Examen

Tuesday, June 14, 2016 11:30 AM

- Remember functii O, Ω
- Ω caracterizeaza problema, limita inferioara
- O caracterizeaza algoritmul, limita
superioara
- Ω = O caz defavorabil=> algoritm optimal
- Cresterea dimensiunii maxime a problemei
rezolvabile cu cresterea vitezei masinii:
- n2= k*n1 O(nk)
- n2= k+n1 O(2n) !si k are valuare FOARTE mica (lgn)!

O problema de optim poate fi transformata intro
problema de decizie

Pentru a aplica teoria NP-completitudinii,
problema de optim trebuie transformata in
una de decizie
Metodologie: impunerea unei limite valorii de
optimizat (inf pt. max ^ sup pt min)
- Ex: Drum-Minim devine Drum prin impunerea unui
k maxim!

NP-completitudine

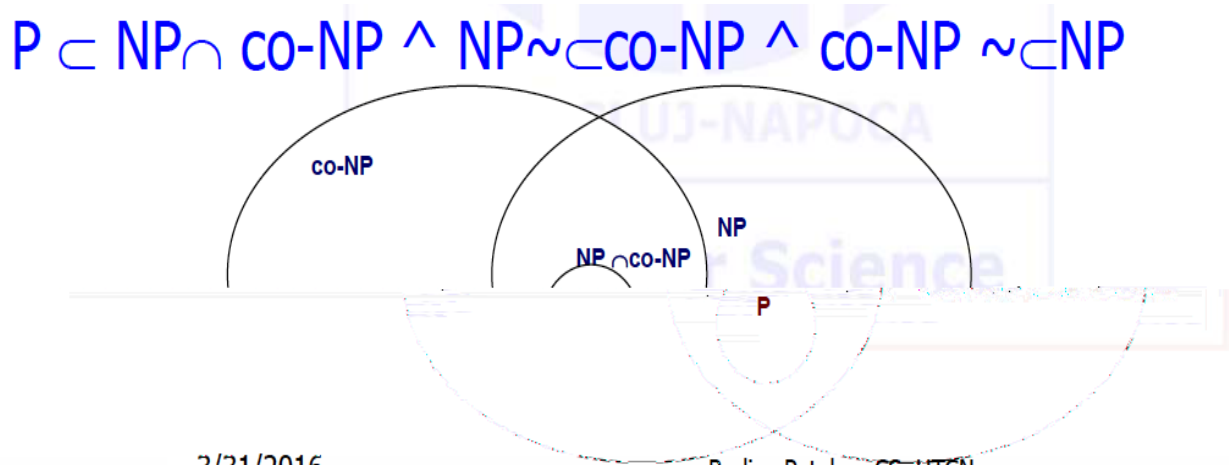
Restrictia doar la rezolvarea problemelor de
decizie NU reprezinta o limitare

