$) corespunde formulei:

$$F = \underbrace{(a \vee b)}_{T1} \wedge \underbrace{(\neg a \vee b)}_{T2} \wedge \underbrace{(a \vee \neg b)}_{T3}$$

.

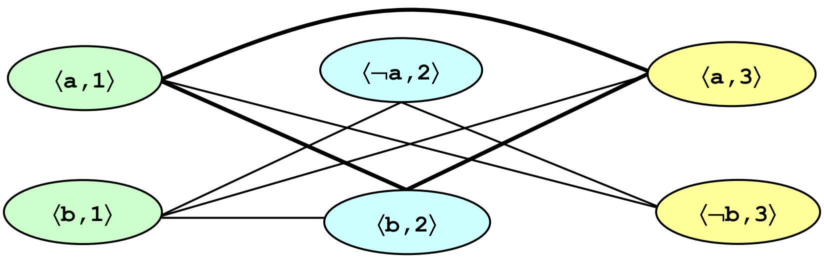


Figura C.10 Graful asociat formulei `F`

Să arătăm că graful G_F poate fi construit cu un algoritm determinist în timp polinomial în raport cu numărul variabilelor din **F**. Fie algoritmul de conversie **F -> G_F**:

```
construcție_GF(F) {
    fie T = {T1,T2,...,Tk} termenii formulei F;
    V = ∅; // nodurile din GF
    E = ∅; // arcele din GF

    // Construcția nodurilor lui GF
    for(i=1; i ≤ k; i++)
       for-each(α∈Ti) V = V ∪ {⟨α,i⟩};

    // Construcția arcelor lui GF
    for-each(⟨α,i⟩ ∈ V)
       for-each(⟨β,j⟩ ∈ V)
            if(α ≠ ¬β ∧ i ≠ j) E = E ∪ {(⟨α,i⟩,⟨β,j⟩)};
    return (V,E);
}
```

Pentru că fiecare variabilă din cele **n** poate să apară de cel mult două ori într-un termen, numărul literalilor dintr-un termen este limitat[2] la **2n**. Rezultă că ciclul de construcție a nodurilor lui G_F are complexitatea $O(kn)$, iar construcția arcelor are complexitatea $O(k^2n^2)$. Rezultă o complexitate a algoritmului polinomială în **n** şi **k**. De asemenea, considerând numărul termenilor limitat polinomial în raport cu **n**, complexitatea algoritmului **construcție_G_F** este polinomială în funcție de **n**.

Putem arăta acum că satisfiabilitatea formulei **F** este înrudită cu problema **k-Clică**.

Lema C.3 O formulă **CNF F** cu **k** termeni este satisfiabilă dacă şi numai dacă graful G_F are o **k-clică**.

F este satisfiabilă ⇒ G_F are o **k-clică**.

Dacă **F** este satisfiabilă atunci în fiecare termen T_i, $i=1,k$, există cel puțin un literal cu valoarea de adevăr **1**. Fie $V' = \bigcup_{i=1}^{k} \{\langle\alpha,i\rangle \mid \alpha\in T_i \wedge \alpha=1\}$ o submulțime cu **k** noduri din G_F, astfel încât **V'** să conțină un singur element $\langle\alpha,i\rangle$ pentru termenul T_i, $i=1,k$.

Să alegem la întâmplare două noduri distincte $\langle\alpha,i\rangle$ şi $\langle\beta,j\rangle$ din **V'**. În mod sigur $i \neq j$ şi observăm că $\alpha \neq \neg\beta$. De exemplu, dacă $\alpha = \neg\beta$ şi $\alpha = x$, $x\in Var$, atunci $\alpha = 1$ implică $x = 1$ şi ar însemna că $\beta = \neg x = 0$, ceea ce contrazice modul de construcție a lui **V'**, pentru că β trebuie să fie **1**. Similar, dacă $\alpha = \neg\beta$ şi $\alpha = \neg x$, $x\in Var$, atunci $\alpha = 1$ implică $x = 0$ şi ar însemna că $\beta = x = 0$.

[2] În particular, problema **3SAT**, unde termenii din formulă au cel mult câte trei literali distincți, este **NP-completă**.

