La classe NP, problemi NP-completi e la congettura P≠NP

Alberto Marchetti Spaccamela
U. Roma La Sapienza
A.A.2018-2019

SAT

Data una formula F del calcolo proposizonale decidere se è soddisfacibile

Data un'assegnazione di valori di verità possiamo decidere se la formula è vera. Costo : polinomiale

Pertanto per decidere se una formula F del calcolo proposizonale decidere se è soddisfacibile possiamo provare tutte le possibili di assegnazioni di valori di verità e verificare se uno soddisfa

- Se SI ➔ la formula è soddisfacibile
- Se nessun valore soddisfa ➔ la formula NON è soddisfacibile

Costo di questo algoritmo?
Costo polinomiale p(n) per ogni possibile assegnazione di valori
Il numero di possibili assegnazioni di una formula con n variabili è 2^n
Costo totale algoritmo ($p(n)\ 2^n$) esponenziale

Macchina di Turing non deterministiche

In una macchina di Turing deterministica dato uno stato q e un simbolo s letto dalla testina, f(q,s)=(q', s', m) determina la transizione da eseguire ed è univocamente determinata.

Una macchina di Turing non deterministica è una macchina di Turing a cui non viene imposto il vincolo che, dato uno stato della macchina e un simbolo letto dalla testina, la transizione da eseguire sia univocamente determinata.

Quindi, f(q,s) non è una funzione ma per un dato q e s possiamo avere più valori.

La macchina termina con successo se una scelta porta a stato finale

Macchina di Turing non deterministiche

Una macchina di Turing non deterministica è una macchina di Turing a cui non viene imposto il vincolo che, dato uno stato della macchina e un simbolo letto dalla testina, la transizione da eseguire sia univocamente determinata.

La computazione di una macchina di Turing T non è più rappresentabile mediante una sequenza di configurazioni.

Tuttavia, essa può essere vista come un albero, detto **albero delle computazioni**, i cui nodi corrispondono alle configurazioni di T e i cui archi corrispondono alla produzione di una configurazione da parte di un'altra configurazione.

Macchina di Turing non deterministiche

La computazione di MdT può essere vista come un albero, detto **albero delle computazioni**, i cui nodi corrispondono alle configurazioni di T e i cui archi corrispondono alla produzione di una configurazione da parte di un'altra configurazione.

Un input x è accettato da una macchina di Turing non deterministica T se l'albero delle computazioni corrispondente a x include almeno un cammino di computazione accettante, ovvero un cammino di computazione che termini in una configurazione finale.

L'insieme delle stringhe accettate da T è detto essere il linguaggio accettato da T ed è indicato con L(T).

La classe NP

La classe P è l'insieme di tutti i linguaggi L per cui esiste una Macchina di Turing deterministica che in tempo polinomiale decide se x appartiene o meno a L

La classe NP è l'insieme di tutti i linguaggi L per cui esiste una macchina di Turing nondeterministica che in tempo polinomiale decide se x appartiene o meno a L (cioè nell'albero delle computazioni esiste almeno un percorso dalla radice ad un noo che termina e accetta)

La classe P è inclusa nella classe NP (una macchina di Turing deterministica è un caso particilare di una macchina di Turing non deterministica)

Esempio: Macchina di Turing non deterministica che risolve SAT

Nota: Il nome NP deriva da **N**ondeterministic **P**olynomial time.

La classe NP

Definizione alternativa

La classe NP è l'insieme di tutti i linguaggi L per cui - se x appartiene a L – allora esiste un certificato c(x) tale che

- c(x) ha lunghezza polinomiale in |x|
- Esiste una Macchina di Turing deterministica che con input x e c(x) verifica che x appartiene a L in tempo polinomiale in |x| e |c(x)|(lunghezza di x e di c(x))

Nota: non si richiede un certificato quando x NON appartiene a L

Nota: le due definizioni sono equivalenti: possiamo vedere le decisioni nondeterministiche della macchina nondeterm. (I def.) come il certificato (usato nella definizione II) che "informa" la MdT deterministica che verifica la soluzione

Ma per quale c… di motivo devo imparare due definizioni equivalenti??

La classe NP

Nota: le due definizioni sono equivalenti: possiamo vedere le decisioni nondeterministiche della macchina nondeterm. (I def.) come il certificato (usato nella definizione II) che "informa" la MdT deterministica che verifica la soluzione

La prima definizione è diretta facendo riferimento ad una macchina di Turing (o un algoritmo non deterministico)

La seconda definizione evidenzia il fatto che per i problemi nella classe NP la verifica di una soluzione è un compito facile (cioè polinomiale). Il problema di trovare la soluzione o dimostrare che una soluzione non esiste è un compito più complicato.

La classe NP

La classe P è l'insieme di tutti i linguaggi L per cui esiste un algoritmo che in tempo polinomiale decide se x appartiene o meno a L

La classe NP è l'insieme di tutti i linguaggi L per cui - se x appartiene a L – allora esiste un certificato c(x)
tale che

- c(x) ha lunghezza polinomiale in |x|
- c(x) certifica che x appartiene a L

Nota: non si richiede un certificato che certifichi quando x NON appartiene a L

La classe P è inclusa nella classe NP (anche con la seconda definizione)
Ovviamente se L appartiene a P abbiamo un certificato
Infatti la sequenza di tutte le istruzioni dell'algoritmo polinomiale che verifica se x appartiene o no a L certifica

- *Se x appartiene a L*
- *Se x non appartiene a L*

La classe NP

Formalmente

La classe NP è l'insieme di tutti i linguaggi L per cui esiste una funzione c(x) e un polinomio p per cui x se x appartiene a L allora

- |c(x)| (lunghezza di c(x)) verifica $|c(x)| \leq p(|x|)$ (cioè la lunghezza di c(x) è polinomiale nella lunghezza di x)
- Esiste una macchina di Turing V che con ingresso x e |c(x)| certifica che x appartiene a L (terminando in uno stato finale) e ha complessità temporale polinomiale in |x| e |c(x)| (lunghezza di x e di c(x))

In molti casi il certificato rappresenta la soluzione del problema stesso

Esempi di certificato (problema :: certificato)

- SAT :: assegnazione di valori Vero / Falso alle variabili
- Dato un grafo esiste un percorso da u a v lungo al massimo h archi :: sequenza di k archi ($k \leq h$) che collega u a v)
- Dato un grafo esiste una cricca (insieme di nodi reciprocamente collegati) di k nodi :: insieme di k nodi mutuamente collegati

La classe NP

Formalmente

La classe NP è l'insieme di tutti i linguaggi L per cui esiste una funzione c(x) e un polinomio p per cui x se x appartiene a L allora

- |c(x)| (lunghezza di c(x)) verifica $|c(x)| \leq p(|x|)$ (cioè la lunghezza di c(x) è polinomiale nella lunghezza di x)
- Esiste una macchina di Turing V che con ingresso x e |c(x)| certifica che x appartiene a L (terminando in uno stato finale) e ha complessità temporale polinomiale in |x| e |c(x)

Esempi di certificato (problema :: certificato)

- SAT :: assegnazione di valori Vero / Falso alle variabili

Fatto: SAT appartiene a NP

La classe NP

Formalmente
La classe NP è l'insieme di tutti i linguaggi L per cui esiste una funzione c(x) e un polinomio p per cui x se x appartiene a L allora

- |c(x)| (lunghezza di c(x)) verifica $|c(x)| \leq p(|x|)$ (cioè la lunghezza di c(x) è polinomiale nella lunghezza di x)
- Esiste una macchina di Turing V che con ingresso x e |c(x)| certifica che x appartiene a L (terminando in uno stato finale) e ha complessità temporale polinomiale in |x| e |c(x)

Nota: l'esistenza di un certificato di appartenenza (x in L) non implica che esista un certificato che dimostra che x NON appartiene a L

Nota: nella definizione al posto di una macchina di Turing si può sostituire un algoritmo oppure un programma scritto in un qualunque linguaggio di programmazione

Linguaggi NP-completi

Abbiamo visto che la classe P è contenuta nella classe NP
DOMANDA: La classe P è uguale alla classe NP?
RISPOSTA: non lo sappiamo

Informalmente, tale problema è equivalente a chiedersi se trovare una soluzione di un problema è, in generale, più difficile che verificare se una soluzione proposta è corretta.
Chiunque si sia posto una simile domanda sa che la risposta più naturale è quella affermativa: per questo motivo, il problema è stato quasi sempre presentato come quello di trovare la dimostrazione della congettura $P \neq NP$
che afferma che la classe P `e diversa dalla classe NP.

La classe P è uguale a NP? (problema da 1 milione di dollari)
CONGETTURA: NO

Linguaggi NP-completi

Riduzione fra linguaggi (definizione vista in indecidibilità)
Intuitivamente, tale tecnica consiste nel dimostrare che, dati due linguaggi L1 e L2, L1 non è più difficile di L2 o, piu precisamente, che se esiste una macchina di Turing che decide L2, allora esiste anche una macchina di Turing che decide L1.

