Introduction to NP-Completeness

Chapter 11

2015

Motivatie

- Numerosi algoritmi au timpul de executie in cazul cel mai defavorabil de forma $O(n^k)$ unde n este dimensiunea intrarii si k este o constanta. De exemplu, algoritmul de sortare prin insertie are timpul de executie de forma $O(n^2)$.
- Intrebari normale:
  - Este adevarat ca toate problemele sunt rezolvabile in timp polinomial? Nu !
  - Este adevarat ca orice problema este rezolvabila algoritmic? Nu ! Spre exemplu celebra "*halting problem*" nu este rezolvabila algoritmic, adica este *nedecidabila*.
- *Halting problem*: Se considera o descriere a unui program $\mathcal{P}$ si o intrare a acestui program $\mathcal{T}$, ambele arbitrare. Sa se determine daca calculul determinat prin rularea lui $\mathcal{P}$ pentru intrarea $\mathcal{T}$ se termina sau se bucleaza infinit.

2015

Demonstratia "*halting problem*"

- Sa presupunem prin reducere la absurd ca exista un algoritm HALT(*P*,*I*) care determina daca *P* se bucleaza la infinit pentru *I*. HALT(*P*,*I*) va intoarce *adevarat* daca *P* se termina pentru *I*, altfel intoatce *fals*.
- Fie urmatorul program Z.
- Sa analizam ce se intampla pentru apelul Z(Z).
- Daca Z(Z) se termina atunci rezulta ca HALT(Z,Z) intoarce *fals*, deci Z nu se termina pentru intrarea Z, contradictie !
- Daca Z(Z) nu se termina atunci HALT(Z,Z) intoarce *adevarat*, deci Z se termina pentru intrarea Z, contradictie !

```
Z(X)
1. if HALT(X,X) then
2.     ▷ loop forever
3. else
4.         return
```


Problemele NP-complete

- Vom identifica si vom caracteriza clasa problemelor *NP-complete* – NPC. Starea lor dpdv al timpului de executie este necunoscuta, incerta:
  - Pentru nici o problema din NPC nu s-a gasit un algoritm polinomial de rezolvare
  - Pentru nici o problema din NPC nu s-a putut demonstra ca nu exista un algoritm polinomial de rezolvare
- Se poate insa demonstra ca problemele din NPC sunt in urmatorul sens "echivalente":

  Daca pentru o problema din NPC exista un algoritm polinomial de rezolvare, atunci pentru orice problema din NPC exista un algoritm polinomial de rezolvare.

Exemple de probleme NP-complete

- Cele mai scurte vs cele mai lungi drumuri in graf: Problema determinarii celor mai scurte cai intr-un graf este polinomiala (chiar si pentru costuri negative ale muchiilor), in timp ce problema celor mai lungi cai elementare intr-un graf este in NPC chiar si pentru cazul in care toate muchiile au costul 1.
- Problema continua vs problema discreta a rucsacului: Problema continua a rucsacului este polinomiala, iar varianta discreta este in NPC.
- Determinarea unui circuit Eulerian vs determinarea unui circuit Hamiltonian: determinarea unui circuit Eulerian este polinomiala in timp ce determinarea unui circuit Hamiltonian este in NPC.
- 2-SAT vs 3-SAT: *k*-SAT este problema determinarii daca o formula Booleana in forma normala *k*-conjunctiva poate fi satisfacuta. O astfel de forma este o conjunctie ($\wedge$) de disjunctii ($\vee$) (o disjunctie se numeste clauza) astfel incat fiecare disjunctie are exact *k* variabile (directe sau complementate). Posibilitatea satisfacerii inseamna existenta unei asignari de valori Boolene variabilelor formulei astfel incat rezultatul sa fie adevarat. 2-SAT este polinomiala in timp ce 3-SAT este in NPC.