Rezultă că arcul $(\langle\alpha,i\rangle, \langle\beta,j\rangle)$ satisface $i \neq j$ şi $\alpha \neq \neg\beta$, deci este un arc din graful G_F. Prin urmare, nodurile `V'` formează o `k-clică` în G_F.

G_F are o `k-clică` $\Rightarrow$ `F` este satisfiabilă.

Fie `V'` nodurile dintr-o `k-clică` a grafului G_F. Pentru că arcele grafului nu pot fi decât între noduri $\langle\alpha,i\rangle$ şi $\langle\beta,j\rangle$ cu $i\neq j$, fiecare termen din formula `F` este reprezentat în `V'` de către un singur nod din `V` şi, implicit, de către cel puţin un literal. De asemenea, pentru oricare două noduri $\langle\alpha,i\rangle$ şi $\langle\beta,j\rangle$ din `V'` avem $\alpha \neq \neg\beta$.

Să construim mulţimea literalilor $S = \{\alpha \mid \exists i\in 1..k \bullet \langle\alpha,i\rangle\in V'\}$. Mulţimea S respectă restricţia: pentru oricare literali distincţi α şi β din S există proprietatea $\alpha \neq \neg\beta$. Prin urmare, pentru orice variabilă `x` prezentă în literalii din nodurile `V'`şi, implicit, în mulţimea S există un singur literal α care conţine variabila `x`: fie $\alpha = x$, fie $\alpha = \neg x$.

Pentru fiecare variabilă `x` din formula `F` asociem o valoare de adevăr conformă regulilor:

- `x = 1`, dacă `x` face parte dintr-un literal $\alpha\in S$ şi literalul are forma $\alpha = x$. Toţi literalii din S care conţin variabila `x` au valoarea de adevăr `1`.
- `x = 0`, dacă `x` face parte dintr-un literal $\alpha\in S$ şi literalul are forma $\alpha = \neg x$,. Toţi literalii din S care conţin variabila `x` au valoarea de adevăr `1`.
- `x =` orice valoare din `{0,1}`, dacă `x` nu face parte din niciun literal din S.

Deoarece asocierile variabilă-valoare conforme regulilor de mai sus satisfac toţi literalii din S şi fiecare termen din `F` are cel puţin un literal în S, rezultă că `F` este satisfiabilă. ■

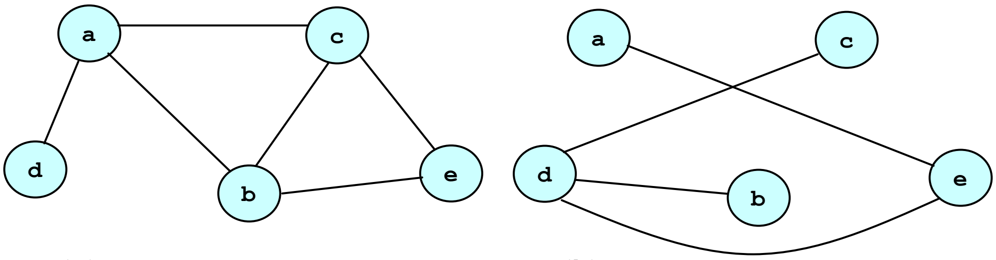
Din lema (C.3) şi din construcţia algoritmului $construcție\_G_F$ obţinem imediat $SAT \leq_p$ `k-Clică`. Deoarece `k-Clică` $\in$ `NP`, rezultă că problema este `NP-completă`. ■

Acoperirea unui graf neorientat

Definiţia C.21 Fie `G=(V,E)` un graf neorientat cu nodurile `V` şi arcele `E`, iar $V'\subseteq V$ o submulţime de noduri din `G`. `V'` este o `k-acoperire` a grafului `G` dacă:

- pentru orice arc $(u,v)\in E$ avem $\{u,v\} \cap V' \neq \emptyset$;
- `card(V') = k`.

O `k-acoperire` este minimă pentru `k` minim. În cazul grafului din figura C.11(b) o acoperire minimă este `{a,d}`. Să numim `k-Acoperire` problema decizională: fiind dat un graf neorientat `G` şi un întreg `k >0` să se determine dacă `G` are o `k-acoperire`.