Un linguaggio L1 è riducibile a un linguaggio L2 se esiste una funzione totale calcolabile f : {0,1}* → {0,1}*, detta riduzione, tale che, per ogni stringa binaria x, x appartiene a L1 se e solo se f(x) appartiene a L2

Per studiare la classe P e NP ci limitiamo a riduzioni polinomiali
Un linguaggio L1 è **polinomialmente** riducibile a un linguaggio L2 se esiste una riduzione f da L1 a L2 che sia calcolabile da un algoritmo con complessità temporale polinomiale.

Linguaggi NP-completi

DOMANDA: La classe P è diversa da NP?

Definizione Un linguaggio L è NP-completo se L appartiene a NP e se ogni altro linguaggio in NP è polinomialmente riducibile a L.

Intuitivamente, un linguaggio NP-completo è tra i più difficili della classe NP, nel senso che se appartenesse alla classe P, allora l'intera classe NP sarebbe inclusa nella classe P.

Teorema SAT è NP-completo.

Prova: complessa. Dimostra che - dato L un linguaggio in NP- siano p e V come nella definizione di NP.
La prova mostra che - data una stringa x - esiste una formula di SAT che è soddisfacibile se e solo se x appartiene a L

Linguaggi NP-completi

Teorema SAT è NP-completo.

Prova: complessa.
Dato un linguaggio L in NP, sappiamo che esiste un algoritmo con complessità temporale polinomiale V e un polinomio p tali che, per ogni stringa x, se è in L, allora esiste una stringa y di lunghezza non superiore a p(|x|) tale che V con x e y in ingresso termina in uno stato finale, mentre se x non appartiene a L, allora, per ogni sequenza y, V con x e y in ingresso termina in uno stato non finale.
L'idea della dimostrazione consiste nel costruire, per ogni x, in tempo polinomiale una formula booleana F(x,y) le cui uniche variabili libere sono p(|x|) variabili $y_0, y_1, \ldots, y_{p(|x|)-1}$, intendendo con ciò che la soddisfacibilità della formula dipende solo dai valori assegnati a tali variabili

Linguaggi NP-completi

DOMANDA: La classe P è diversa da NP?

Un linguaggio L è NP-completo se L appartiene a NP e se ogni altro linguaggio in NP è polinomialmente riducibile a L.

Intuitivamente, un linguaggio NP-completo è tra i più difficili della classe NP, nel senso che se appartenesse alla classe P, allora l'intera classe NP sarebbe inclusa nella classe P.

Fatti:

- ci sono migliaia di problemi NP-completi
- se dimostriamo che un problema NP-completo ha algoritmo polinomiale allora P=NP
- Per nessun problema NP completo conosciamo un algoritmo polinomiale

Questo ci suggerisce che P≠NP

Riduzione polinomiale da SAT a 3SAT

Data una formula F con k clausole si definisce una formula F' che è soddisfacibile se e solo se F è soddisfacibile (modifica clausole e aggiunge variabili)
Una clausola con k letterali (k≠3) si trasforma nel seguente modo

- $k = 1$: in questo caso,

$$D_c = \{\{l_0, y_0^c, y_1^c\}, \{l_0, y_0^c, \neg y_1^c\}, \{l_0, \neg y_0^c, y_1^c\}, \{l_0, \neg y_0^c, \neg y_1^c\}\}$$

  Osserviamo che le quattro clausole in D_c sono soddisfatte se e solo se l_0 è soddisfatto.

- $k = 2$: in questo caso, $D_c = \{\{l_0, l_1, y_0^c\}, \{l_0, l_1, \neg y_0^c\}$. Osserviamo che le due clausole in D_c sono soddisfatte se e solo se l_0 oppure l_1 è soddisfatto.

- $k > 3$: in questo caso,

$$D_c = \{\{l_0, l_1, y_0^c\}, \{\neg y_0^c, l_2, y_1^c\}, \{\neg y_1^c, l_3, y_2^c\}, \ldots, \{\neg y_{k-4}^c, l_{k-2}, l_{k-1}\}\}$$

Programmazione a numeri interi

La programmazione a numeri interi (PI) è un fondamentale pro blema di ottimizzazione. Possiamo pensare a una program-mazione lineare con variabili limitate ad assumere solo valori interi

Input: un insieme V di variabili intere, un insieme di disuguaglianze su V, una funzione obiettivo di massimizzazione f(V) e un intero B.

Output: esiste un attributo di interi a V tale che tutte le disuguaglianze siano vere e $f(V) \geq B$?

Esempio

due variabili v_1 e v_2 con valori interi e i vincoli

$v_1 \geq 1$, $v_2 \geq 1$, $v_1 + v_2 \leq 3$

Funzione obiettivo $f(V) = v_1 + 2\, v_2$ $\quad B=3$

Una soluzione del problema è $v_1 = 1$, $v_2 = 2$

Se poniamo B=6 non abbiamo soluzione (max valore f(V) compatibile con i vincoli è 5)

Riduzione da 3SAT a programmazione intera

Dimostriamo che la programmazione di interi è difficile usando una riduzione da 3-SAT. Per questa particolare riduzione, il problema generale funzionerebbe altrettanto bene, anhe se in questo utilizzare 3-SAT facilita la riduzione.

In che direzione deve andare la riduzione?
Vogliamo dimostrare che la programmazione di interi è difficile, e sappiamo che 3-SAT è difficile.
Se potessi risolvere il 3-SAT usando la programmazione di interi e la programmazione di interi fosse facile, ciò significherebbe che 3-SAT sarebbe facile.

Quindi dobbiamo “tradurre” 3-SAT nella programmazione a numeri interi.

Riduzione da 3SAT a programmazione intera

Ogni istanza di 3SAT contiene variabili booleane (vero / falso).
Ogni istanza di programmazione intera contiene variabili intere (valori ristretti a 0,1,2, ...) e vincoli.
Nel seguito le variabili intere corrispondono alle variabili booleane

Data una formula con n variabili per ogni variabile x_i del problema 3SAT abbiamo due variabili v_i e w_i
Per limitare ogni variabile di programmazione intera a valori di 0 o 1, aggiungiamo i seguenti vincoli $0 \leq v_i \leq 1$ e $0 \leq w_i \leq 1$ per ogni i

Intuitivamente $v_i = 1$ e $w_i = 0$ rappresentano l'assegnazione x_i =vero mentre $w_i = 1$ e $v_i = 0$ rappresentano l'assegnazione x_i =falso

Aggiungiamo i vincoli per ogni i
$1 \leq v_i + w_i \leq 1$ che impone che al massimo una fra v_i e w_i sia 1 (equivalgono a $v_i + w_i = 1$)

Riduzione da 3SAT a programmazione intera

Data una formula con n variabili per ogni variabile x_i del problema 3SAT abbiamo due variabili v_i e w_i . $v_i = 1$ e $w_i = 0$ rappresentano l'assegnazione x_i =vero mentre $w_i = 1$ e $v_i = 0$ rappresentano l'assegnazione x_i =falso

Aggiungiamo i vincoli per ogni i
$1 \leq v_i + w_i \leq 1$ che impone che al massimo una fra v_i e w_i sia 1

Per ogni clausola nell'istanza 3-SAT definiamo un vincolo.
Ad esempio data la clausola C= (x_1 or x_2 or x_3) definiamo il vincolo
$v_1 + v_2 + v_3 \geq 1$ che impone che 1 fra le variabili x_1,x_2 , x_3 sia vera

Anologamente per la clausola C= (x_1 or (not x_2) or (not x_5))
definiamo il vincolo
$v_1 + w_2 + w_5 \geq 1$ che impone che o x_1 è vera oppure una fra le variabili x_2 , x_3 sia falsa

Per soddisfare il vincolo, almeno uno dei letterali della clausola deve essere impostato su 1, quindi a un letterale vero.

Riduzione da 3SAT a programmazione intera

Per stabilire che questa riduzione sia corretta dobbiamo verificare due cose:

• Qualsiasi soluzione SAT fornisce una soluzione al problema IP
Infatti data una qualsiasi soluzione SAT, un valore letterale vero corrisponde ad una variabile posta a 1 nel programma intero, poiché la clausola è soddisfatta.
Pertanto, la somma in ogni vincolo della clausola è almeno 1

• Qualsiasi soluzione IP fornisce una soluzione SAT
In qualsiasi soluzione dell'istanza di programmazione intera, tutte le variabili devono essere impostate su 0 o su 1.
Se v_i =1, assumi x_i =vero; se w_i =1, assumi x_i =falso.
Dato che v_i e w_i non possono essere entrambi posti 1, quindi è un'assegnazione corretta alle variabili.
Inoltre per ogni clausola abbiamo che almeno un letterale deve essere vero.

La riduzione funziona in entrambi i modi, quindi la programmazione intera deve essere difficile.

