UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Computabilità, Complessità e Logica

Prof. Adriano Peron

Complessità: Tempo

Complessità computazionale

- Contesto
- La prova della decidibilità di un problema è un passo preliminare indispensabile per dimostrare la trattabilità di un problema
- Dal punto di vista pratico la decidibilità non è spesso una proprietà sufficiente
- E indispensabile valutare se la quantità di risorse necessaria per la risoluzione del problema è affrontabile
- Le due dimensioni principali per l'analisi della complessità sono
  - Tempo (numero di passi elementari)
  - Spazio (memoria necessaria)

Complessità computazionale

- Contesto
- L'appartenenza di un problema a una classe di complessità (come già la decidibilità/indecidibilità) sono proprietà intrinseche dei problemi
- Il modello adottato per la definizione di computabilità (le Macchine di Turing) è anche lo strumento per definire le classi di complessità
- Il tempo di esecuzione di una MdT è il numero di passi elementari della MdT
- Il tempo di esecuzione è valutato in funzione di un unico parametro: la dimensione dell'input sul nastro all'inizio della computazione
- Il tempo considerato è il valore peggiore ottenibile rispetto alla dimensione dell'input

Complessità computazionale

- **Definizione**
- **Si consideri una MdT deterministica M. La complessità temporale di M è la funzione $f: N \rightarrow N$ dove f(n) è il massimo numero di passi usato da M su un input di lunghezza n.**
- **Si dice anche che:**
  - **M lavora in tempo $f(n)$**
  - **M è una $f(n)$-MdT**

- **E' spesso estremamente complesso definire con precisione la funzione f della complessità temporale.**
- **Si considerano approssimazioni che enfatizzano l'andamento della MdT per grandi input: analisi asintotica**

Complessità asintotica: notazione O

- Definizione
- Siano due funzioni $f, g: N \rightarrow R^{+}$.
- Si scrive $f(n) = 0\big(g(\boldsymbol{n})\big)$ se
- Esistono costanti intere c e $\boldsymbol{n_0}$ tali che per ogni $n \geq n_0 \;\; f(n) \leq c \cdot g(n)$
- $g(\boldsymbol{n})$ è un limite superiore asintotico per $f(n)$

Se f(n) è descritta mediante una espressione con diversi fattori g(n) permette di considerare solo il fattore di ordine superiore senza eventuali coefficienti.

Ad esempio

$f(n) = 6\boldsymbol{n}^3 + 4\boldsymbol{n}^2 + 20$ si ha $f(n) = O(n^3)$

Classi di complessità temporale

- **Definizione**

Sia $\mathbf{t}: N \rightarrow R^{+}$ **una funzione.**

La classe di complessità TIME(t(n)) è l'insieme dei linguaggi che possono essere decisi mediante O(t(n))-MdT deterministiche.

Esempio. Si consideri il linguaggio $\boldsymbol{L = \{a^k b^k : \ k \geq 0\}}$

Sia M una MdT che decide L

1. **M controlla che non ci siano b prima di a (una scansione 2n passi, O(n))**
2. **M itera la marcatura di una a e una b (ogni iterazione richiede una scansione O(n))**
3. **Il numero di iterazioni è** $\frac{n}{2}$
4. **Complessivamente M è una O(**n^2 **)-MdT.**

Classi di complessità temporale

Esempio. Si consideri il linguaggio $L = \{a^k b^k : \ k \geq 0\}$

Sia M una MdT con due nastri che decide L

1. **M controlla che non ci siano b prima di a (una scansione e ritorno di 2n passi, O(n))**
2. **M copia a^k sul secondo nastro (una scansione e ritorno di 2n passi, O(n))**
3. **Con una sola scansione controlla a^k sul secondo nastro con b^k** sul primo
4. **Complessivamente M è una O(n)-MdT.**

E' possibile dimostrare che non esiste una O(n)-MdT con un solo nastro che accetta L.

Classi di complessità temporale

- **Considerazioni.**
- **Nel determinare la decidibilità di un linguaggio/problema è irrilevante il modello computazionale**
- Il risultato non dipende da determinismo/non determinismo o dall'uso di più nastri
- **Nel determinare la classe di complessità di un linguaggio/problema è rilevante il modello di MdT utlizzato.**
- L'uso di MdT deterministiche e non deterministiche porta alla definizione di classi di complessità distinte
- L'uso di MdT multi-nastro permette di abbassare la classe di complessità di qualche linguaggio.
- Quanto può incidere l'uso di multi-nastro nell'abbassamento della classe di complessità?

Classi di complessità temporale

- **L'uso di un multi-nastro può avere dunque un effetto al più quadratico (rispetto al nastro singolo) nel determinare la classe di complessità!**
- **TEOREMA.**
- **Sia $g(n)$ una funzione con $g(n) \geq n$. Per ogni O($g(n)$)-MdT multinastro esiste una O($g(n)^2$) −MdT a nastro singolo equivalente.**

1. **Si usa la simulazione della MdT M' con un nastro della MdT a k nastri già usata nella prova di equivalente espressività dei due modelli.**
2. **I k nastri vengono sequenzializzati sul singolo nastro di M' opportunamente delimitati da un simbolo separatore**