(a) (b)

Figura C.11 Un graf `(a)` şi complementarul său `(b)`

Teorema C.9 Problema `k-Acoperire` este `NP-completă`.

a) Să arătăm că problema `k-Acoperire` este în `NP`, deci există un algoritm nedeterminist şi tractabil de rezolvare.

```
N_k-Acoperire(G,k) {
    fie V nodurile, iar E arcele grafului G;
    V' = ∅; // Nodurile k-acoperirii

    // Construcție k-acoperire potențială
    for(i=1; i ≤ k; i++) {
       u = choice(V);
       if(u ∈ V') fail;
       V' = V' ∪ {u};
    }

    // V' conține k noduri diferite din V. Formează o acoperire?
    for-each(u ∈ V)
        for-each(v ∈ V) if((u,v)∈E ∧ u∉V' ∧ v∉V') fail;
    success;
}
```

Este uşor de observat că algoritmul `N_k-Acoperire` are complexitatea $\Theta(k^2)+O(kn^2)$ pentru căutare liniară $u \in V'$, $u \notin V'$ şi $v \notin V'$.

b) Alegem o problemă `NP-completă`, fie ea `k-Clică`, şi demonstrăm reductibilitatea polinomială `k-Clică` $\leq_p$ `k-Acoperire`.

Definiția C.22 Fie $G=(V,E)$ un graf neorientat, iar $\overline{G}=(V,\overline{E})$ un graf cu arcele $\overline{E}=\{\forall u \in V, v \in V \mid (u,v) \notin E \bullet (u,v)\}$ (arcele unui graf complet cu nodurile V din care se elimină E). Graful $\overline{G}$ este complementarul lui G. În figura C.11 este reprezentat un graf şi complementarul său.

Reducem problema `k-Clică` pentru graful $G=(V,E)$ la problema `(n-k)-Acoperire` pentru graful $\overline{G}=(V,\overline{E})$, unde $n=card(V)$. Reducerea înseamnă în primul rând construcția grafului complementar.

```
construcție_Ḡ(G) {
    fie V nodurile grafului G;
    fie E arcele grafului G;
    Ē= ∅;

    for-each(u ∈ V)
       for-each(v ∈ V) if((u,v)∉E ∧ u≠v)  Ē= Ē∪{(u,v)};
    return (V,Ē);
}
```

Construcția este în timp polinomial în raport cu `n=card(V)`. Pentru a finaliza demonstrația `k-Clică` $\leq_p$`(n-k)-Acoperire` trebuie să arătăm că cele două probleme sunt echivalente.

Lema C.4 Fie un graf neorientat `G`. `G` are o `k-clică` dacă şi numai dacă $\overline{G}$ are o `(n-k)-acoperire`.

`G` are o `k-clică` ⇒ $\overline{G}$ are o `(n-k)-acoperire`.

Fie `V'` o `k-clică` a lui `G`. Să alegem la întâmplare un arc `(u,v)`$\in\overline{E}$. Atunci, conform definiției (C.22) avem `(u,v)`$\notin$`E`. Prin urmare `u` şi `v` nu pot să facă parte simultan din `k-clică`, pentru că altfel ar fi conectate prin arcul `(u,v)` care nu este în `E`. Înseamnă că cel puțin unul dintre nodurile `u` şi `v` face parte din `V-V'`. Cu alte cuvinte, arcul `(u,v)` este acoperit de cel puțin un nod din mulțimea `V-V'`. Pentru că `(u,v)` a fost ales la întâmplare, rezultă că orice arc din $\overline{E}$ este acoperit de cel puțin un nod din mulțimea `V-V'`, mulțime cu `n-k` noduri. Mulțimea `V-V'` este o `(n-k)-acoperire` a grafului $\overline{G}$.

$\overline{G}$ are o `(n-k)-acoperire` ⇒ `G` are o `k-clică`.