∈ = apartine

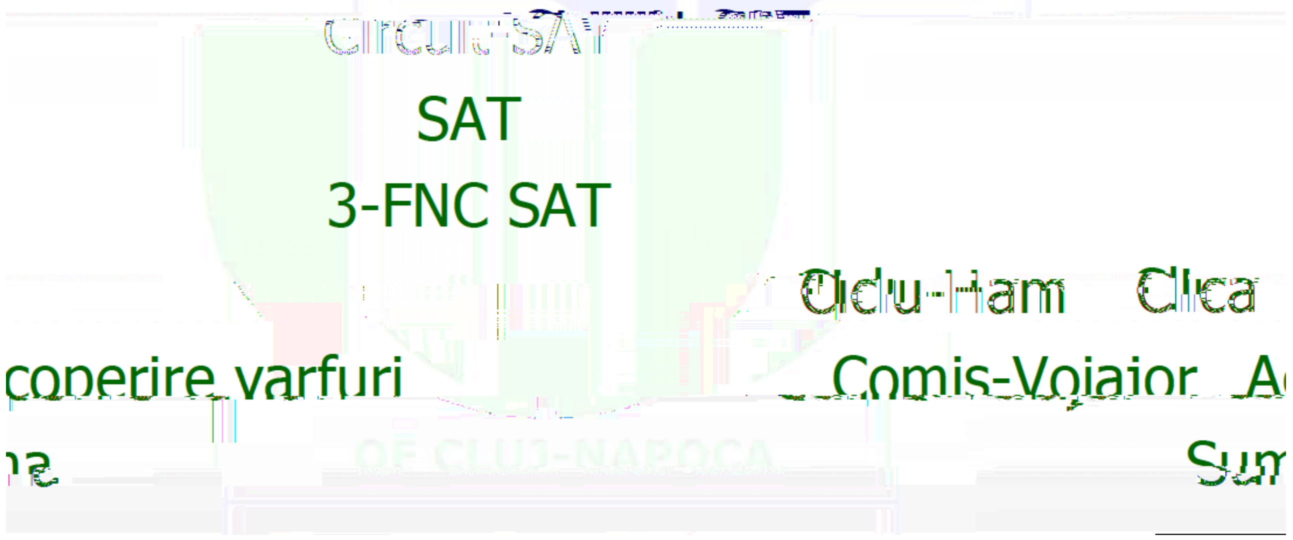
Problema Optim ∈ P
=> Problema Decizie ∈ P
Problema Decizie ∈ NP
=> Problema Optim ∈ NP
?Daca Problema Decizie ∈P? =>?
Problema Optim ∈ P
Nu putem afirma
Problema Optim ∈ NP
Nu putem afirma

pentru probleme pentru care NU se cunosc algoritmi poli de clarificare (decizie) (i.e. nu pot decide in timp polinomial = NP) se pot construi algoritmi poli de verificare

Deci pt o problema NP, versiunea sa de verificare devine P.



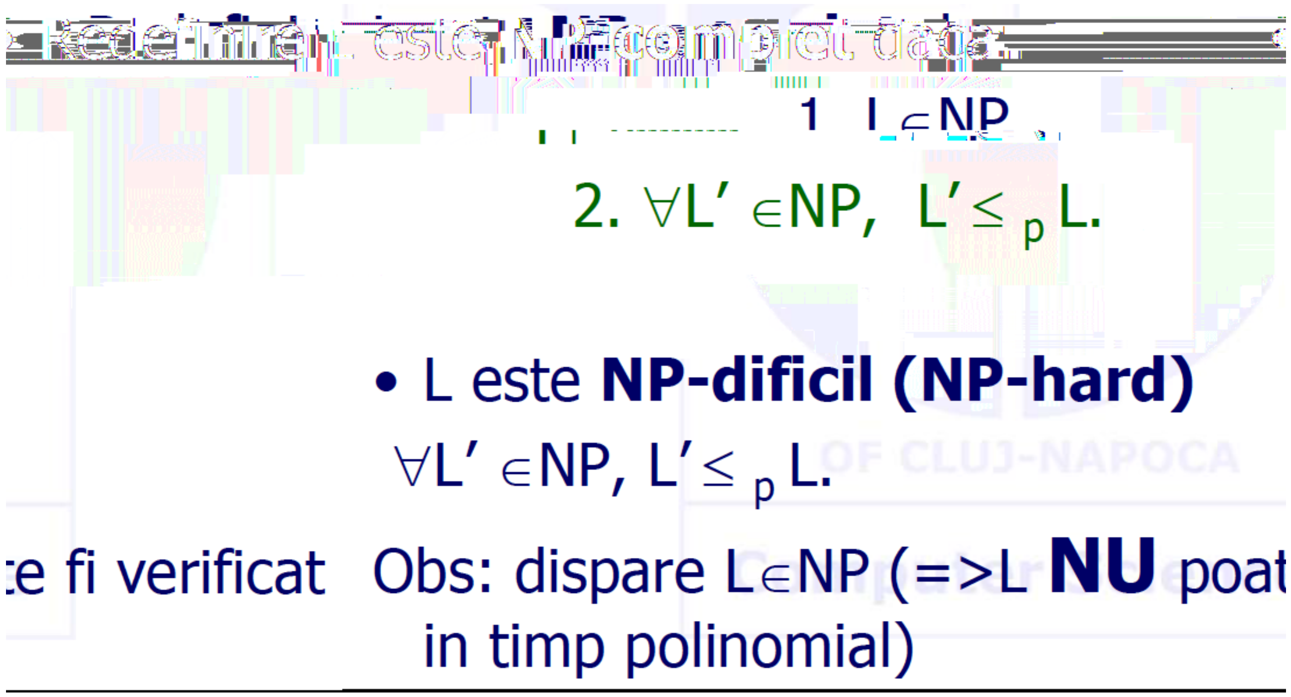
Reductibilitate - Rezolvarea unei probleme daca se stie rezolva o alta problema