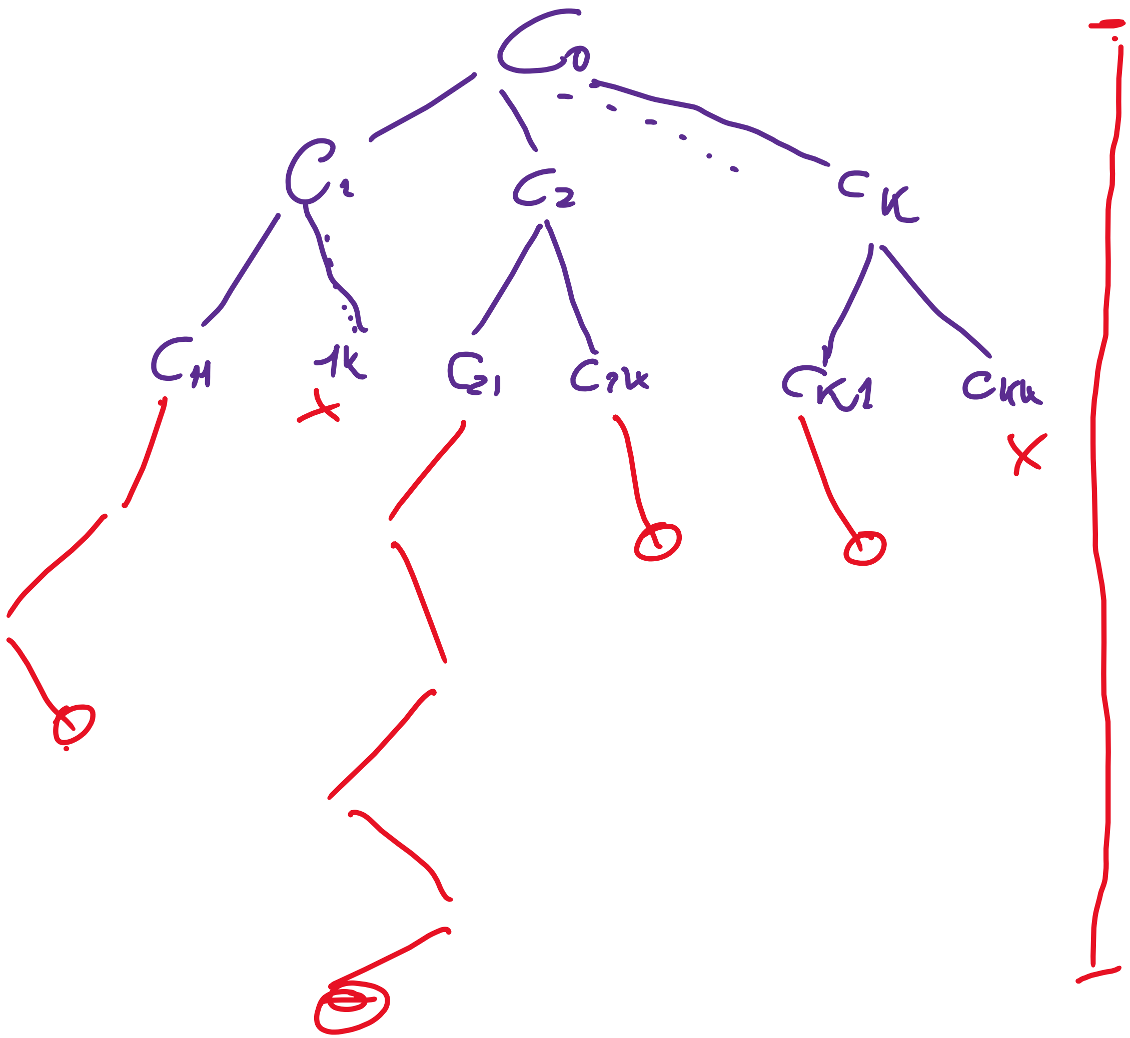
Classi di complessità temporale

1. Per simulare un passo di M,
    1. M' scandisce il nastro per raccogliere l'informazione utile per la transizione (posizioni correnti sui nastri)
    2. Aggiorna informazioni della testine e dei nastri
    3. Implementa operazioni di shift di una posizione se viene richiesta l'estensione di qualche nastro (al più k estensioni)
2. Osservazione:
3. Poiché M opera in tempo O(g(n)) la parte occupata di ciascun nastro può essere al più O(g(n)).
4. Quindi la lunghezza del nastro di M' è al più k volte O(g(n)) e dunque O(g(n)).
5. Ogni scansione del nastro richiede dunque tempo O(g(n)).
6. Complessivamente ogni simulazione di un passo di M richiede O(g(n)).
7. La simulazione di O(g(n)) passi di M richiede tempo $O(g(n))^2$) in M'

Complessità temporale: non-determinismo

- **Quanto può incidere l'uso del non-determinismo nel determinare la complessità del problema?**
- Sappiamo che per ogni MdT non-deterministica che decide un linguaggio L è possibile trovare una MdT deterministica che decide L.
- Ricordiamo che una MdT non-deterministica che decide un linguaggio deve terminare per ogni scelta non-deterministica operata.
- Ogni ramo dell'albero delle computazioni è finito.
- **Come calcolare il tempo di esecuzione per una MdT non-deterministica?**
- E' la computazione di lunghezza massima tra quelle scelte
- (Il cammino di lunghezza massima nell'albero delle computazioni)

Complessità temporale: non-determinismo



tempo:
cammino
lunghezza
massima.

Classi di complessità temporale

- **L'uso del non-determinismo può portare ad una riduzione esponenziale nella complessità.**
- **TEOREMA.**
- **Sia $g(n)$ una funzione con $g(n) \geq n$. Per ogni O($g(n)$)-MdT non-deterministica a singolo nastro esiste una $2^{O(g(n))}$ −MdT deterministica a nastro singolo equivalente.**

1. Si usa la simulazione della MdT M' deterministica della MdT M non-deterministica già usata nella prova di equivalente espressività dei due modelli.
2. La macchina deterministica M' che simula M usa 3 nastri.
3. La lunghezza della computazione più lunga nell'albero delle computazioni è O($g(n)$)
4. **Basta dunque considerare un albero delle computazioni di profondità O($g(n)$)**