Fie `V'` o acoperire cu `(n-k)` noduri a grafului $\overline{G}$. Să alegem la întâmplare două noduri `u` şi `v` din `V-V'`. Arcul `(u,v)` nu poate fi în $\overline{E}$, pentru că nu este acoperit de `u` sau `v`. Prin urmare, `(u,v)`$\in$`E`. Rezultă că între oricare pereche de noduri din `V-V'` există un arc din `E`. Deci mulțimea `V-V'` formează în `G` o clică cu `k` noduri. ■

Deoarece `k-Acoperire` $\in$ `NP`, `k-Clică` $\leq_p$ `(n-k)-Acoperire`, conform lemei (C.4), şi `k-Clică` $\in$ `NP-complete` rezultă `k-Acoperire` $\in$ `NP-complete` ■

Problema 3SAT

Problema `3SAT` constă în verificarea satisfiabilității unei formule `CNF` în care fiecare termen are cel mult `3` literali. La fel ca şi `SAT` este folosită ca punct de plecare în verificarea `NP-durității` multor probleme.

Teorema C.11 $3SAT \in NP$-complete

1. $3SAT \in NP$. Algoritmul nedeterminist, cu complexitate polinomială, folosit pentru a verifica $SAT \in NP$ şi, implicit $3SAT \in NP$, este simplu şi este lăsat ca exerciţiu.

2. $3SAT \in NP$-dure. Să construim o reducere $SAT \leq_p 3SAT$.

Fie $F = T_1 \wedge T_2 \wedge \ldots \wedge T_k$ o formulă CNF cu n variabile. Construim transformarea $f(F) = R(T_1) \wedge R(T_2) \wedge \ldots \wedge R(T_k)$, astfel încât pentru un termen $T = \alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_q$ format cu literalii α_i, $i=1,q$, algoritmul $R(T)$ generează o formulă CNF de forma:

$$R(T) = (\neg x_1 \vee \alpha_1 \vee \alpha_2) \wedge (\neg x_2 \vee x_1 \vee \alpha_3) \wedge \ldots \wedge (\neg x_{i-1} \vee x_{i-2} \vee \alpha_i) \wedge \ldots \wedge (x_{q-3} \vee \alpha_{q-1} \vee \alpha_q)$$

Formula $R(T)$ are $q-2$ termeni cu câte cel mult 3 literali şi conţine, în afara literalilor α_i, $i=1,q$, variabilele suplimentare x_i, $i=1,q-C$.

```
R(T){ // T= α1 ∨ α2 ∨...∨ αq
   if(q ≤ 3) return T;
  generează o nouă variabilă x;
   return (¬x ∨ α1 ∨ α2) ∧ R(x ∨ α3 ∨ α4 ... αq);
}
```

De exemplu, pentru $T = \alpha_1 \vee \alpha_2 \vee \alpha_3 \vee \alpha_4 \vee \alpha_5$ avem:

$$\begin{aligned}
R(\alpha_1 \vee \alpha_2 \vee \alpha_3 \vee \alpha_4 \vee \alpha_5) \\
&= (\neg x_1 \vee \alpha_1 \vee \alpha_2) \wedge R(x_1 \vee \alpha_3 \vee \alpha_4 \vee \alpha_5) \\
&= (\neg x_1 \vee \alpha_1 \vee \alpha_2) \wedge (\neg x_2 \vee x_1 \vee \alpha_3) \wedge R(x_2 \vee \alpha_4 \vee \alpha_5) \\
&= (\neg x_1 \vee \alpha_1 \vee \alpha_2) \wedge (\neg x_2 \vee x_1 \vee \alpha_3) \wedge (x_2 \vee \alpha_4 \vee \alpha_5)
\end{aligned}$$

Notăm $c_f(n)$ complexitatea transformării $f(F)$, n fiind numărul de variabile ce pot să apară în F, şi $c_R(n)$ complexitatea transformării $R(T)$, unde T este un termen din F. Evident, $c_f(n) = c_R(n)^k$. Recurenţa de complexitate pentru algoritmul R este $c_R(m) = c_R(m-1) + \Theta(1)$, $4 \leq m \leq n$, cu condiţia la limită $c_R(3) = \Theta(1)$, iar complexitatea este $c_R(n) = O(n)$. Rezultă $c_f(n) = O(n^k)$. Deci transformarea f este polinomială.

Folosim notaţia T satisfiabil[α] $\Leftrightarrow$ $R(T)$ satisfiabil[α] pentru a spune că termenul T este satisfiabil dacă şi numai dacă formula $R(T)$ este satisfiabilă, pentru aceeaşi legare a variabilelor din literalii α_i, $i=1,q$.