Probleme de decizie

- Numeroase probleme sunt *probleme de optimizare*. O problema de optimizare cere determinarea unei solutii *x* dintr-o multime de solutii posibile *S* care sa minimizeze / maximizeze o functie de cost $c(x)$.
- De exemplu, problema discreta a rucsacului cere maximizarea profitului total al obiectelor incarcate in rucsac.
- Definirea problemelor NPC se aplica *problemelor de decizie*. O problema de decizie cere sa raspundem afirmativ sau negativ la intrebarea: exista o solutie $x \in S$ astfel incat costul sau sa fie cel mult / cel putin egal cu o valoare data *M* ?
- Observatie: Orice problema de optimizare P_{opt} se poate transforma intr-o problema de decizie P_{dec} a.i. P_{opt} *este cel putin la fel de grea* ca P_{dec}.
- Spre exemplu, fiind data o multime *I* de *n* obiecte, fiecare avand greutatea $w_i > 0$ si profitul $p_i > 0$, un numar natural *M* reprezentand capacitatea rucsacului si un numar natural *K* reprezentand o limita inferioara de profit, exista o submultime de obiecte $I' \subseteq I$ astfel incat:

  $\Sigma_{i \in I'} w_i \leq M$

  $\Sigma_{i \in I'} p_i \geq K$?

Algoritmi nedeterministi pentru probleme de decizie

- Un algoritm nedeterminist de rezolvare a unei probleme de decizie lucreaza in 2 etape:
  - *Etapa de ghicire*, in care se alege o solutie posibila
  - *Etapa de verificare*, in care se verifica daca solutia aleasa satisface conditiile problemei.
- Pentru a descrie un algoritm nedeterminist ce lucreaza dupa acest sablon, se introduc instructiunile:
  - *choice*(*M*) care intoarce un element arbirar din multimea *M*
  - *success* care semnaleaza terminarea cu succes a algoritmului
  - *failure* care semnaleaza terminarea cu esec a algoritmului
- Intuitiv, rularea unui algoritm nedeterminist reprezinta o *incercare* (engl. *trial*) de rezolvare a problemei.

Exemplu de algoritm nedeterminist

NON-DET-DISCRETE-KNAPSACK(n, v, p, M, K, x)

1. **for** $i \leftarrow 1,n$ **do**
2. $\quad x[i] \leftarrow$ **choice**$(\{0, 1\})$
3. $wChoice \leftarrow 0$
4. $pChoice \leftarrow 0$
5. **for** $i \leftarrow 1,n$ **do**
6. $\quad wChoice[i] \leftarrow wChoice[i] + w[i] \times x[i]$
7. $\quad pChoice[i] \leftarrow pChoice[i] + p[i] \times x[i]$
8. **if** $(wChoice \leq M) \wedge (pChoice \geq K)$ **then**
9. $\quad$ **success**
10. **else**
11. $\quad\quad$ **failure**

Clasele P si NP

- Definitie: Se noteaza cu:
  - P clasa problemelor *P* pentru care exista un algoritm determinist care rezolva problema *P* in timp polinomial.
  - NP clasa problemelor *P* pentru care exista un algoritm nedeterminist care rezolva problema *P* in timp polinomial.
- Observatie: Este evident ca $P \subseteq NP$, dar se crede ca $P \subset NP$.
- Teorema: Daca o problema *P* apartine clasei NP atunci exista polinoamele *p* si *q* astfel incat problema *P* poate fi rezolvata de un algoritm determinist in timp $O(p(n)\times 2^{q(n)})$.
- Demonstratie: Ghicirea unei solutii fiind in timp polinomial, dimensiunea unei solutii este un polinom $q(n)$. Exista un numar exponential $2^{q(n)}$ de alegeri posibile de solutii. Verificarea unei solutii dureaza un timp polinomial $p(n)$. Rezulta timpul total de generare si verificare de $O(p(n)\times 2^{q(n)})$.

Reducere

- Definitie: Fie *P* si *Q* doua probleme si o functie $g(n)$. Se spune ca *P* se poate transforma in timp $O(g(n))$ in *Q* daca exista o functie *t* astfel incat:
  - *t* transforma o instanta $p \in P$ intr-o instanta $t(p) \in Q$
  - *t* are complexitatea in timp $O(g(n))$
  - Pentru orice instanta $p \in P$, instantele *p* si $t(p)$ au acelasi raspuns (fiind probleme de decizie, valoare de adevar)
- Acest lucru se noteaza cu $P \propto_{g(n)} Q$ si se citeste "*P* se reduce in timp $g(n)$ la *Q*".
- Daca functia *g* este un polinom atunci se spune ca "*P* se reduce polinomial la *Q*" si se noteaza prin $P \propto Q$.
- Observatie: Daca $P \propto Q$ atunci putem interpreta intuitiv ca "*P* nu este mai greu de rezolvat decat *Q*".
- Propozitie: Relatia $\propto$ este *tranzitiva*, din $P \propto Q$ *si* $Q \propto R$ rezulta ca $P \propto R$.