Classi di complessità temporale

1. Sia d il massimo numero di scelte non-deterministiche in una configurazione.
2. Il numero massimo di foglie (massimo numero di computazioni da simulare) $d^{g(n)}$.
3. Ogni computazione richiede tempo O($g(n)$).
4. Il tempo richiesto da M’ è dunque O($g(n)d^{g(n)}$) = $2^{O(g(n))}$.
5. M’ utilizza 3 nastri.
6. Per il teorema sulle MdT multi-nastro è possibile trovare una MdT M’’ deterministica a singolo nastro equivalente con tempo di esecuzione $(2^{O(g(n))})^2 = 2^{O(2g(n))} = 2^{O(g(n))}$

La classe di complessità P

- La classe di complessità P include tutti i linguaggi/problemi che siano decisi da una MdT deterministica che operi in tempo polinomiale
- Poiché le MdT deterministiche multi-nastro hanno solo una differenza quadratica rispetto alle MdT deterministiche a singolo nastro entrambi i modelli possono essere usati indifferentemente nella definizione della classe P.
- Non sono invece considerate le MdT non-deterministiche che hanno una possibile differenza esponenziale nei tempi di esecuzione.
- **DEFINIZIONE. P è la classe dei linguaggi che possono essere decisi da una MdT deterministica in tempo polinomiale**

$$P = \bigcup_k TIME(n^k)$$

La classe P

- **La classe P rappresenta la classe dei problemi che possono essere praticamente risolubili in modo esatto.**
- La classe è invariante a modelli computazionali che differiscono solo in modo polinomiale nei tempi di esecuzione.

Esempi di linguaggi in P

L= $\{< \boldsymbol{A} >, \boldsymbol{w} : \boldsymbol{w} \in \boldsymbol{\Sigma}^*, \boldsymbol{A}$ automa regolare deterministico accetta w$\}$

Idea.

- E' possibile scrivere una MdT che avendo $< A, w >$ sul nastro di ingresso simula l'esecuzione di A sulla parola b
- Se la simulazione termina in stato di accettazione la MdT accetta
- Se la la simulazione termina in stato di non accettazione la MdT rifiuta.
- La MdT richiede un numero di scansioni del nastro pari a m=|w|
- La scansione del nastro richiede n=| $< A >, w$ | (m < n)
- Complessivamente L $\in TIME(n^2)$

Esempi di linguaggi in P

L$= \{< \boldsymbol{G} > \boldsymbol{s}, \boldsymbol{r}: \boldsymbol{G}$ **un grafo, s ed r nodi,** $\boldsymbol{r}$ è $\boldsymbol{raggiungibile\ da\ s}\}$

- La MdT
  - Si marca lo stato s
  - Iterativamente se il nodo q è marcato e (q,a,q') è un arco si marca anche q' e si marca come utilizzato l'arco.
  - Si termina quando r è stato marcato o non è possibile marcare ulteriori nodi
- la MdT termina con accettazione se e solo se dopo la marcatura dei nodi raggiungibili r è stato marcato
- Il numero di iterazioni è limitato dal numero degli archi di un grafo (un arco viene utilizzato al più una volta) .
- Una scansione per ogni iterazione.
- L $\in TIME(n^2)$

Esempi di linguaggi in P

Sono in P i seguenti linguaggi?

1. L$= \{< G, w >: w \in \Sigma^*, G\ grammatica\ context\ free$, G genera w$\}$
2. L$= \{< G >: G$ una grammatica contex free, L$(G) = \emptyset\}$
3. L$= \{< A >: A\ automa\ regolare,\ L(A)$ è $infinito\}$
4. L$= \{< A,' A'' >: A', A''\ automi\ regolari,\ L(A') \cap L(A'') = \emptyset\}$

La classe di complessità NP

- **La classe di complessità NP include tutti i linguaggi/problemi che siano decisi da una MdT non-deterministica che operi in tempo polinomiale**
- **DEFINIZIONE.**

Sia $\mathbf{t}: N \to R^{+}$ **una funzione.**

La classe di complessità NTIME(t(n)) è l'insieme dei linguaggi che possono essere decisi mediante O(t(n))-MdT non-deterministiche.

- **DEFINIZIONE. NP è la classe dei linguaggi che possono essere decisi da una MdT deterministica in tempo polinomiale**
- $\mathbf{N}P = \bigcup_k NTIME(n^k)$

La classe NP

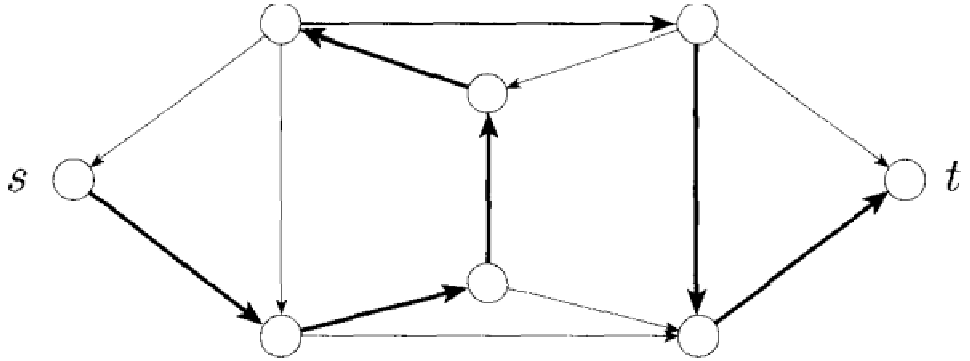
- La classe NP ovviamente include la classe P poiché una MdT deterministica è un caso particolare di MdT non-deterministica.
- La classe è invariante a modelli di MdT.
- La classe NP include problemi per i quali non è nota l'esistenza di decisori che lavorino in tempo polinomiale

Esempi di linguaggi in NP

Il problema del cammino Hamiltoniano in un grafo diretto G=<V,E>

(V è l'insieme dei vertici, E $\subseteq V \times V$ è l'insieme degli archi diretti)

Un cammino Hamiltoniano tra un vertice s e un vertice t è un cammino da s a t che include tutti i vertici V una sola volta.