Riduzione da SAT a PI: conclusioni

Notare le seguenti proprietà, che sono valide in generale per NP-complete:

- La riduzione ha preservato la struttura del problema. Ci ha permesso di formulare il problema SAT in un formato diverso.
- Le possibili istanze IP che possono risultare da questa trasformazione sono solo un piccolo sottoinsieme di tutte le possibili istanze IP. Tuttavia, poiché alcuni di essi sono difficili, il problema generale deve essere difficile.
- La trasformazione cattura l'essenza del perché IP è difficile. Non richiede grandi coefficienti e si liita valori per le variabili 0/1 è sufficiente. Non ha nulla a che fare con l'iniquità con un numero elevato di variabili. La programmazione a numeri interi è difficile perché soddisfare un insieme di vincoli è difficile quando richiediamo valori interi per le variabili.
- Infatti se eliminiamo il vincolo di interezza e permettiamo di assegnare valroi razionali alle variabili il problema diventa facile

Programmazione a numeri interi: conclusioni

Abbiamo mostrato una riduzione da 3-SAT a PI

Questa riduzione dimostra che PI è tanto difficile quanto SAT

Possiamo dire che PI sia NP-completo?

Programmazione a numeri interi: conclusioni

Abbiamo mostrato una riduzione da 3-SAT a PI

Questa riduzione dimostra che PI è tanto difficile quanto SAT

Possiamo dire con questo che PI sia NP-completo? **NO**

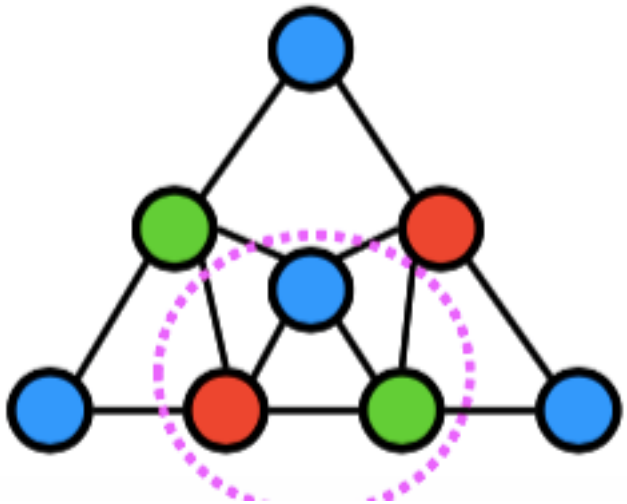
Dobbiamo ancora dimostrare che PI sia nella classe NP
Questa dimostrazioni non è semplice (non la vediamo) e permette di stabilire il seguente teorema

Teorema La programmazione a numeri interi è NP-completo

Colorazione di grafi

Dato un grafo G siamo interessati a colorare i suoi nodi in modo tale che nodi adiacenti abbiano colori diversi

- Il numero cromatico di G, C(G) è il minimo numero di colori necessario per colorare tutti i suoi nodi
- Chiaramente se un grafo ha n nodi abbiamo che $1 \leq C(G) \leq n$
- Il problema di decisione associato è: dato un grafo G può essere colorato con k colori (k intero)?



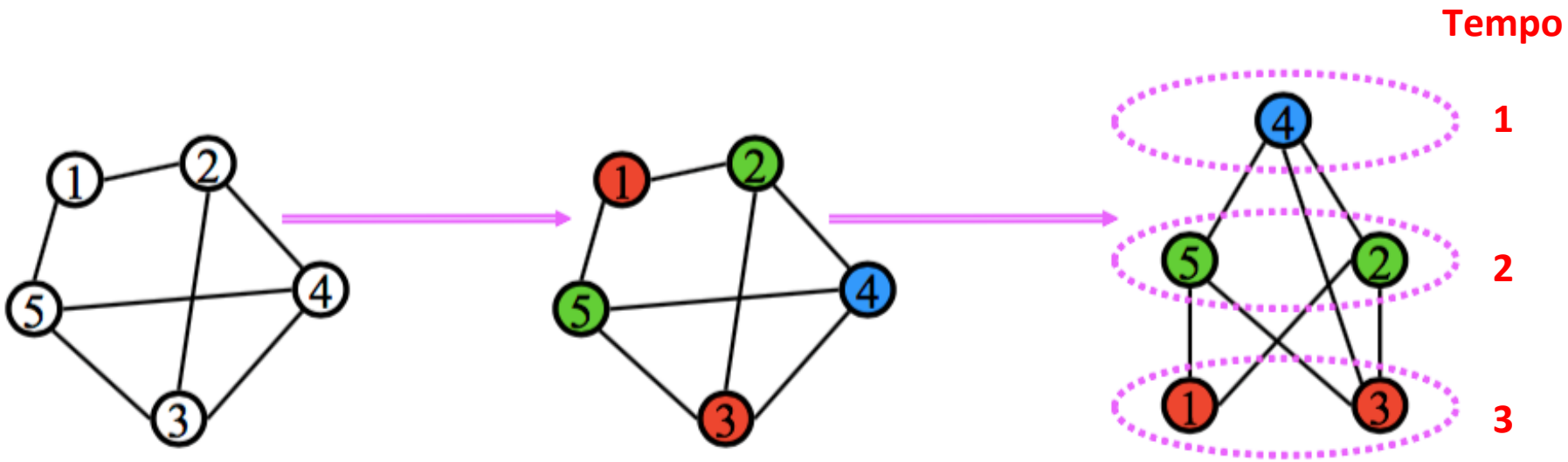
La figura mostra una colorazione con 3 colori

Si noti che se tre nodi sono mutuamente adacenti e formano una cricca allora richiedono tre colori diversi (vedi nodi cerchiati a sinistra)

Colorazione di grafi

Esempio di applicazione: sequenziamento di lavori (scheduling)

- Dobbiamo assegnare lavori a intervalli di tempo
- Alcuni lavori sono in conflitto e non possono essere eseguiti insieme (ad esempio usano risorse condivise)
- Modella i lavori con i nodi di un grafo e i conflitti come archi
- Il minimo numero di colori usato rappresenta il minomo tempo di completamento (“makespan”) per finire i lavori



Riduzione da 3SAT a 3-colorazione di grafi

Dimostriamo che la decidere se un grafo è colorabile con 3 colori di interi è difficile usando una riduzione da 3-SAT.
Idea costruire un gadget che simula OR
Tre colori: verde, blu (Rappresenta T - vero, rosso (rapp. F- falso)

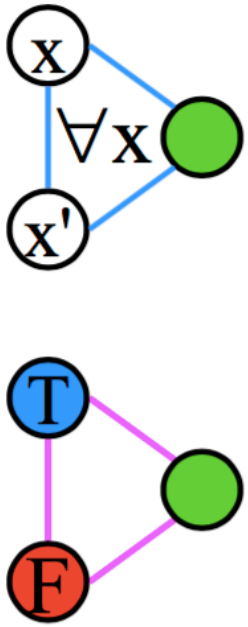
Per ogni variabile x abbiamo
(x' rappresenta not x)

Il grafo a destra mostra x e x' collegati a verde ➔

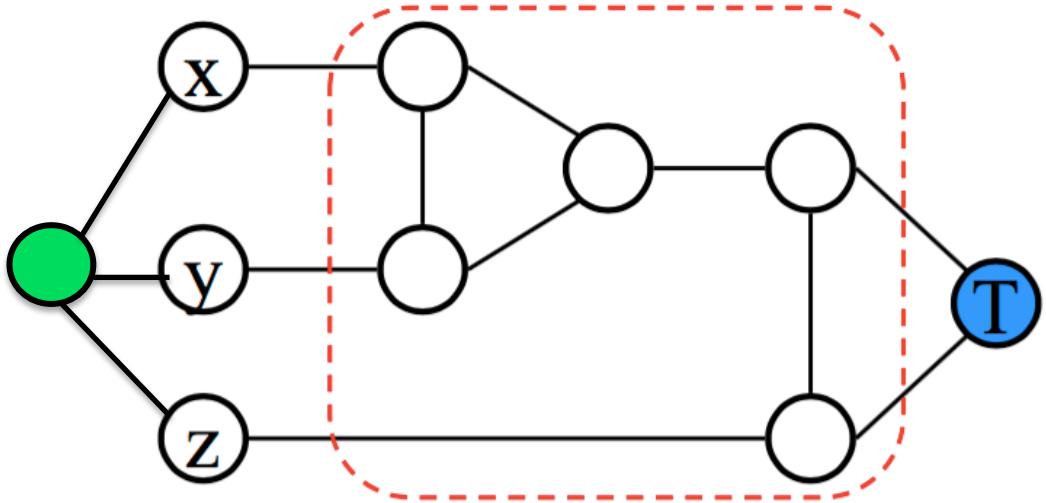
- x=T e x'=F

Oppure

- X=F e x'=T

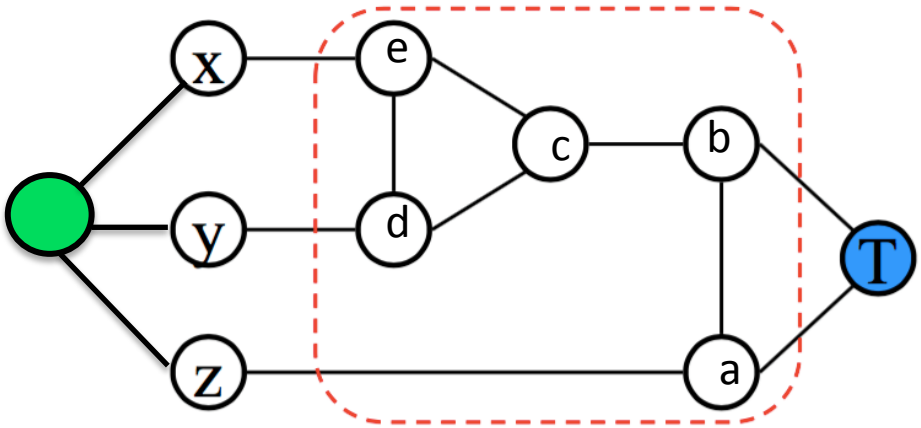


Per ogni clausola del tipo (x or y or z)
il grafo seguente richiede che almeno uno fra x, y, z sia colorato blu (T)