• Circuit-SAT = circuit combinational logic, doar cu porti ^, V, ~, si iesire unica. Q: e satisfiabil?
• SAT=formula logica (^, V, ~, ->, <->). Q: are o combinatie a variabilelor a.i. valoarea logica=T?
• 3-FNC-SAT (forma normal conjuctiva)= constrangere a SAT a.i. apar doar 3 variabile, si in plus e o conjunctie de disjunctii

disjunctii.
• Clica= cel mai mare subgraf complet al unui graf
• Acoperire varfuri= submultime minimala V' a lui V, a.i. oricare ar fi(x,y) din E, x sau y e din V'
• Suma=submultime minimala dintr-o multime data de numere a.i. suma lor = tinta (varianta 0)
Circuit-SAT <=P SAT <=P 3-FNC-SAT <=P
<=P Clica <=P Acoperire Varfuri <=P Suma

Circuit-SAT <=P SAT <=P 3-FNC-SAT <=P
<=P Ciclu-Ham <=P Comis-Voiajor



• Redefinire: L este NP-complet daca:

1. $L \in NP$,

2. $\forall L' \in NP,\ L' \leq_p L.$

• L este **NP-dificil (NP-hard)**

$\forall L' \in NP,\ L' \leq_p L.$

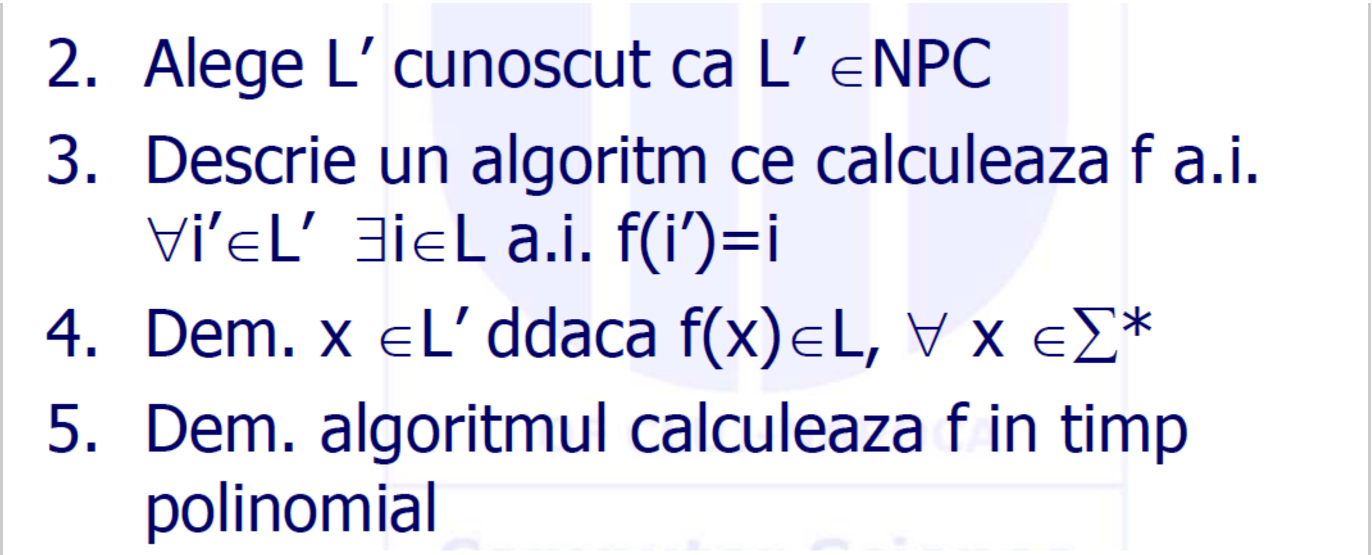
Obs: dispare $L \in NP$ (=>L **NU** poate fi verificat in timp polinomial)