Esempi di linguaggi in NP

$$L_{HPATH} = \{< G, s, t > : G\ grafo\ diretto, \exists\ \ un\ cammino\ Hamiltoniano\ da\ s\ a\ t\}$$

TEOREMA. $L_{HPATH} \in NP$

Si può decidere il linguaggio con una MdT non-deterministica che opera i seguenti passi:

1. Genera una sequenza casuale di vertici (usando la scelta non deterministica) senza ripetizioni con s come vertice iniziale e t come vertice finale che includa tutti i vertici
2. Verifica che la sequenza sia un cammino da s a t.
3. La macchina accetta se la sequenza è un cammino e rifiuta altrimenti

Il passo 1 e 2 possono essere fatti in tempo polinomiale

Esempi di linguaggi in NP

OSSERVAZIONI

- La MdT usa una tecnica generale permessa dal non-determinismo:
- **1) generazione non-deterministica di una soluzione.**
- Se una soluzione corretta al problema esiste il metodo di generazione non-deterministico la deve includere
- **2) verifica che la soluzione proposta sia accettabile.**
- Poiché il criterio di accettazione di una MdT non deterministica richiede l'esistenza di una computazione accettante, se la soluzione esiste essa sarà generata in una computazione portando ad accettazione.
- **Non si conoscono algoritmi deterministici per decidere L_{HPATH} in tempo polinomiale**

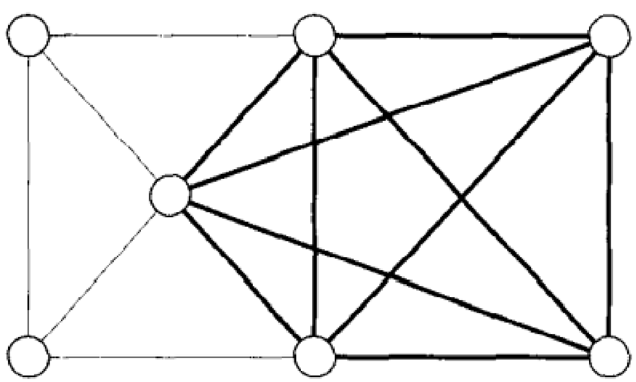
Esempi di linguaggi in NP

Il problema dell'esistenza di una k-clique in un grafo non diretto G=<V,E>

(V è l'insieme dei vertici, E $\subseteq V \times V$ è l'insieme degli archi non diretti)

Una k-clique è un sottoinsieme $S \subseteq V$ di vertici di cardinalità k tale che

- Per ogni coppia di nodi s,s' in S esiste un arco tra s e s'



una 5-clique

Esempi di linguaggi in NP

$$L_{kCLIQUE} = \{< G, k >: G\ grafo\ indiretto\ con\ una\ k - clique\}$$

TEOREMA. $L_{kCLIQUE} \in NP$

Si può decidere il linguaggio con una MdT non-deterministica che opera i seguenti passi:

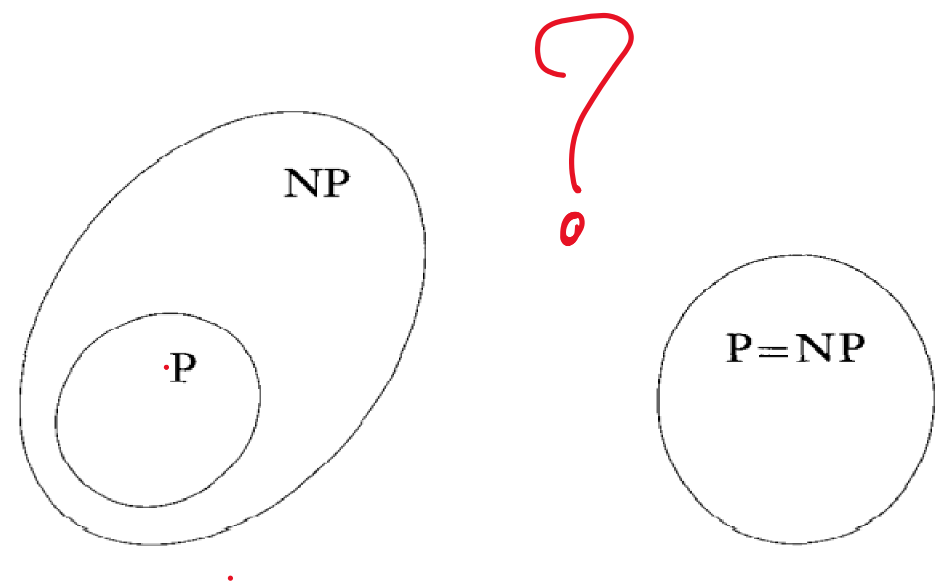
1. Genera una sequenza casuale di k vertici
2. Si verifica l'esistenza di un arco per ogni coppia di vertici scelti
3. La macchina accetta se la verifica ha successo

Il passo 1 e 2 possono essere fatti in tempo polinomiale

Confronto di P e NP

- Il ricorso al non-determinismo permette di perlustrare lo spazio delle soluzioni mediante un approccio di **forza bruta** (generare tutte le soluzioni)
- La sola richiesta è che la verifica della soluzione proposta sia in tempo polinomiale
- Nel caso deterministico la **forza bruta non è ammessa** se lo spazio delle soluzioni non è polinomiale nella dimensione dell'input.
- **P è naturalmente incluso in NP.**
- **Se, al contrario, sia $P \subset NP$ o $P = NP$ è il più significativo problema non risolto dell'informatica teorica.**

Confronto di P e NP



- Il metodo migliore conosciuto per risolvere deterministicamente problemi in NP richiede tempo esponenziale. Unica cosa certa.
- $NP \subseteq EXPTIME = \bigcup_k TIME\left(2^{n^k}\right)$

NP-completezza

- I linguaggi/problemi NP-completi sono un sottoinsieme dei linguaggi/problemi in NP che sono pienamente rappresentativi della complessità della classe.
- Intuitivamente un linguaggi/problema è NP-completo quando ogni linguaggio/problema nella classe NP può essere ricondotto ad esso.
- Per formalizzare la nozione di riconducibilità serve raffinare la nozione di riduzione già introdotta per stabilire la decidibilità
- Nel caso della decidibilità la riduzione da un linguaggio ad un altro è operata da una funzione computabile.