Riduzione da 3SAT a 3-colorazione di grafi

Tre colori: verde, blu (Rappresenta T - vero, rosso (rapp. F- falso)
Per ogni clausola (x or y or z) il grafo seguente richiede che almeno uno fra x, y, z sia colorato blu (T)
Prova: per contraddizione: esiste colorazione in cui x,y,z non sono blu; poiché sono adiacenti ad un nodo verde segue che x,y, z sono colorati rosso



NOTA: la costruzione finale permette di assumere che i due nodi verde e blu sopra possano essere colorati con questi colori

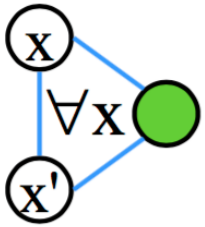
1. z rosso implica le seguenti colorazioni:
   - a= verde che quindi implica
   - b= rosso
2. Ora osserva i nodi d,e,c
2.a Devono avere tre colori diversi
2.b Ognuno è adiacente ad un nodo rosso e quindi non possiamo colorarli rosso
2.c Pertanto d,e,c non possono essere colorati usando solo verde, rosso, blu
3. Quindi l'assunzione fatta che nessuno fra x,y,z sia blu è falsa

Riduzione da 3SAT a 3-colorazione di grafi

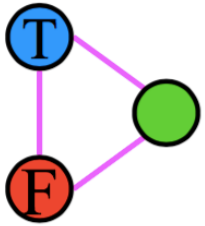
Data la formula (assumiamo che x' = not x)

(x or y or z) and (not x or not y or z) and (not x or y or nor z)

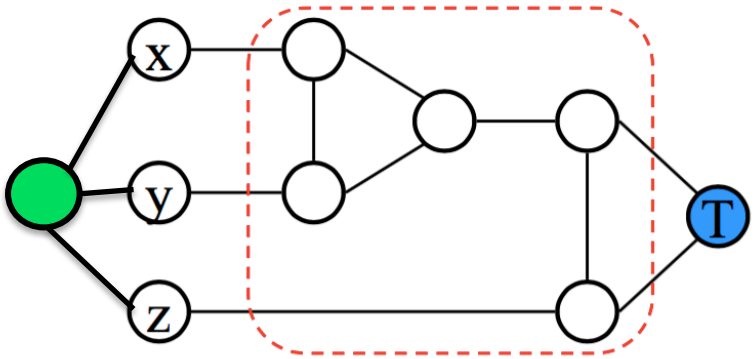
Otteniamo il grafo grande a destra

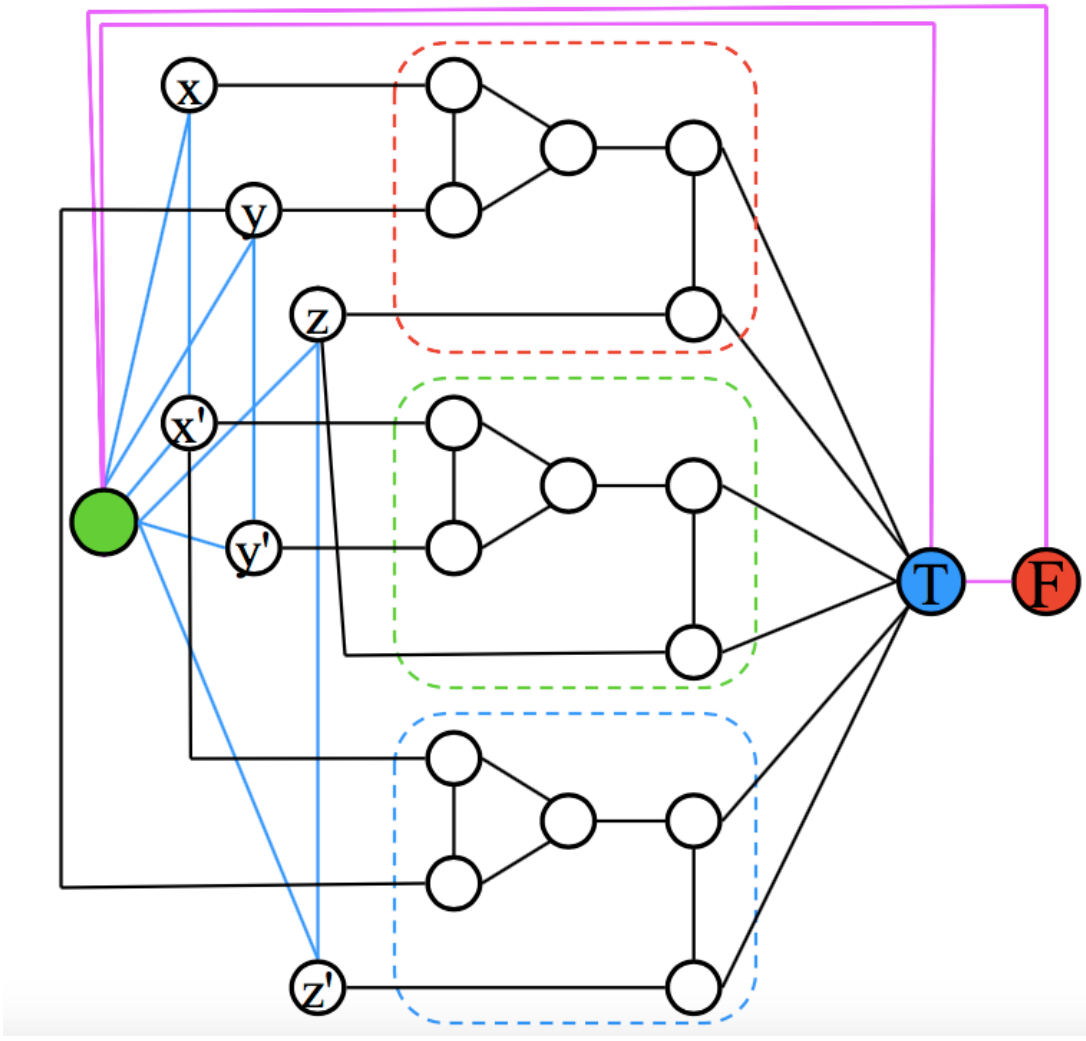


Impone che x e x' abbiano rosso o blu valori diversi fra loro



Impone che almeno uno fra x,y,z sia blu (vero)