NPC (informal) = poate fi verificata in timp polinomial, dar nu poate fi rezolvata (decisa) in timp polinomial

Lema2: Circuit-SAT $\in$ NP
Lema3: Circuit-SAT $\in$ NP - dificila
Lema4: Circuit-SAT $\in$ NP-completa
Lema5: Daca L' $\leq_p$ L, cu L' $\in$ NPC, atunci L $\in$ NP-hard. Daca in plus L $\in$ NP, atunci L' $\in$ NPC.

Metodologie pentru demonstratia

1. $L \in NP$



2. Alege L′ cunoscut ca L′ $\in$NPC
3. Descrie un algoritm ce calculeaza f a.i. $\forall i' \in L' \ \exists i \in L$ a.i. $f(i')=i$
4. Dem. $x \in L'$ ddaca $f(x) \in L$, $\forall\ x \in \Sigma^*$
5. Dem. algoritmul calculeaza f in timp polinomial

NPC a majoritatii problemelor se poate dem
prin reducerea SAT la acele probleme

Curs 4

SUMA - Algoritm reducere = in esenta reprezentarea
binara a grafului (i.e. matrice de incidenta)

Ciclul hamiltonian- reducerea
G poate fi construit in timp polinomial:
• phi contine k propozitii si max 3k literali
• Vom avea k grafuri B si max 3k grafuri A
• G va avea O(k) muchii si varfuri
• Constructia lor se realizeaza in timp polinomial
raportat la dimensiunea lui phi

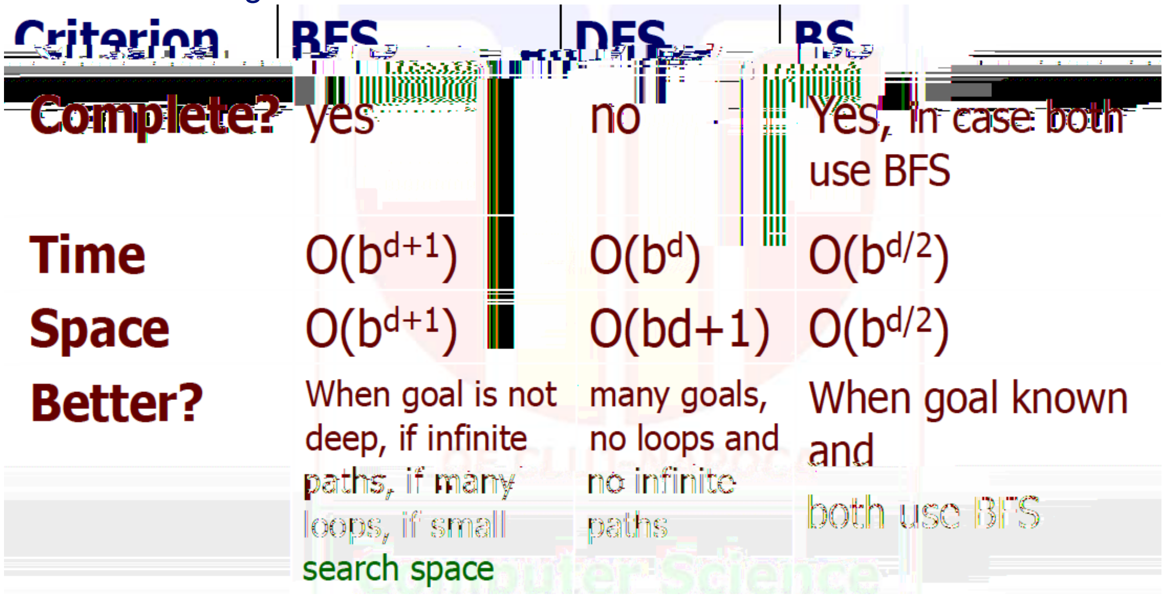
Comis Voiajor - un tur (ciclu ham) vizitand fiecare
oras exact o data, terminand de unde a
inceput
PCV => Ciclu Ham
• Pres G’ are un ciclu h’ de cost 0
• Consideram acel ciclu, care este ciclu ham in
acelasi timp (muchiile de cost 0 in h’ sunt cele care apartin E, deci
sunt in G)