Complexitate

- Teorema
  - Daca P are complexitatea $\Omega(f(n))$ si $P \propto_{g(n)} Q$ atunci Q are complexitatea timp $\Omega(f(n) - g(n))$
  - Daca Q are complexitatea $O(f(n))$ si $P \propto_{g(n)} Q$ atunci P are complexitatea timp $O(f(n) + g(n))$
- Corolar
  - Daca $P \notin$ P si $P \propto Q$ atunci $Q \notin$ P.
  - Daca $Q \in$ P si $P \propto Q$ atunci $P \in$ P.
- Pe baza corolarului se poate arata neapartenenta, respectiv apartenenta unei probleme la clasa P.

Probleme NP-complete si NP-dificle

- Intuitiv, pentru a arata ca P $\subset$ NP ar trebui sa gasim o problema in NP care nu este in P.
- Candidate vor fi acele probleme P cu proprietatea ca orice problema $Q \in NP$ avem $Q \propto P$, adica intuitiv orice problema din NP nu este mai greu de rezolvat decat P, cu alte cuvinte *P este foarte grea*.

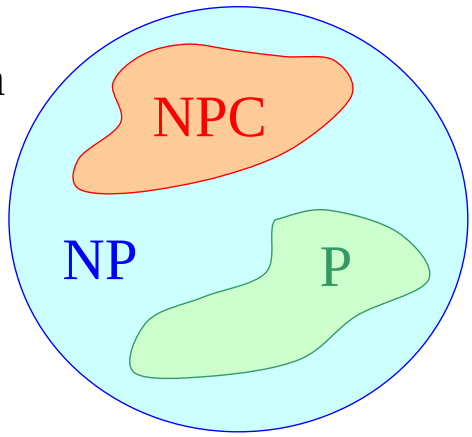
- Definitie
  - O problema P este *NP-dificila* – NPD daca pentru orice problema $Q \in NP$ avem $Q \propto P$.
  - O problema P este *NP-completa* – NPC daca $P \in$ NP $\cap$ NPD.

Problema satisfacerii – SAT

- Definitie: Fie X o multime de n variabile Booleene.
  - Un *literal* este o variabila x sau o variabila negata $\neg x$. O *clauza* este o disjunctie de literali si este reprezentata prin multimea literalilor disjunctiei.
  - O *asignare* este o functie $\alpha: X \rightarrow \{0,1\}$. Fie l un literal. Daca $l = x$ atunci $\alpha(l) = \alpha(x)$. Daca $l = \neg x$ atunci $\alpha(l) = \neg\alpha(x)$. Fie $c = \{l_1, \ldots, l_k\}$ o clauza ce reprezinta formula logica $l_1 \vee \ldots \vee l_k$. $\alpha(c) = 1$ daca si numai daca exista un literal $l \in c$ astfel incat $\alpha(l) = 1$.
  - O asignare α *satisface* o clauza c daca si numai daca $\alpha(c) = 1$.
- Definitie:
  - Fie $C = \{c_1, \ldots, c_m\}$ o multime de clauze. Ea reprezinta formula logica determinata de conjunctia clauzelor din C, adica $c_1 \wedge \ldots \wedge c_m$. Multimea C *poate fi satisfacuta* daca si numai daca exista o asignare care satisface toate clauzele din C.
  - *Problema satisfacerii* (engl. *Satisifiability*) – SAT pentru o multime de clauze C cere sa se determine daca multimea C poate fi satisfacuta.

Probleme NP-complete

- Teorema (*Cook*): SAT $\in$ NPC
- SAT este prima problema dovedita a fi in clasa NPC in 1971 !
- Numeroase probleme practice sunt in NPC, ceea ce inseamna ca practic nu exista un algoritm polinomial de rezolvare a lor.
- Observatie: Daca exista o problema $P \in$ NPC $\cap$ P atunci pentru orice problema $Q \in$ NP avem $Q \in$ P, adica NP = P !
- Deoarece $P \in$ NPC si $Q \in$ NP rezulta ca $Q \propto P$. Tinand cont ca $P \in$ P din tranzitivitatea relatiei $\propto$ rezulta ca $Q \in$ P.



Cum demonstram ca o problema este in clasa NPC?