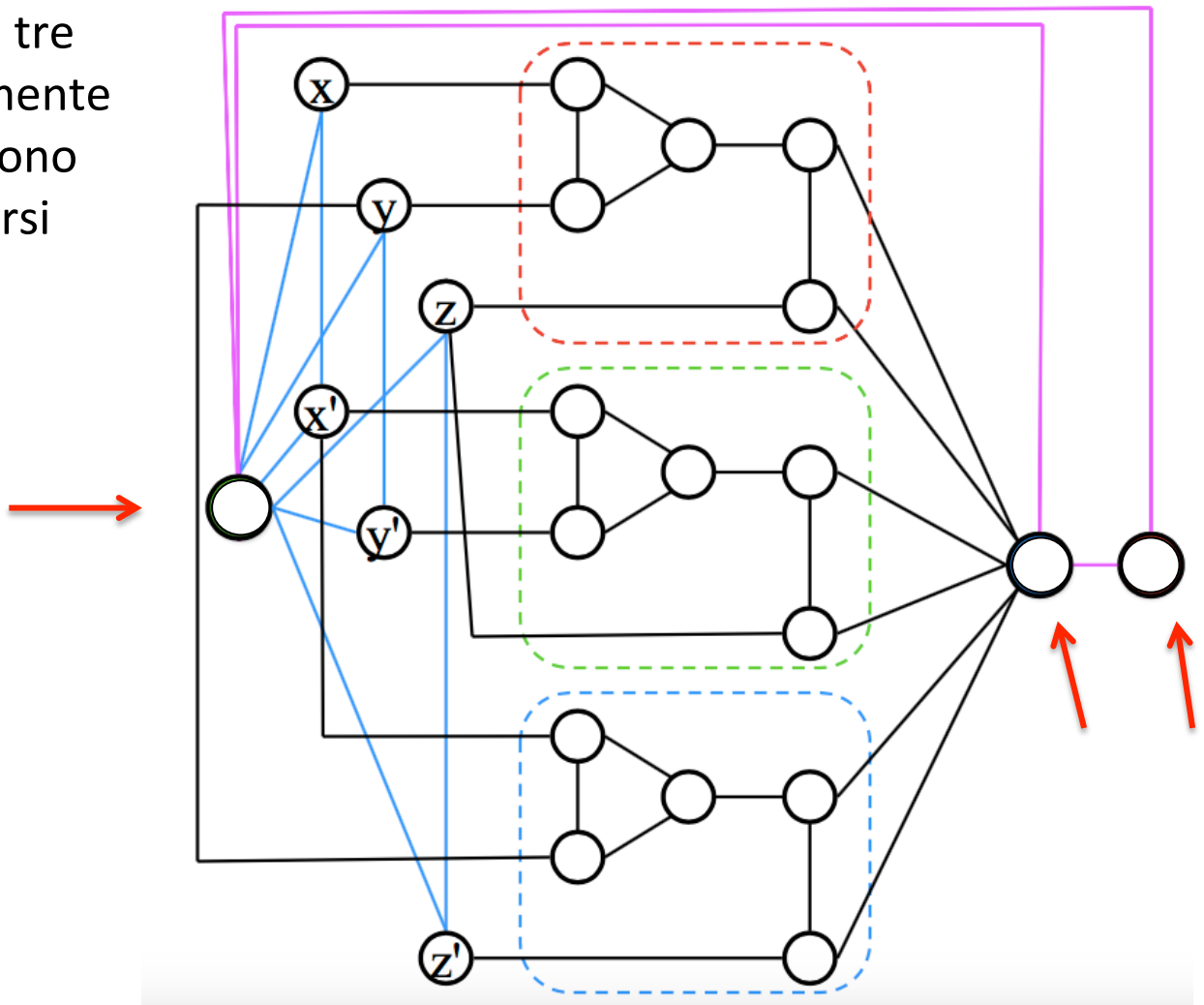


Data la formula (assumiamo che x’ = not x)

(x or y or z) and (not x or not y or z) and (not x or y or nor z)

Otteniamo il grafo

I tre nodi indicati da tre frecce sono mutuamente connessi quindi devono usare tre colori diversi



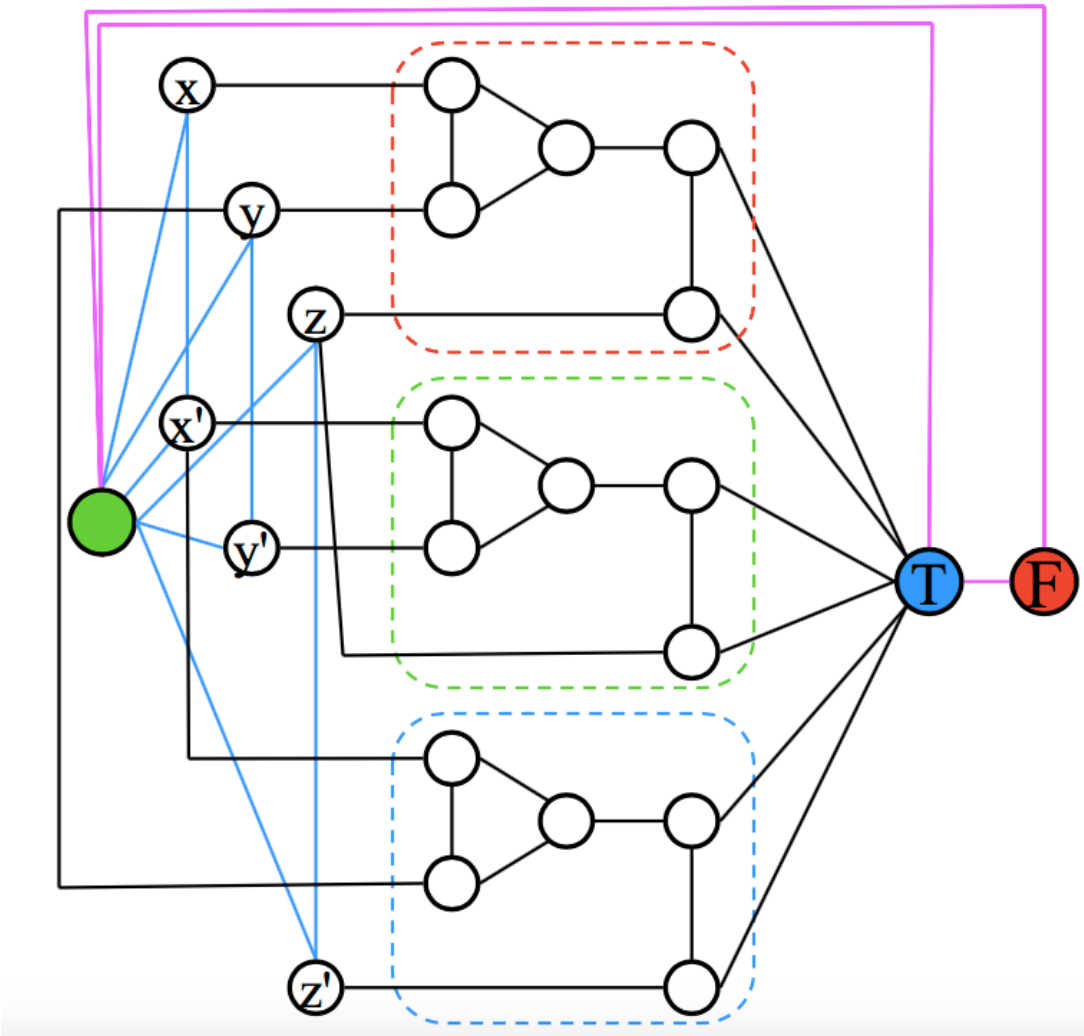
Data la formula (assumiamo che x’ = not x)

(x or y or z) and (not x or not y or z) and (not x or y or nor z)

Otteniamo il grafo

I tre nodi già colorati sono mutuamente connessi quindi devono usare tre colori diversi

Posso scegliere tre colori arbitrari come in figura



Riduzione da 3SAT a 3-colorazione di grafi

verde, blu (vero), rosso (falso)

Data la formula

(x or y or z) and (not x or not y or z) and (not x or y or nor z)

(x or y or z) and (x' or y' or z) and (x' or y or z')

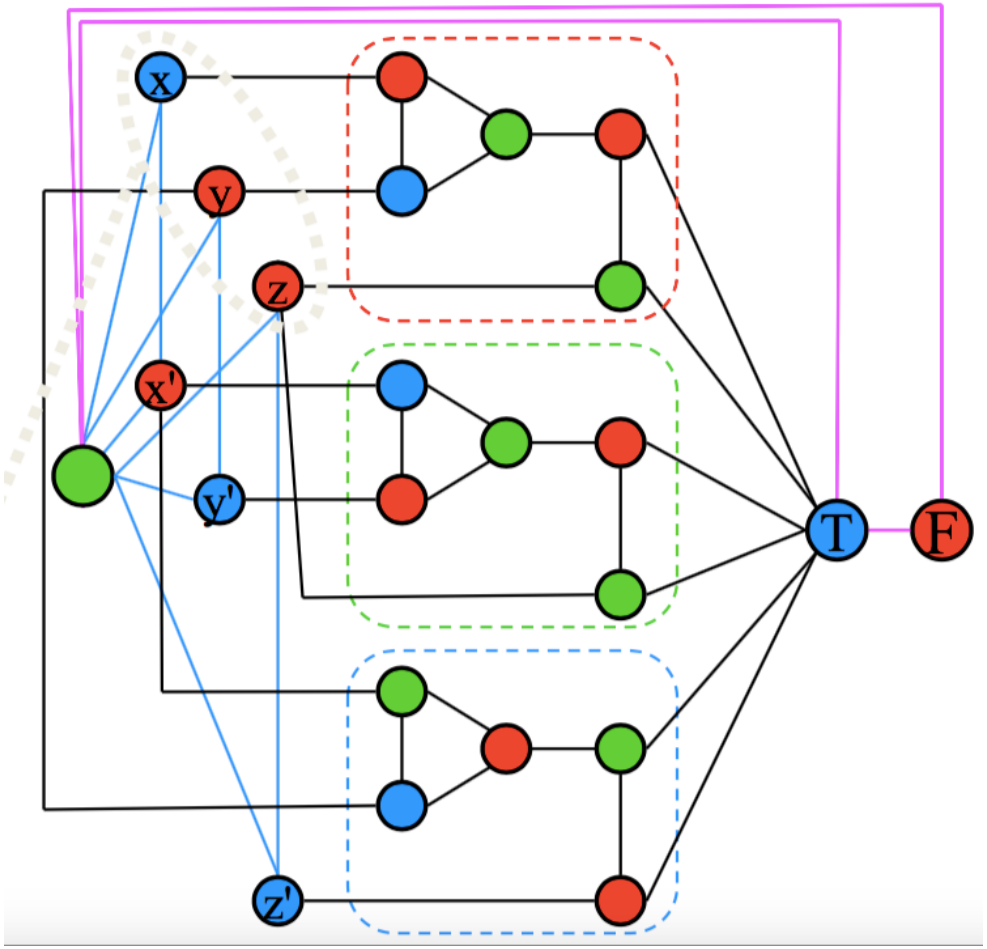
La figura mostra 3 colorabile con

x= blu, y= rosso, z= rosso

La figura verifica che

- x'=rosso, y'= blu, z= blu
- Per ogni gadgetdi clausola almeno uno dei nodi collegati a sinistra è blu (vero)