T satisfiabil[α] $\Rightarrow$ $R(T)$ satisfiabil[α].

T satisfiabil[α] impune ca cel puţin un literal din termenul T să fie adevărat. Fie $\alpha_j=1$, $1 \leq j \leq q$ un asemenea literal. Atunci pentru a satisface $R(T)$ putem lega variabilele x_i următoarele valori: $x_i=0$, $1 \leq i < j-1$ şi $x_i=1$, $\max(1,j-1) \leq i \leq q-C$.

R(T) satisfiabil[α] ⇒ T satisfiabil[α].

Să presupunem există o legare a variabilelor x_i, $i=1,q-3$, din R(T) astfel încât R(T)=1, iar α_i=0, i=1,q. Atunci, putem elimina literalii α_i din R(T) fără a influența satisfiabilitatea formulei. Formula devine:

$$R'(T) = \neg x_1 \wedge (\neg x_2 \vee x_1) \wedge \ldots \wedge (\neg x_{i-1} \vee x_{i-2}) \wedge \ldots \wedge x_{q-3}$$

Dar R'(T) nu este satisfiabilă. Într-adevăr, parcurgând formula de la stânga spre dreapta sunt impuse următoarele legări ale variabilelor: x_1=0, x_2=0,..., x_{q-3}=0 şi x_{q-3}=1, ceea ce este imposibil. Prin urmare, trebuie să avem α_j=1 pentru cel puțin un indice $1 \leq j \leq q$. Înseamnă că T=1.

Din implicațiile de mai sus, rezultă proprietatea: **T satisfiabil[α] ⇔ R(T) satisfiabil[α]** şi, implicit, **F satisfiabil ⇔ f(F) satisfiabil**. Transformarea $f(F) = R(T_1) \wedge R(T_2) \wedge \ldots \wedge R(T_k)$ este într-adevăr o reducere SAT $\leq_p$ 3SAT, deci 3SAT∈NP-dure. Pentru că 3SAT∈NP şi 3SAT∈NP-dure. ■

Problema 2SAT

Problema **2SAT** constă în verificarea satisfiabilității unei formule **CNF** în care fiecare termen are cel mult 2 literali. Numim **2CNF** mulțimea unor asemenea formule. Spre deosebire de problema **NSat**, N≥3, care este **NP-completă**, **2SAT** este rezolvabilă determinist în timp polinomial, deci **2SAT∈P**. Ca demonstrație, vom arăta că rezolvarea problemei se reduce la deciderea existenței unor drumuri într-un graf.

Fie **F∈2CNF** o formulă. Notăm **Trm** mulțimea termenilor şi **Var** mulțimea variabilelor din F, **k=card(Trm)** şi **n=card(Var)**.

Să construim graful orientat G_F=(V,E) în felul următor:

$$V = \{\forall x \in Var \bullet \langle x \rangle\} \cup \{\forall x \in Var \bullet \langle \neg x \rangle\}$$

$$E = \bigcup_{T \in Trm} arce(T),$$

unde **arce(T)** este mulțimea arcelor construite pentru termenul **T** aşa cum se arată în tabelul (C.1).

Tabelul C.1 Arce corespunzătoare unui termen

termen T	formulă echivalentă pentru T	arce(T)
x	$(\neg x \Rightarrow x)$	$\{(\langle\neg x\rangle,\langle x\rangle)\}$
$\neg x$	$(x \Rightarrow \neg x)$	$\{(\langle x\rangle,\langle\neg x\rangle)\}$
$(x \vee y)$	$(\neg x \Rightarrow y) \wedge (\neg y \Rightarrow x)$	$\{(\langle\neg x\rangle,\langle y\rangle), (\langle\neg y\rangle,\langle x\rangle)\}$
$(\neg x \vee y)$	$(x \Rightarrow y) \wedge (\neg y \Rightarrow \neg x)$	$\{(\langle x\rangle,\langle y\rangle), (\langle\neg y\rangle,\langle\neg x\rangle)\}$
$(x \vee \neg y)$	$(\neg x \Rightarrow \neg y) \wedge (y \Rightarrow x)$	$\{(\langle\neg x\rangle,\langle\neg y\rangle), (\langle y\rangle,\langle x\rangle)\}$
$(\neg x \vee \neg y)$	$(x \Rightarrow \neg y) \wedge (y \Rightarrow \neg x)$	$\{(\langle x\rangle,\langle\neg y\rangle), (\langle y\rangle,\langle\neg x\rangle)\}$