Demonstratia NP-completitudinii prin reducere

- Propozitie: Fie $P \in$ NP si $Q \in$ NPC astfel incat $Q \propto P$. Atunci $P \in$ NPC.
- Observatie: Intuitiv, propozitia se bazeaza pe *reducerea* lui P la o problema cunoscuta Q din clasa NPC.
- Aceasta propozitie ne ofera urmatoarea procedura pentru a demonstra ca o problema P este in clasa NPC.
  - Se construieste un algoritm nedeterminist pentru a rezolva P. Rezulta ca $P \in$ NP.
  - Se porneste de la o problema Q despre care se stie ca este in NPC. Un exemplu este SAT.
  - Se construieste o reducere polinomiala a lui P la Q, de unde rezulta ca $P \propto Q$.
  - Conform propozitiei va rezulta ca $P \in$ NPC.

Problema 3-SAT

- Se considera o multime de clauze $C = \{c_1, \ldots, c_m\}$ construite folosind o multime finita U de variabile a.i. $|c_i| = 3$ pentru toti $1 \leq i \leq m$. Exista o asignare de valori de adevar variabilelor din U astfel incat fiecare clauza din C sa aiba valoarea adevarat?
- Un exemplu de problema 3-SAT este definita de clauzele $C = \{\{\neg x_1, x_4, x_5\}, \{x_2, \neg x_3, x_5\}, \{\neg x_1, x_2, \neg x_4\}, \{x_1, \neg x_3, \neg x_6\}\}$
- Demonstratia se realizeaza prin reducere. Se arata ca orice problema SAT definita de o multime U de variabile si o multime C de clauze se poate transforma intr-o problema 3-SAT definita de o multime U' de variabile si o multime C' de clauze formate din exact 3 literali astfel incat C poate fi satisfacuta dnd C' poate fi satisfacuta. Rezulta astfel ca SAT $\propto$ 3-SAT. Dar SAT $\in$ NPC, asa ca rezulta 3-SAT $\in$ NPC.

Reducerea SAT la 3-SAT

- Reducerea va inlocui fiecare $c_j \in C$ cu o multime de clauze C_j' construite cu U si o multime aditionala de variabile U_j', definita in continuare. Putem astfel defini problema 3-SAT la care se reduce problema SAT initiala:
  $C' = \cup_{1 \le j \le m} C_j'$
  $U' = U \cup (\cup_{1 \le j \le m} U_j')$
- Fie $c_j = \{z_1, \ldots, z_k\}$. In functie de valoarea lui k consideram 4 cazuri:
- k=1. $U_j' = \{y_j^1, y_j^2\}$. $C_j' = \{\{z_1, y_j^1, y_j^2\}, \{z_1, y_j^1, \neg y_j^2\}, \{z_1, \neg y_j^1, y_j^2\}, \{z_1, \neg y_j^1, \neg y_j^2\}\}$
- k=2. $U_j' = \{y_j^1\}$. $C_j' = \{\{z_1, z_2, y_j^1\}, \{z_1, z_2, \neg y_j^1\}\}$
- k=3. Nu e nevoie de transformare, deci $U_j' = \varnothing$ si $C_j' = \{c_j\}$
- $k \ge 4$. $U_j' = \{y_j^i \mid 1 \le i \le k\text{-}3\}$. $C_j' = \{\{z_1, z_2, y_j^1\}\} \cup \{\{\neg y_j^i, z_{i+2}, y_j^{i+1}\} \mid 1 \le i \le k\text{-}4\} \cup \{\{\neg y_j^{k-3}, z_{k-1}, z_k\}\}$. C_j' are k-2 elemente.
- Lasam ca exercitiu verificarea faptului ca C poate fi satisfacuta dnd C' poate fi satisfacuta.