Quindi se assumiamo x=vero, y falso, z= falso otteniamo che 1 e 2 precedenti implicano che qusta sia un'assegnazione di valori di verità che rende vera la formula



Riduzione da SAT a 3colorazione: conclusioni

1. Non è difficile modificare la prova precedente per dimostrare che decidere se un grafo è colorabile con k colori (k >3) è NP-completo

2. Cosa rende difficile il problema della colorazione: vertici di grado alto?
- NO: possiamo modificare la riduzione in modo tale che ogni nodo abbia massimo grado 3

3. I seguenti problemi sono facili (polinomiali)
- Decidere se un grafo può essere colorato con 1 colore (banale)
- Decidere se un grafo può essere colorato con 2 colori (buon esercizio)

4. Per alcune classi di grafi il problema è semplice
- Ogni albero può essere colorato con due colori

4. Ma non sempre restringere lo studio semplifica
- Grafo planare: un grafo che può essere disegnato su un folgio senza avere incroci fra gli archi
- Colorare un grafo planare con 3 colori è NP- completo (difficile)
- Colorare un grafo planare con 4 colori è facile (polinomiale)

Vertex Cover e Independent Set

Vertex Cover
Dato un grafo G ed un intero k ci chiediamo se esiste un insieme S di k nodi che coprono tutti gli archi. (S copre i nodi se per ogni arco (u,v) abbiamo che almeno uno fra u e v appartiene a S

Independent set
Dato un grafo G ed un intero k ci chiediamo se esiste un insieme S di k nodi che non sono collegati fra loro (cioè se u e v appartengono a S (non esiste arco (u,v)

E' facile vedere che i due problemi sono in NP
Usa come certificato un insieme di k nodi

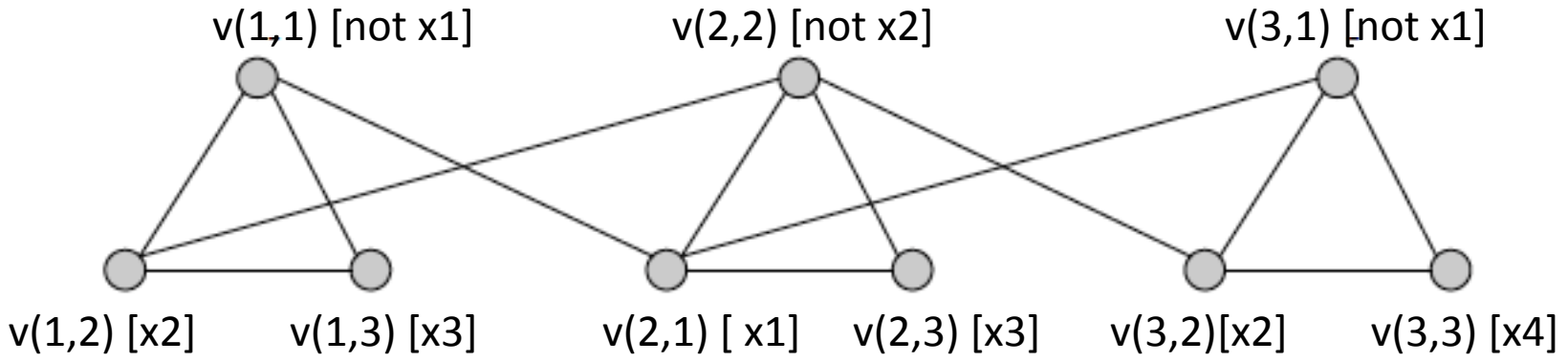
Algoritmo (deterministico)
Scegli un insieme S di k nodi
    Verifica se S è un vertex cover (o un independent set)

Riduzione da 3-SAT a Independent set

Data una formula f di 3SAT con m clausole si definisce un grafo con 3m nodi, 3 per ciascuna clausola e indichiamo con v(C,i) il nodo associato al letterale l_i della clausola C. Gli archi sono così definiti

- I tre nodi associati ad una medesima clausola sono mutuamente collegati da un arco
- Due nodi v(C,i) e v(C',j) associati a due letterali di due clasuole diverse (C≠C') sono collegati da un arco se la variabile x corrisponde sia al letterale i della clausola C che al letterale j della clausola C' e in un caso è positiva e in un altro è negativa

Esempio. Data la formula:
((not x1) or x2 or x3) and (x1 or (not x2) or x3) and ((not x1) or x2 or x4)
Otteniamo il seguente grafo (in cui fra [..] è indicato il letterale della clausola

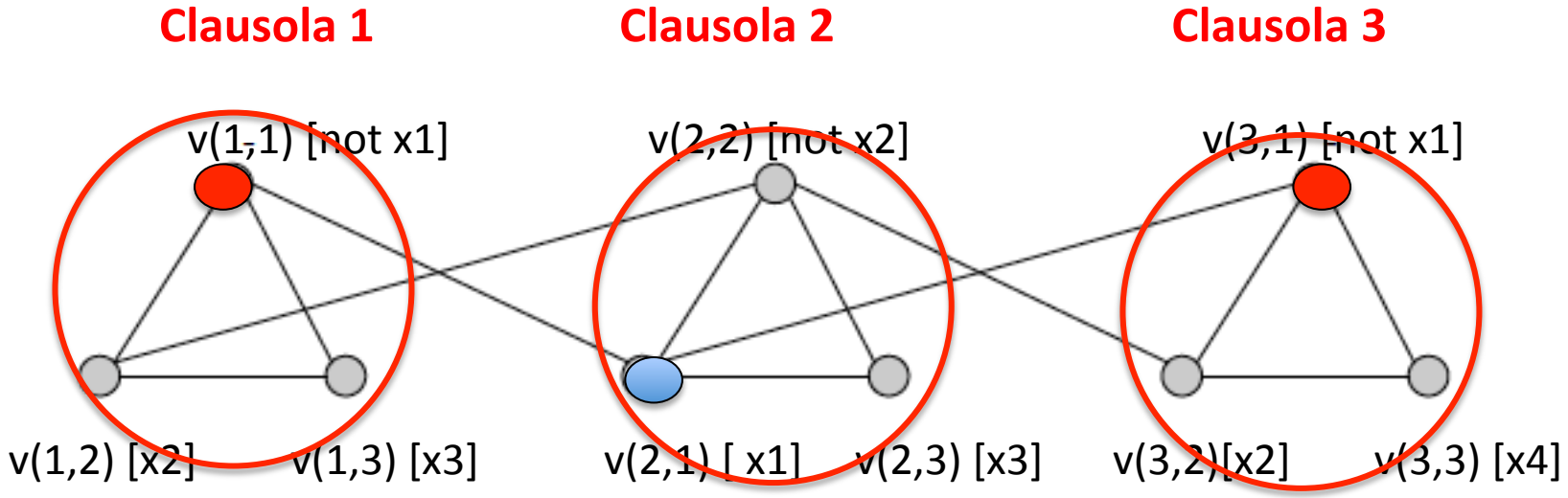


Riduzione da 3-SAT a Independent set

Data una formula F di 3SAT con m clausole si definisce un grafo con 3m nodi, 3 per ciascuna clausola e indichiamo con v(C,i) il nodo associato al letterale l_i della clausola C. Gli archi sono così definiti

- I tre nodi associati ad una medesima clausola sono mutuamente collegati da un arco
- Due nodi v(C,i) e v(C',j) associati a due letterali di due clasuole diverse (C≠C') sono collegati da un arco se la variabile x corrisponde sia al letterale i della clausola C che al letterale j della clausola C' e in un caso è positiva e in un altro è negativa

Esempio. Data la formula F:
((not x1) or x2 or x3) and (x1 or (not x2) or x3) and ((not x1) or x2 or x4)
Otteniamo il seguente grafo (in cui fra [..] è indicato il letterale della clausola
Nota che il nodo blu (v(2,1)) ha due archi con nodi rossi di altre clausole perché v(2,1) corrisponde al letterale x1 mentre i nodi rossi corrispondono al letterale not x1



Riduzione da 3-SAT a Independent set

Teorema G contiene un insieme indipendente di taglia m se e solo se f è soddisfacibile