NP-completezza

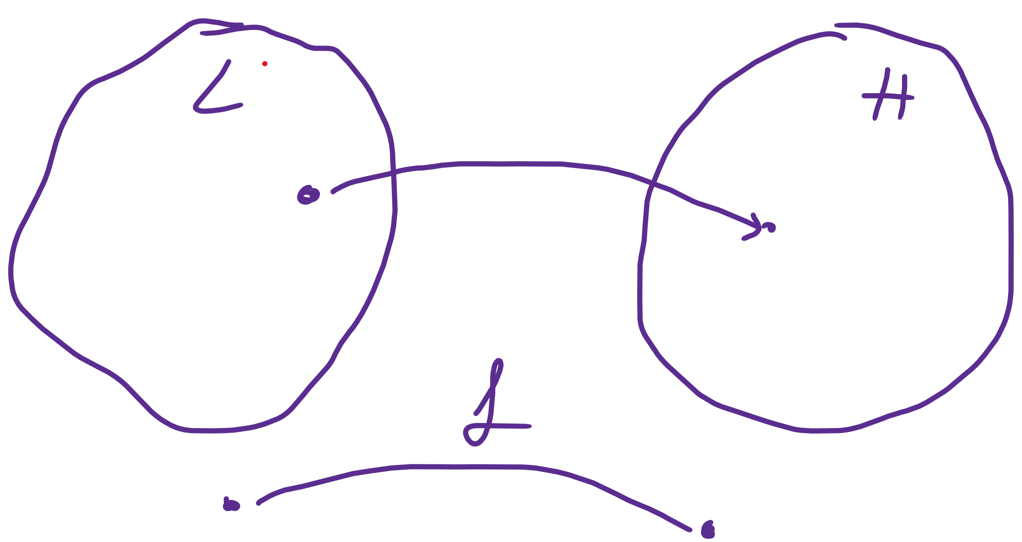
- **Funzione computabile**: Una funzione $f: \Sigma^* \to \Sigma^*$ è computabile se esiste una MdT che per ogni input $w \in \Sigma^*$ termina riportando sul nastro $f(w)$.
- Nella definizione di funzione computabile non c'è vincolo sulla complessità della funzione.
- Nello studio della complessità dei problemi serve garantire che la trasformazione del problema non alteri la classe di appartenenza del problema trasformato.
- Per questo si richiede che le trasformazioni siano operate con funzioni computabili in tempo polinomiale.

Funzione computabile in tempo polinomiale: Una funzione $f: \Sigma^* \to \Sigma^*$ è computabile in tempo polinomiale se esiste una MdT che per ogni input $w \in \Sigma^*$ termina riportando sul nastro $f(w)\ in\ tempo\ polinomiale$.

NP-completezza

- **Riducibilità in tempo polinomiale: Un linguaggio L è riducibile in tempo polinomiale ad un linguaggio H se esiste una funzione** $f: \Sigma^* \rightarrow \Sigma^*$ computabile in tempo polinomiale tale che per ogni $w \epsilon \Sigma^*$

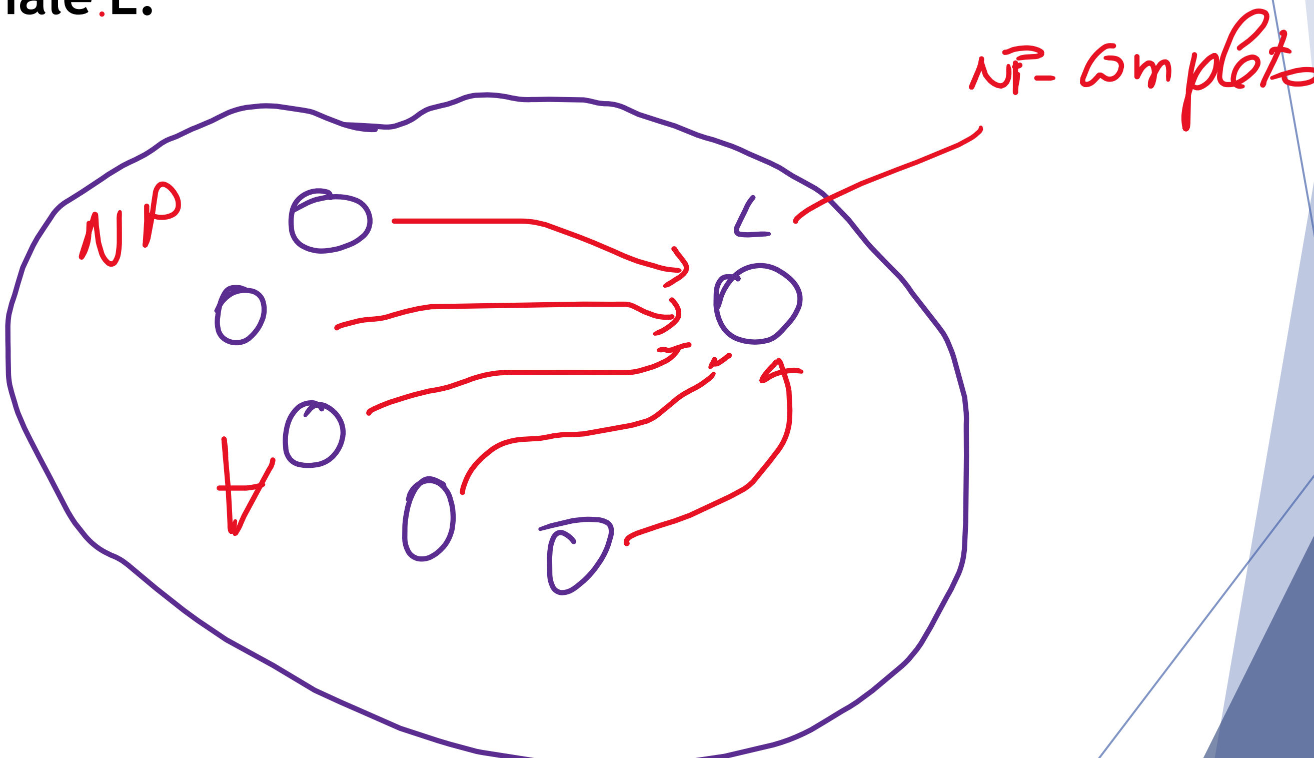
$$\boldsymbol{w \in L \iff f(w) \in H}$$