2015

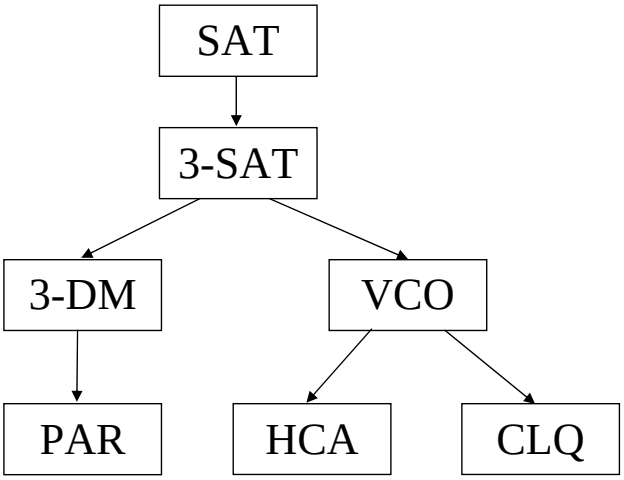
Problema 2-SAT

- Aparent paradoxal, problema 2-SAT este in clasa P !
- Se prezinta succint algoritmul lui Krom din 1967.
- Daca fiecare variabila x apare in clauze fie directa, fie complementata (dar nu ambele!), atunci se alege o asignare a lui x cu 1 sau 0, dupa cum aparitia lui x in clauze este directa sau complementata. Trivial rezulta ca multimea de clauze poate fi satisfacuta. De ce ?
- Sa presupunem ca exista o variabila x pentru care exista doua clauze $C_1 = \{a, x\}$ si $C_2 = \{b, \neg x\}$ ce contin x si $\neg x$. Se inlocuiesc cele doua clauze cu clauza $C_3 = \{a,b\}$. Multimea initiala de clauze C poate fi satisfacuta dnd multimea rezultata $C' = (C \setminus \{C_1, C_2\}) \cup \{C_3\}$ poate fi satisfacuta.
- Se repeta acest pas pana cand fie multimea de clauze contine doua clauze $\{x,x\}$ si $\{\neg x, \neg x\}$, caz in care multimea initiala de clauze nu poate fi satisfacuta, fie fiecare variabila x apare in clauzele rezultat directa sau complementata (dar nu ambele!), caz in care multimea initiala de clauze poate fi satisfacuta.
- Tema: Descrieti algoritmul lui Krom in pseudocod si implementati-l in C.

2015

Alte probleme din NPC

- Problema identificarii 3-dimensionale (engl. 3 *Dimensional Matching*) – 3-DM.
- Problema acoperirii varfurilor unui graf (engl. *Vertex Cover*) – VCO.
- Problema clicii (engl. *Clique*) – CLQ.
- Problema circuitului Hamiltonian (engl. *Hamiltonian Circuit*) – HAC.
- Problema partitiei (engl. *Partition*) – PAR.

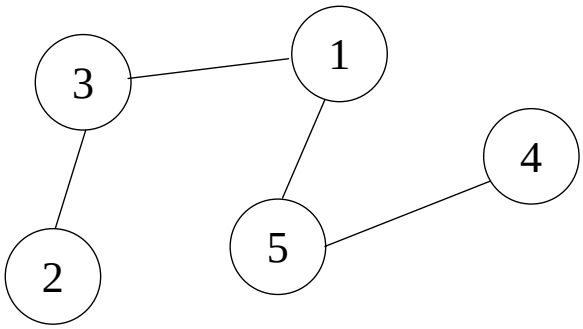


Problema identificarii 3-dimensionale – 3-DM

- Fie 3 multimi disjuncte W, X si Y avand fiecare acelasi numar q de elemente si fie $M \subseteq W \times X \times Y$ o submultime a produsului cartezian $W \times X \times Y$.
- Se numeste *identificare* (engl. *matching*) o submultime $N \subseteq M$ astfel incat pentru orice doua triplete distincte $(w_1,x_1,y_1) \in M$' si $(w_2,x_2,y_2) \in M$' avem $w_1 \neq w_2$, $x_1 \neq x_2$, si $y_1 \neq y_2$.
- Problema 3-DM cere sa se determine daca multimea M contine o identificare N astfel incat $|N| = q$.
- Spre exemplu, daca $W = \{a,b,c\}$, $X = \{1,2,3\}$, $Y = \{4,5,6\}$ si $M = \{(\underline{a,1,4}), (a,2,5), (b,2,4), (\underline{c,3,5}), (a,3,6), (\underline{b,2,6})\}$, o solutie este determinata de elementele subliniate.