Avem, $\mathtt{card(V)=2n}$, iar $\mathtt{card(E)\leq 2k}$ şi observăm că graful $\mathtt{G_F}$ poate fi construit în timp polinomial în raport cu $\mathtt{n}$ şi $\mathtt{k}$. Un exemplu este ilustrat în figura C.1.

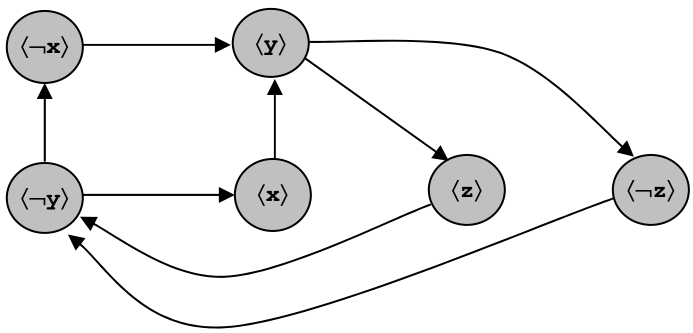


Figura C.1 Graful $\mathtt{G_F}$ corespunzător formulei $\mathtt{F=(x\vee y)\wedge(\neg x\vee y)\wedge(z\vee\neg y)\wedge(\neg z\vee\neg y)}$

Lema C.6 $\mathtt{F}$ nu este satisfiabilă $\Leftrightarrow$ $\exists \mathtt{x}\in\mathtt{Var}\ \bullet$ există drumuri $\langle\mathtt{x}\rangle..\langle\neg\mathtt{x}\rangle$ şi $\langle\neg\mathtt{x}\rangle..\langle\mathtt{x}\rangle$ în graful $\mathtt{G_F}$.

$\exists \mathtt{x}\in\mathtt{Var}\ \bullet$ există drumuri $\langle\mathtt{x}\rangle..\langle\neg\mathtt{x}\rangle$ şi $\langle\neg\mathtt{x}\rangle..\langle\mathtt{x}\rangle$ în $\mathtt{G_F} \Rightarrow$ $\mathtt{F}$ nu este satisfiabilă.

Fie $\mathtt{x}\in\mathtt{Var}$ astfel încât există un drum $\mathtt{d_1}=\langle\mathtt{x}\rangle..\langle\neg\mathtt{x}\rangle$ şi un drum $\mathtt{d_2}=\langle\neg\mathtt{x}\rangle..\langle\mathtt{x}\rangle$. Fie $(\langle\alpha\rangle,\langle\beta\rangle)$ şi $(\langle\beta\rangle,\langle\gamma\rangle)$, cu α, β şi γ literali, două arce consecutive dintr-un drum în $\mathtt{G_F}$. Arcele corespund implicaţiilor $\alpha\Rightarrow\beta$ şi $\beta\Rightarrow\gamma$, care trebuie să fie adevărate pentru ca $\mathtt{F}$ să poată fi satisfăcută. Folosind tautologia $\alpha\Rightarrow\beta\ \wedge\ \beta\Rightarrow\gamma\ \Rightarrow\ \alpha\Rightarrow\gamma$ rezultă că implicaţia $\alpha\Rightarrow\gamma$ trebuie să fie adevărată pentru ca $\mathtt{F}$ să poată fi satisfăcută. Implicaţia $\alpha\Rightarrow\gamma$ corespunde unui arc $(\langle\alpha\rangle,\langle\gamma\rangle)$, existent sau indus, în $\mathtt{G_F}$, arc ce poate substitui cele două arce consecutive. Prin inducţie, urmând căile $\mathtt{d_1}$ şi $\mathtt{d_2}$ obţinem:

- pentru $\mathtt{d_1}$: trebuie să avem $\mathtt{x}\Rightarrow\neg\mathtt{x}$ pentru ca $\mathtt{F}$ să poată fi satisfăcută.
- pentru $\mathtt{d_2}$: trebuie să avem $\neg\mathtt{x}\Rightarrow\mathtt{x}$ pentru ca $\mathtt{F}$ să poată fi satisfăcută.