NP-completezza

- **NP-completezza: Un linguaggio L è NP-completo se**

1. **L $\in NP$;**
2. **Ogni linguaggio H $\in NP$ è riducibile in tempo polinomiale L.**



NP-completezza

- L'importanza teorica dei linguaggi NP-completi è riassunta nel seguente teorema
- **TEOREMA: Se L è un linguaggio NP-completo e L $\in \boldsymbol{NP}$ allora NP=P.**

L'importanza dei linguaggi NP-completi è sia teorica che pratica:

Teorica: per provare che P=NP o che P≠NP è sufficiente provare che un linguaggio NP-completo appartiene a P (P=NP) o che non può appartenere a P (P≠NP)

Pratica: poiché è ritenuto verosimile che P≠NP è poco pratico dedicare del tempo cercare una soluzione polinomiale per un problema NP-completo.

Un problema NP-completo

- Si conoscono esempi di problemi NP-completi? Si

Si consideri il linguaggio delle espressioni booleane.

Sia Var un insieme di variabili booleane (ad esempio, x,y)

Una formula booleana expr è definita dalla seguente sintassi

$$Expr ::= True \mid False \mid x \mid \neg Expr \mid Expr \wedge Expr \mid Expr \vee Expr$$

con x appartenente a Var.

Esempio di formule booleana.

$(\neg x \wedge y) \vee (x \wedge \neg y)$ (solo una tra x e y è vera)

$(\neg x \wedge y \wedge z) \vee (x \wedge \neg y \wedge z) \vee (x \wedge y \wedge \neg z)$ (esattamente due sono vere)

Un problema NP-completo

- Valutazione di una formula booleana φ
- La valutazione della formula booleana richiede l'assegnazione di un valore di verità 0/1 (0 = False, 1=True) alle variabili
- Sia $\rho : Var \longrightarrow \{0,1\}$ la funzione che assegna un valore di verità alle variabili booleane
- La funzione $val(\rho, \varphi)$ restituisce un valore booleano in $\{0,1\}$ ed è definita induttivamente rispetto alla struttura delle formule booleane

$val(\rho, True)$ = 1

$val(\rho, False)$ = 0

$val(\rho, x) = \rho(x)$

$val(\rho, \neg\varphi) =\neg\ val(\rho, \varphi)$ (si complementa il valore di $val(\rho, \varphi)$)

$val(\rho, \varphi' \wedge \varphi'') = val(\rho, \varphi') \wedge val(\rho, \varphi'')$ (and-logico delle due valutazioni)

$val(\rho, \varphi' \vee \varphi'') = val(\rho, \varphi') \vee\ val(\rho, \varphi'')$ (or-logico delle due valutazioni)

Un problema NP-completo

- Problema della soddisfacibilità di una formula booleana φ
- Esiste una funzione $\rho : Var \longrightarrow \{0,1\}$ che rende vera φ, cioè tale che $val(\rho, \varphi')$=1?
- In caso affermativo si dice che la formula φ è soddisfacibile.
- In caso negativo si dice che la formula φ è insoddisfacibile.
- Linguaggio L_{SAT}={< φ > : φ espressione booleana soddisfacibile}

L_{SAT} è un linguaggio NP-completo!

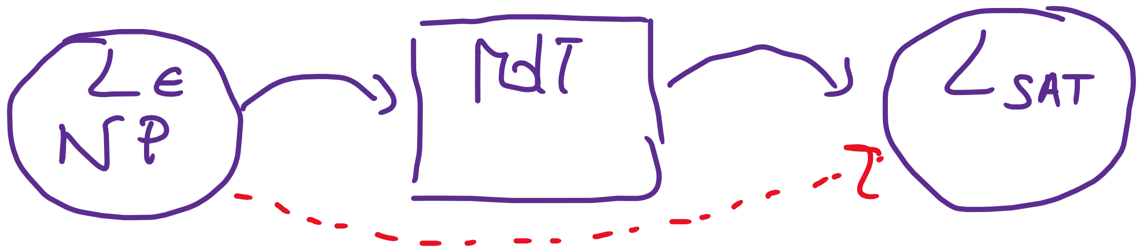
NP-completezza

TEOREMA: L_{SAT} è un linguaggio NP-completo.