Problema acoperirii varfurilor unui graf – VCO

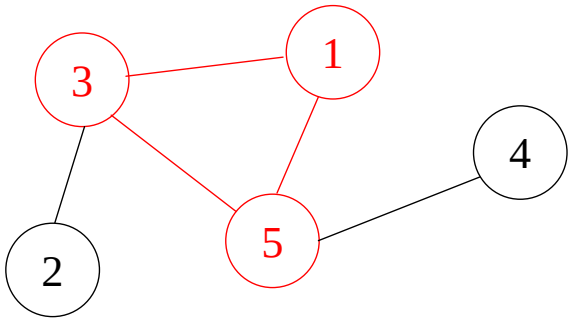
- Se considera un graf $G = \langle V,E \rangle$ si fie un numar natural $K \leq |V|$.
- Se numeste *acoperire* (engl. *cover*) a lui G o submultime $V' \subseteq V$ a varfurilor lui G astfel incat pentru orice muchie / arc $(u,v) \in E$ cel putin unul dintre varfurile u si v se afla in V'.
- Se cere sa se determine daca exista o acoperire V' a lui G de dimensiune cel mult K, adica $|V'| \leq K$.
- Spre exemplu, fie graful G definit de $V = \{1, 2, 3, 4, 5\}$ si $E = \{(1,5), (1,3), (2,3), (4,5)\}$.



- Fie $K = 2$. Oo acoperire care satisface conditiile este $V' = \{3, 5\}$.
- Daca $K = 1$ nu exista nici o acoperire care sa satisfaca conditiile din enunt.

Problema clicii – CLQ

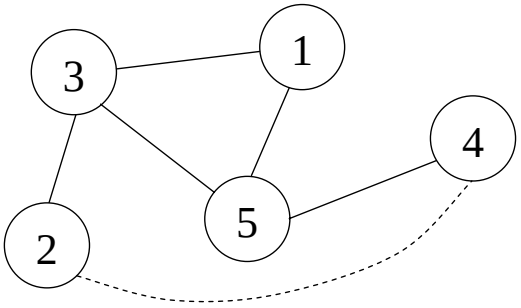
- Se considera un graf $G = \langle V,E \rangle$ si fie un numar natural $J \leq |V|$.
- Se numeste *clica* (engl. *clique*) un subgraf complet C al lui G. Numarul varfurilor lui C se numeste *dimensiunea* lui C.
- Este adevarat ca G contine o clica de dimensiune cel putin J ? Aceasta proprietate este echivalenta cu faptul ca exista o submultime $W \subseteq V$ astfel incat:
  - $J \leq |V|$
  - Pentru orice $u,v \in W$ avem $(u,v) \in E$.
- Spre exemplu, graful urmator contine o clica de dimensiune 3, dar nu contine clici de dimensiune 4.



- In acest caz problema CLQ va avea un raspuns afirmativ pentru $J = 2$ si $J = 3$, dar va avea un raspuns negativ pentru $J = 4$.

Problema circuitului Hamiltonian – HAC

- Se considera un graf $G = \langle V,E \rangle$ astfel incat $|V| = n$.
- Se numeste *circuit Hamiltonian* al lui G o ordonare a varfurilor sale $(v_1, v_2, \ldots, v_n)$ astfel incat $(v_n, v_1) \in E$ si $(v_i, v_{i+1}) \in E$ pentru orice $1 \leq i < n$.
- Spre exemplu, sa consideram urmatorul graf, considerand doar muchiile desenate cu linie plina. Acest graf nu contine un circuit Hamiltonian.



- Daca insa se adauga si muchia (2,4) desenata cu linie punctata, graful va contine urmatorul circuit Hamiltonian: (3, 2, 4, 5, 1).

Problema partitiei – PAR

- Se considera o multime finita A astfel incat fiecare element $i \in A$ are ponderea egala cu un numarul natural nenul w_i.
- Exista o submultime $B \subseteq A$ astfel incat:

  $\Sigma_{i \in A} = \Sigma_{i \in B \setminus A}$
- Notand cu $C = B \setminus A$, perechea de multimi (B,C) este o *partitie binara* a lui A. Problema cere sa determinam daca exista o partitie binara a lui A astfel incat ponderile cele doua multimi ale partitiei sa fie egale.
- Spre exemplu, considerand o multime cu 5 elemente $A = \{1, 2, 3, 4, 5\}$ si vectorul de ponderi $w = (1, 4, 2, 7, 2)$, o partitie cu proprietatea din enunt va fi definita de $B = \{2, 3, 5\}$.

Bibliografie

- Michael R. Garey, David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Series of Books in the Mathematical Sciences, W. H. Freeman, 1979