Implicaţia $\mathtt{x}\Rightarrow\neg\mathtt{x}$ este adevărată pentru $\mathtt{x=0}$, iar Implicaţia $\neg\mathtt{x}\Rightarrow\mathtt{x}$ este adevărată pentru $\mathtt{x=1}$. Imposibil, pentru că variabila $\mathtt{x}$ trebuie legată la o valoare unică.

$\mathtt{F}$ nu este satisfiabilă $\Rightarrow \exists \mathtt{x}\in\mathtt{Var}\ \bullet$ există drumuri $\langle\mathtt{x}\rangle..\langle\neg\mathtt{x}\rangle$ şi $\langle\neg\mathtt{x}\rangle..\langle\mathtt{x}\rangle$ în $\mathtt{G_F}$

Dacă $\mathtt{F}$ nu este satisfiabilă, atunci există cel puţin o variabilă, fie ea $\mathtt{x}$, astfel încât $\mathtt{x}$ să nu poată fi legată la o unică valoare. Înseamnă că avem $\mathtt{x=1}$ şi $\mathtt{x=0}$ simultan, iar din formula $\mathtt{F}$ se pot deduce implicaţiile $\neg\mathtt{x}\Rightarrow\mathtt{x}$ şi $\mathtt{x}\Rightarrow\neg\mathtt{x}$. Cele două implicaţii corespund arcelor, existente sau induse, $(\langle\neg\mathtt{x}\rangle,\langle\mathtt{x}\rangle)$ şi $(\langle\mathtt{x}\rangle,\langle\neg\mathtt{x}\rangle)$ din $\mathtt{G_F}$. Înseamnă că în $\mathtt{G_F}$ există drumuri $\langle\mathtt{x}\rangle..\langle\neg\mathtt{x}\rangle$ şi $\langle\neg\mathtt{x}\rangle..\langle\mathtt{x}\rangle$. ■

Ca ilustrare a propoziţiei (C.1), formula $\mathtt{F}$ din figura 1 nu este satisfiabilă. Există drumurile: $\langle\neg\mathtt{z}\rangle,\langle\neg\mathtt{y}\rangle,\langle\neg\mathtt{x}\rangle,\langle\mathtt{y}\rangle,\langle\mathtt{z}\rangle$ şi $\langle\mathtt{z}\rangle,\langle\neg\mathtt{y}\rangle,\langle\neg\mathtt{x}\rangle,\langle\mathtt{y}\rangle,\langle\neg\mathtt{z}\rangle$ din care se deduc implicaţiile $\neg\mathtt{z}\Rightarrow\mathtt{z}$ şi $\mathtt{z}\Rightarrow\neg\mathtt{z}$, adică echivalenţa imposibilă $\mathtt{z}\Leftrightarrow\neg\mathtt{z}$.

Teorema C.11' **2SAT∈P**

Fie **Constr(F)=G_F** algoritmul determinist de construcție, în timp polinomial **p(n,k)**, a grafului **G_F** asociat unei formule **F** Putem construi următorul algoritm care rezolvă problema **2SAT**:

```
2SAT(F){
   GF= Constr(F); // complexitate polinomială în n şi k
   Var= variabile(F);
   for-each(x∈Var)
      if(drum(GF,⟨x⟩,⟨¬x⟩) ∧ drum(GF,⟨¬x⟩,⟨x⟩)) return 0;
   return 1;
}
```

Funcția **drum(G_F,u,v)** testează dacă în graful orientat **G_F** există un drum de la **u** la **v** şi are complexitate polinomială în raport cu numărul de noduri din graf, cel mult **2n**, **n=card(Var)**. Testul se poate efectua printr-o simplă parcurgere în adâncime a grafului **G_F** pornind de la **u**. În concluzie, algoritmul **2Sat** are complexitate polinomială în **n**, deci **2SAT∈P**. ■