Prova. Per provare che ogni linguaggio L in NP è riducibile in tempo polinomiale a L_{SAT} si usa la seguente idea:

1) Ogni linguaggio L in NP ha una MdT M non deterministica che lo decide in tempo polinomiale

2) Riducendo in tempo polinomiale la decisione in tempo polinomiale delle MdT a L_{SAT} si prova che ogni linguaggio in NP è riducibile in tempo polinomiale a L_{SAT}



Prova riduzione MdT a tempo polinomiale a L_{SAT}

- Sia M che decide un linguaggio L in tempo $O(n^k)$.
- Ogni computazione (accettante o di rifiuto) può avere al più n^k passi
- La porzione del nastro usata può essere al più di n^k celle.
- Una computazione può essere descritta da una matrice $n^k \times n^k$ dove ogni riga corrisponde a una configurazione.
- Configurazione: stringa w in $\Gamma^* \times Q \times \Gamma^*, |w| = n^k$

$Q_1 Q_2 Q_3 \ldots Q_i \, q \, Q_{i+1} \ldots Q_{n^k}$

stato q
posizione testa
i+1

- La prima riga è la configurazione iniziale
- La i+1-esima riga è la configurazione raggiunta dalla configurazione i-esima in un passo
- La computazione è accettante se lo stato associato ad una riga della tabella è q_{accept}

Formula di L_{SAT} per codificare una computazione

- Idea. Scrivere una formula booleana φ_{SAT} che è soddisfacibile se e solo se esiste una computazione accettante per la parola $a_1 \dots a_n$.
- Insieme di variabili booleane $x_{i,j,s}$:
  - $1 \leq i, j \leq n^k, s \in \Gamma$
  - Vale 1 se la cella (i,j) ha associato il simbolo s;
  - Vale 0 altrimenti

Formula di L_{SAT} per codificare una computazione

Inizializzazione della prima configurazione φ_{Init}

#	q_0	w_1	w_2	...	w_n	⊔	...	⊔	#	start configuration
#									#	second configuration

$$\varphi_{Init} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge \quad \wedge x_{1,n+3,w_n} \wedge \bigwedge_{n+4 \le j \le n^k-1} x_{1,j,\sqcup} \wedge x_{1,n^k,\#}$$

Formula di L_{SAT} per codificare una computazione

Inizializzazione colonne laterali $\varphi_\#$

#	q_0	w_1	w_2	...	w_n	⊔	...	⊔	#	start configuration
#									#	second configuration

$$\varphi_\# = \bigwedge_{2 \le i \le n^k} x_{i,1,\#} \wedge x_{i,n^k,\#}$$

Formula di L_{SAT} per codificare una computazione

Ogni cella deve essere associata ad un solo simbolo di $Q \cup \Gamma \cup \{\sqcup\}$. *Formula* φ_{Mark}

almeno un simbolo associato a ij

non più di un simbolo associato ad ij

$$\varphi_{MARK} = \bigwedge_{\substack{2 \le i \le m^k \\ 2 \le j \le m^k - 1}} \left(\bigvee_{s \in \Gamma \cup Q \cup \{\sqcup\}} x_{ijs} \right) \wedge \bigwedge_{\substack{s, s' \in \Gamma \cup Q \cup \{\sqcup\} \\ s \neq s'}} \left(\overline{x_{ijs}} \vee \overline{x_{ijs'}} \right)$$

Formula di L_{SAT} per codificare una computazione

Una riga deve avere una configurazione accettante

a $Formula\ \varphi_{Accept}$

$$\varphi_{Accept} = \bigvee_{1 \le i,j \le n^k} x_{ij\,Accept}$$

Formula di L_{SAT} per codificare una computazione

- Step. Serve scrivere dei vincoli che garantiscono che la riga i+1 è ottenuta applicando una transizione alla riga i
- Osservazione:
  - Possono cambiare solo le celle adiacenti a quella in cui è posizionata la testina. Tutte le altre celle devono restare invariate.

	$j-1$	j	$j+1$
i	Q_1	Q_2	Q_3
$i+1$		Q_2	

← non c'è testina nella questa

$$x_{ijs} \rightarrow x_{i+1\,js}$$

stesso simbolo

Formula di L_{SAT} per codificare una computazione

- **Step.** Propagazione dei simboli invariati.

$$\varphi_{Inv} = \bigwedge_{1 \le i < n^k} \bigvee_{s, s', s'' \in \Gamma \cup \{u, \#\}} x_{ijs} \wedge x_{i,j-1,s'} \wedge x_{i,j+1,s'} \longrightarrow x_{i+1,j,s}$$

$$\begin{array}{c|c|c|c|} & j-1 & j & j+1 \\ \hline i & s' & s & s'' \\ \hline i+1 & & s & \\ \hline \end{array}$$

$\leftarrow$ non c'è testina nella finestra

$x_{ijs} \longrightarrow x_{i+1,js}$

stesso simbolo

Formula di L_{SAT} per codificare una computazione

- Step. Transizioni con spostamento a destra. La transizione $(q_1, a, b, q_2, R)\epsilon\delta$.

-

s	q_1	a
s	b	q_2

$$\varphi_{step}(i,j) = \bigvee_{\substack{s\in\Gamma\cup\{\#\} \\ q_1\in Q \\ a\in\Gamma\cup\{u\} \\ (q_1,a,b,q_2,R)\in\delta}} x_{i,j,q_1} \wedge x_{i,j-1,s} \wedge x_{i,j+1,a} \rightarrow x_{i+1,j,b} \wedge x_{i+1,j-1,s} \wedge x_{i+1,j+1,q_2}$$

Formula di L_{SAT} per codificare una computazione

- Step. Transizioni con spostamento a sinistra. La transizione $(q_1, a, b, q_2, L)\epsilon\delta$.

s	q_1	a
q_2	s	b

$$\varphi_{step}(i,j) = \bigvee_{\substack{s\in\Gamma\cup\{\#\} \\ q_1\in Q \\ a\in\Gamma\cup\{u\} \\ (q_1,a,b,q_2,L)\in\delta}} x_{i,j,q_1} \wedge x_{i,j-1,s} \wedge x_{i,j+1,a} \rightarrow x_{i+1,j,s} \wedge x_{i+1,j-1,q_2} \wedge x_{i+1,j+1,b}$$