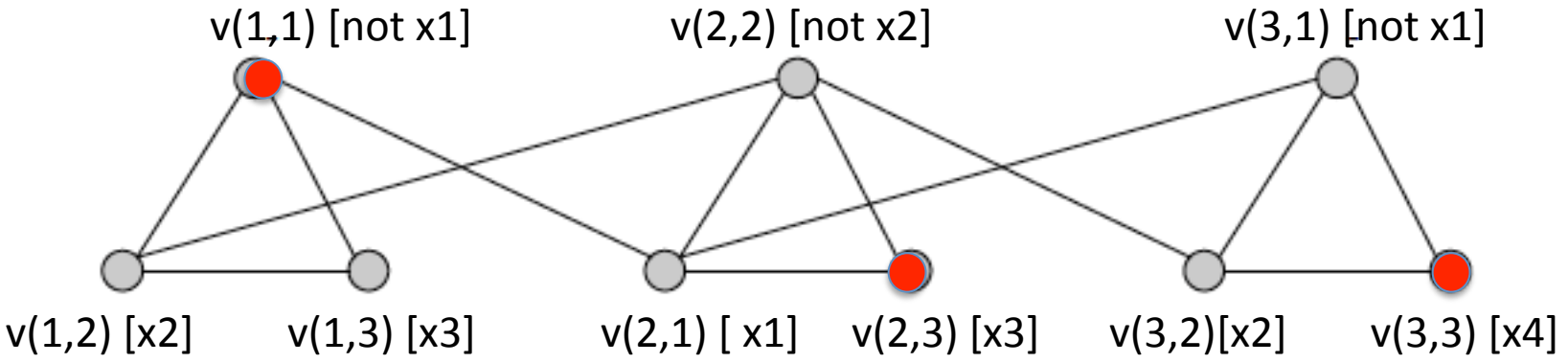
Prova 1 ⇒ (dato insieme indipendente S di dimensione m ottieni soluzioni a 3SAT)

Sia S ins. Indipendente di taglia m. E' facile vedere che

- S contien esattamente un vertice in ogni triangolo (nodi triangolo sono adiacenti, m triangoli)
- Poni questi letterali a Vero
- Questa assegnazione è consistente (gli archi fra clausole impediscono di dare a x sia il valore vero che falso) e soddisfa F (ogni clausola ha un letterale vero)

Assegnazione corrispondente a nodi rossi (insieme indipendente) da assegnazione x1=Falso x3=x4= Vero (x2 può essere sia Vero che Falso e la formula è soddisfatta)

F= ((not x1) or x2 or x3) and (x1 or (not x2) or x3) and ((not x1) or x2 or x4)



Riduzione da 3-SAT a Independent set

Teorema G contiene un insieme indipendente di taglia m se e solo se f è soddisfacibile

Prova

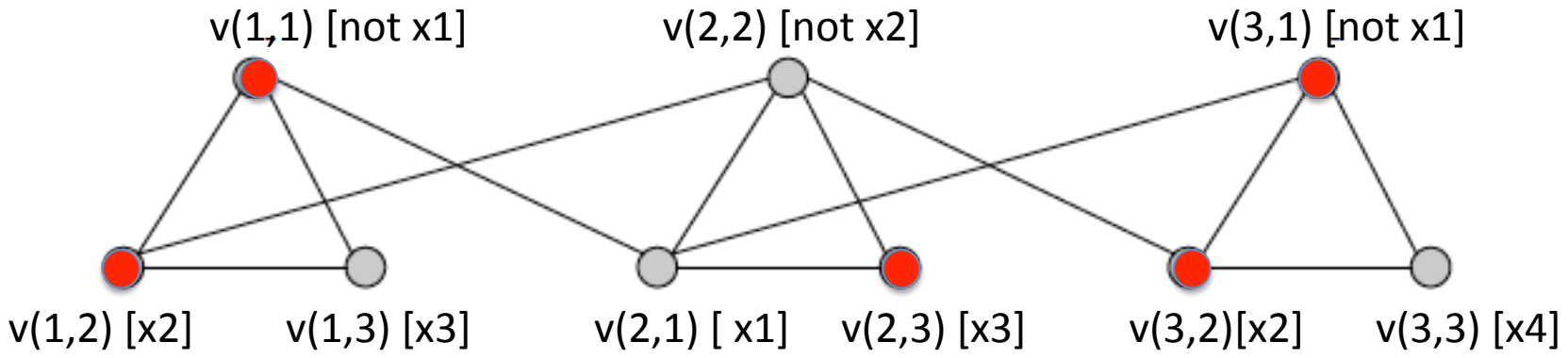
2 $\Leftarrow$ (data soluzione di 3SAT ottieni insieme indipendente)

Data un'assegnazione soddisfa F scegli un letterale vero da ogni triangolo (ci deve essere dato che ciascuna clausola è soddisfatta).

Questo insieme è un insieme indipendente di taglia m (ricorda m triangoli)

F= ((not x1) or x2 or x3) and (x1 or (not x2) or x3) and ((not x1) or x2 or x4)

Assegnazione soddisfa F: x1=x4= Falso x2=x3=Vero



Riduzione da 3-SAT a Independent set

Teorema G contiene un insieme indipendente di taglia m se e solo se f è soddisfacibile

Prova

2 $\Leftarrow$ (data soluzione di 3SAT ottieni insieme indipendente)

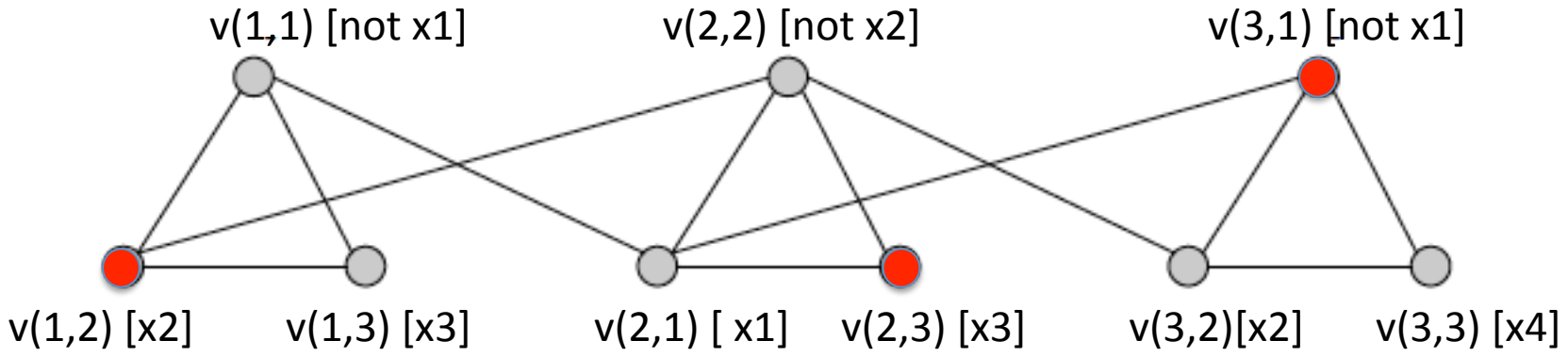
Data un'assegnazione soddisfa F scegli un letterale vero da ogni triangolo (ci deve essere dato che ciascuna clausola è soddisfatta).

Questo insieme è un insieme indipendente di taglia m (ricorda m triangoli)

F= ((not x1) or x2 or x3) and (x1 or (not x2) or x3) and ((not x1) or x2 or x4)

Assegnazione soddisfa F: x1=x4= Falso x2=x3=Vero

Nodi rossi sono insieme indipendente



Riduzione da 3-SAT a Independent set

Teorema G contiene un insieme indipendente di taglia m se e solo se f è soddisfacibile

Prova

1 ⇒ (dato insieme indipendente S di dimensione k ottieni soluzioni a 3SAT)

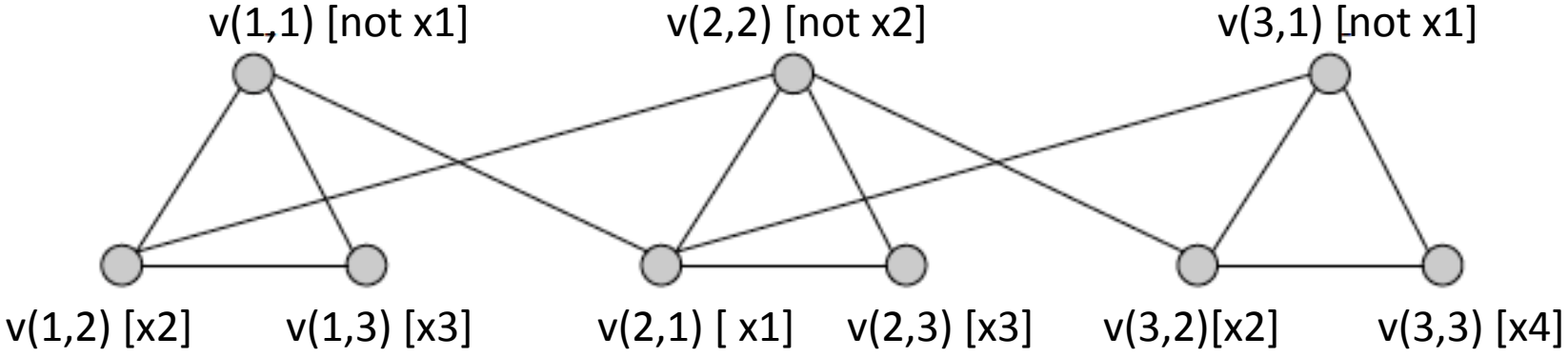
Sia S ins. Indipendente di taglia m. E' facile vedere che

- S deve contenere esattamente un vertice in ogni triangolo
- Poni questi letterali a Vero
- Questa assegnazione è consistente (gli archi fra clausole impediscono di dare a x sia il valore vero che falso) e soddisfa F (ogni clausola ha un letterale vero)

2 ⇐ (data soluzione di 3SAT ottieni insieme indipendente)

Data un'assegnazione soddisfa F scegli un letterale vero da ogni triangolo. Questo insieme eè un insieme indipendente di taglia m