Curs 5
Pt acoprerire varfuri, daca gradul varfurilor e

max 3, (inseamna ca fiecare nou varf va
acoperi max 3 arce noi, inseamna ca dim
multimilor in var acoperire multime max 3)
Suma Aprox (S,t,epsilon) e o schema
de aproximare complet polinomiala pt
problema sumei iar valoarea returnata este
de cel mult (1- epsilon) ori mai mica decat solutia
optimala

Curs 6
Search techniques
b=branching factor
d=depth of the search

BFS - asociata o coada FIFO (in latime)
Root is first expanded, all its successors
The search is complete – if there is at least one
goal node no deeper than d
Guarantees to find a solution
DFS - in adancime
Starts with the root which is first expanded
Next, the current node (deepest node reached so far) is expanded
asociata o coada LIFO
BS - bidirectional search
Run two simultaneous searches
• One from the initial state
• One from the goal



Criterion	BFS	DFS	BS
Complete?	yes	no	Yes, in case both use BFS
Time	$O(b^{d+1})$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O(bd+1)$	$O(b^{d/2})$
Better?	When goal is not deep, if infinite paths, if many loops, if small search space	many goals, no loops and no infinite paths	When goal known and both use BFS

Programming techniques

no optimal solution- they report that
cautare sistematica dupa solutii

Generate and test - brute force
ia in considerare toate posibilitatile si le verifica daca sunt bune, toate combinatiile variabilelor
foarte incet

Backtracking BT
verificare in pasi a validitatii coditiilor
BT enuma posibilitatile reale pentru variabile, in combinatie cu GT
Gaseste solutie
exponential
util cand stim ca trebuie sa existe o solutie

Divide et impera
rezolvarea a 2 sau mai multe probleme simultan
divide-rupere problema
conquer- rezolvare problema
combine - combina rezultatele

Curs 7 - pana la 38
Fara Tabu si Genetic

Greedy
Indicated when an ordinary solution is required
(although sometimes finds optimal solutions
nu asigura optimalitatea
utilizat pentru aproximari
pentru probl NPC - aproximari bune
search for global optimum based on local optimum
it is worth (if affortable) sacrificing optimality in
favor of reducing overall complexity

Dynamic programming
used to solve optimization problems
utilizabil cand problema e recurenta
impartire in subprobleme rezolvabile si fiecare rezolvare e optimala
reutilizare in loc de recalculare
utilizat cand problema nu are solutie cu greedy

Branch and Bound
Can be viewed as an “intelligent” (or informed) version of
bfs in the search space
Changes the searching through the entire search space by
adding some criteria, according to which the complexity of
the BFS can be reduced

Turing

a Turing machine can be adapted to simulate the logic of any computer algorithm

• is useful to explain the functions of a CPU

• Turing machines capture the informal notion of effective method in logic and mathematics, and provide a precise definition of an algorithm or “mechanical procedure”

A Turing machine M consists of:

• Tape - divided into cells; each cell contains a symbol from some finite alphabet; ⊔ a special blank symbol; for some models the tape has a left end marked with a special symbol; the tape extends or is indefinitely extensible to the right (in some models in both or none direction).