F= ((not x1) or x2 or x3) and (x1 or (not x2) or x3) and ((not x1) or x2 or x4)



Riduzione da SAT a

Vertex Cover
Dato un grafo G ed un intero k ci chiediamo se esiste un insieme S di k nodi che coprono tutti gli archi. (S copre i nodi se per ogni arco (u,v) abbiamo che almeno uno fra u e v appartiene a S

Independent set
Dato un grafo G ed un intero k ci chiediamo se esiste un insieme S di k nodi che non sono collegati fra loro (cioè se u e v appartengono a S (non esiste arco (u,v)

E' facile vedere che i due problemi sono in NP
Algoritmo
Scegli un insieme S di k nodi
    Verifica se S è un vertex cover (o un independent set)

Riduzioni NP: sommario

Dimostrare che A si riduce a B

Richiede trasformazione f:A→ B (istanze di A in istanze di B) tale che

- Se istanza x di A appartiene al linguaggio anche f(x) appartiene e B
- Se istanza y di B appartiene al linguaggio anche $f^{-1}(y)$ appartiene e A

Data riduzione da A a B: B è almeno tanto difficile quanto A

- A difficile (NP-completo) → B difficile
- B facile (polinomiale) → A facile
- A facile (polinomiale)→ ?: la riduzione non mi aiuta a capire se B sia polinomiale o no

.

Riduzioni NP: Robustezza definizione

1. Composizione di polinomi è un polinomio
quindi se riduco SAT a problema A e poi A a problema B e poi B a problema C la composizione delle riduzioni fornisce una riduzione polinomiale da SAT a C, quindi concludo C è NP-difficile (e se C è in NP allora è NP-completo)

2. La definizione data non dipende dal modello di calcolo
I modelli di calcolo noti (MdT, JAVA, Python ecc.) sono tutti polinomialmente equivalenti; infatti un algoritmo polinomiale in un modello A si traduce in un algoritmo polinomiale in un altro modello B e viceversa.

Problemi di ottimizzazione

Colorazione di grafi
Problema di decisione: Esiste una colorazione con k o meno colori?
Problema di ottimizzazione: Qual è il minimo numero di colori necessario per colorare un grafo?

Problemi di ottimizzazione

La definizione di NP si estende a problemi di ottimizzazione riducendo la ricerca dell'ottimo a più problemi di riconoscimento di linguaggi
Esempio determinare numero cromatico Crom(G) di un grafo G
Il problema richiede di trovare il più piccolo k tale che G è k colorabile

1. Il numero cromatico di un grafo con n nodi è compreso fra 1 e n
2. Per trovare k(G) effettuiamo una ricerca binaria iniziando con h= n/2 e ci chiediamo se G sia h colorabile
   - Se sì allora ci chiediamo se G sia n/4 colorabile
   - Se no ci chiediamo se G sia 3n/4 colorabile
3. Risolvendo (log n) problemi di riconoscimento risolviamo il problema di ottimizzazione

Si applica a tutti i problemi che consideriamo e giustifica l'attenzione su problemi di decisione

Problemi di ottimizzazione e problemi NP-hard

Posso avere problemi più difficili di NP

Un linguaggio L è NP-completo se L appartiene a NP e se ogni altro linguaggio in NP è polinomialmente riducibile a L.

Un linguaggio L è NP-hard (NP-difficile) ~~se L appartiene a NP e~~ se ogni altro linguaggio in NP è polinomialmente riducibile a L (L può non appartenere a NP)

I linguaggi NP-hard possono essere più difficili di NP

Esempi
Formule logiche con quantficatori (per ogni, esiste)
Quantificatori esempio:
Data una formula booleana f(x,y,v,w,z) mi chiedo se
per ogni x [(esiste y per cui f e' vera) oppure (per ogni v f e' falsa)]
Esempio gioco degli scacchi un giocatore si chiede se
Esiste una mossa del bianco tale che (Per ogni mossa del nero (esiste una mossa del bianco che (per ogni mossa del nero…..
Richiede spazio polinomiale

Problema della fermata
non decidibile

Teoria NP-completezza: conclusioni

Desiderata

Classificare i problemi distinguendo quelli che possono essere risolti in tempo polinomiale da quelli che non si possono risolvere in tempo polin.

Notizia frustrante

Molti problemi fondamentali non sono classificabili (anche se studiati da decenni)

Cosa sappiamo

Teoria NP-completezza mostra che molti di questi problemi sono "computazionalmente equivalenti" e appaiono essere formulazioni differenti dello stesso problema

Spesso il confine fra facile (polinomiale) e difficile (NP completo) è noto ma in alcuni casi controintuitivo

Facile (polinomiale)	Difficile (NP-hard)
2-SAT	3-SAT
Colorare i nodi di un grafo con 2 colori	Colorare i nodi di un grafo con 3 colori
Programmazione lineare	Programmazione a numeri interi
Cammino minimo in un grafo	Cammino più lungo in un grafo
Vertex Cover in grafi bipartiti	Vertex Cover in grafi generali
Test per stabilire se un numero è primo	Fattorizzazione di un numero intero

Teoria NP-completezza: classi di complessità

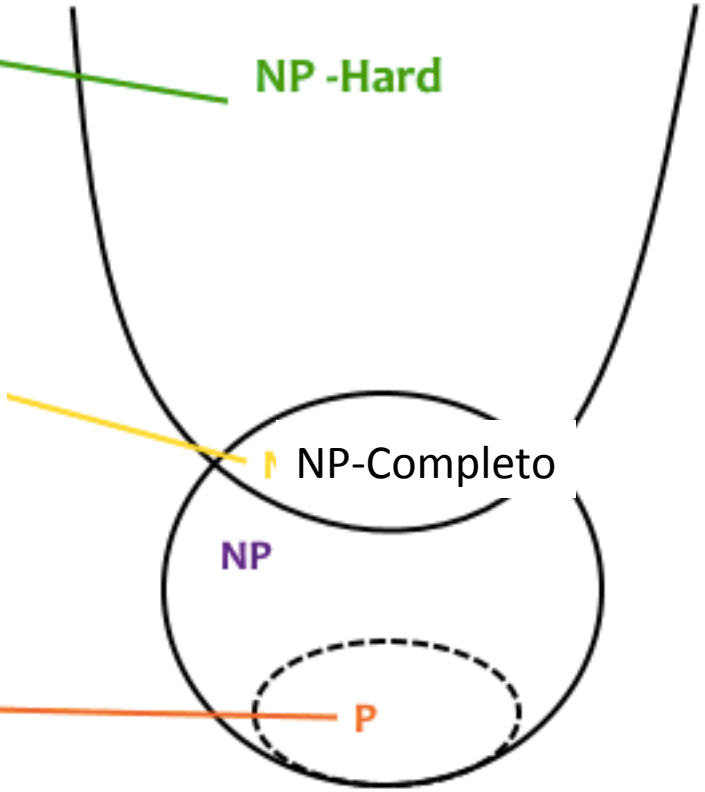
La figura riassume le relazioni fra i diversi problemi assumendo che la congettura P≠NP sia vera

NP-Hard:
L è un problema **NP hard** se esiste una riduzione da un problema NP completo a L (però L non appartiene necessariamente alla classe P)

Problema della fermata
Soddisfacibilità di formule logiche con quantificatori

SAT, 3SAT,
Cricca in un grafo
Colorazione di grafi

Massimo accoppiamento in un grafo
2SAT, Cammino più breve in un grafo,
Calcolo Massimo Comun Divisore,



Teoria NP-completezza: implicazioni pratiche

Il vostro capo vi chiede di trovare un algoritmo veloce (polinomiale) per un problema; avete due settimane di tempo

Non riuscite a trovare un algoritmo veloce e dopo due settimane il capo vi chiama e si lamenta; voi siete in difficoltà
Avete un'altra settimana di tempo ….

Teoria NP-completezza: implicazioni pratiche

Dopo una settimana voi riuscite a provare che il problema datovi è NP-completo

A questo punto potete dire al capo che: "*Io non sono riuscito a trovare un algoritmo polinomiale ma nessuno di questi ricercatori è riuscito a farlo. La congettura è che non esiste un algoritmo veloce; quindi… (a vostra scelta…)*"

Esercizi

- Dimostrare che SAT, colorazione di un grafo, sono in NP
- Ridurre Independent set a vertex cover (e viceversa)
- Ridurre vertex cover a Programmazione a numeri interi
- Ridurre Independent set a Programmazione a numeri interi
- Ridurre colorazione di un grafo con 3 colori a Programmazione a numeri interi
- Il problema Cricca è così definitio: dato un grafo G e un intero k ci si cheide se esiste un insieme di k nodi a due a due collegati da un arco. Ridurre Independent set a Cricca
- Dimostrare che il problema di soddisfacibilità per formule logiche in forma disgiuntiva è polinomiale (una formula logica disgiuntiva è del tipo

  (l1 and l2 and l3) or (l4 and … and...) or (... and … and...) ...

  esempio formula disgiuntiva

  (x1 and x2 and (not x4))or((not x1) and x2 and x3) or ((not x1) and x2 and x4)