• Head - read and write symbols on the tape and move the tape left and right one (and only one) cell at a time

• Table - action table, or transition function (relation for nodeterm TM)

• State register - stores the state of the table; one special start state with which the state register is initialized.

halting states: accept or reject

A Turing machine M is defined by a 4-tuple (Q,Σ,δ,q0)

• Q finite set of states (K in textbook, S in others)

• Σ finite set of symbols (input alphabet of TM over a finite alphabet; should contain some special symbols: blank and first symbol)

• q0 initial state ∈ Q

• δ the transition function (represents the “program” of the machine)

• Turing-decidable language =recursive language

A language L is decided by M,if on every w, the TM finishes in a halting configuration

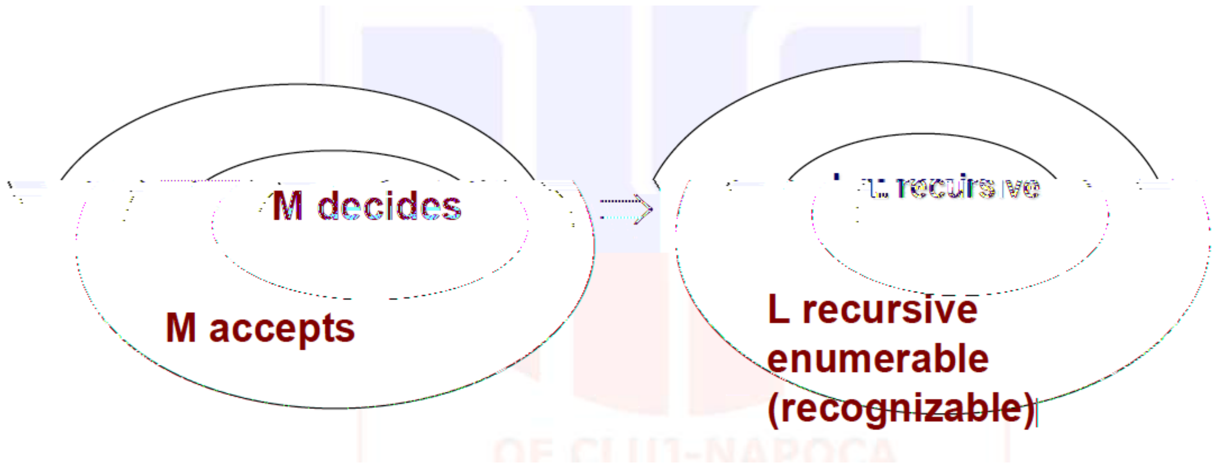
A language L is Turing-decidable if and only if there is a TM M that decides L.

A language L is accepted by M if on every w, if either M finishes in the accepting state or it never stops

• A language L is Turing-acceptable if and

only if there is a TM M that accepts L.
• Turing-acceptable language=recursively enumerable language

Every Turing decidable is Turing acceptable.



M is an N for which the relation drond is restricted to a function.

The Halting Problem
• U leads to undecidable problems
• Undecidable =
problems that have no algorithms
languages that are not recursive
• Identifications
Problems – Languages
Algorithms – Turing Machines
• Hence, instead of discussing about problems and algorithms that solve the given problems, we deal with TM that decides (or even accepts/recognize) language(s).

dovedirea undecidability
    reducere la absurd

5 minute de interogat masina si judecatorul nu isi poate da seama ca e masina