Formula di L_{SAT} per codificare una computazione

- Step. Propagazione accettazione

a	q_{accept}	b
a	q_{accept}	b

$$\varphi_{Accept}(i,j) = \bigvee_{a,b \in \Gamma \cup \{\#,\sqcup\}} x_{i,j,q_{accept}} \wedge x_{i,j-1,a} \wedge x_{i,j+1,b} \rightarrow x_{i+1,j,q_{accept}} \wedge x_{i+1,j-1,a} \wedge x_{i+1,j+1,b}$$

Formula di L_{SAT} per codificare una computazione

- Step.

$$\varphi_{STEP} \bigwedge_{\substack{1 \le i \le n^k - 1 \\ 2 \le j \le n^k - 1}} \varphi(i,j)_{stepR} \vee \varphi(i,j)_{stepL} \vee \varphi(i,j)_{ACCEPT}$$

$$\boxed{L_{SAT} = \varphi_{STEP} \wedge \varphi_{Init} \wedge \varphi_{ACCEPT} \wedge \varphi_{\#}}$$

Polinomialità della riduzione

- Serve provare che la riduzione è computabile in tempo polinomiale (rispetto a n).
- Numero delle variabili booleane:
- $O(n^{2k})$ (n^{2k} celle, $n^{2k}|\Gamma|$, $|\Gamma|$ è costante e non dipende da n)
- Dimensione per ogni parte della formula:
  - φ_{Init} ha dimensione $O(n^k)$
  - φ_{Mark} ha dimensione $O(n^{2k})$
  - $\varphi_{\#}$ ha dimensione $O(n^k)$
  - φ_{Accept} ha dimensione $O(n^{2k})$
  - φ_{Step} ha dimensione $O(n^{2k})$

Polinomialità della riduzione

- Serve provare che la riduzione è computabile in tempo polinomiale (rispetto a n).
- Numero delle variabili booleane:
- $O(n^{2k})$ (n^{2k} celle, $n^{2k}|\Gamma|$, $|\Gamma|$ è costante e non dipende da n)
- Dimensione della formula: $O(n^{2k})$
- La generazione di ogni parte della formula è dell'ordine della dimensione della formula.
- Complessivamente la riduzione è polinomiale in n

Polinomialità della riduzione

- Serve provare che la riduzione è computabile in tempo polinomiale (rispetto a n).
- Numero delle variabili booleane:
- $O\left(n^{2k}\right)$ (n^{2k} celle, $n^{2k}\,|\Gamma|$, $|\Gamma|$ è costante e non dipende da n)
- Dimensione della formula: $O\left(n^{2k}\right)$
- La generazione di ogni parte della formula è dell'ordine della dimensione della formula.
- Complessivamente la riduzione è polinomiale in n

Altri problemi NP-completi

- Il linguaggio L_{3SAT}
- Un caso speciale del problema della soddisfacibilità.
- Le formule sono vincolate a soddisfare un formato ristretto:
- 1. Forma normale congiuntiva:
- Una formula è una congiunzione di clausole

$$\bigwedge_{i=1\ldots n} C_i$$

- Una clausola è una disgiunzione di letterali

$$C_i = l_{i,1} \vee l_{i,2} \vee \ldots \vee l_{i,ki}$$

- Un letterale è una variabile o la negazione di una variabile

Forma normale congiuntiva (CNF)

- Teorema. Per ogni formula booleana φ esiste una formula booleana φ' in CNF ad essa equivalente ($\varphi \equiv \varphi'$).
- La proprietà è una conseguenza diretta delle seguenti equivalenze.

$$\varphi_1 \wedge (\varphi_2 \vee \varphi_3) \equiv (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \varphi_3)$$
$$\varphi_1 \vee (\varphi_2 \wedge \varphi_3) \equiv (\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \varphi_3)$$

$$\neg(\varphi_1 \vee \varphi_2) \equiv \neg\varphi_1 \wedge \neg\varphi_2$$
$$\neg(\varphi_1 \wedge \varphi_2) \equiv \neg\varphi_1 \vee \neg\varphi_2$$
$$\neg\neg\varphi_1 \equiv \varphi_1$$

Forma normale congiuntiva (CNF)

- Per ogni formula booleana φ esiste una formula booleana φ' in CNF ad essa equivalente ($\varphi \equiv \varphi'$).
- Purtroppo la trasformazione in CNF di una formula φ può portare in alcuni casi ad una esplosione esponenziale nella dimensione di φ.

$$(X_1 \wedge Y_1) \vee (X_2 \wedge Y_2) \vee \cdots \vee (X_n \wedge Y_n).$$

Equivalente a

$$(X_1 \vee X_2 \vee \cdots \vee X_n) \wedge (Y_1 \vee X_2 \vee \cdots \vee X_n) \wedge (X_1 \vee Y_2 \vee \cdots \vee X_n) \wedge (Y_1 \vee Y_2 \vee \cdots \vee X_n) \wedge \cdots \wedge (Y_1 \vee Y_2 \vee \cdots \vee Y_n).$$

Che contiene 2^n clausole, ogni clausola contiene o X_i o Y_i per ogni i.

Forma normale congiuntiva (CNF)

- Riduzione del numero dei letterali in una clausola
- Sfruttando la seguente equivalenza è possible limitare il numero di letterali in una clausola.

$$l_1 \vee .. \vee l_i \vee l_{i+1} \vee .. \vee l_k \quad \equiv (X \vee l_1 \vee .. \vee l_i) \wedge (\neg X \vee l_{i+1} \vee .. \vee l_k)$$

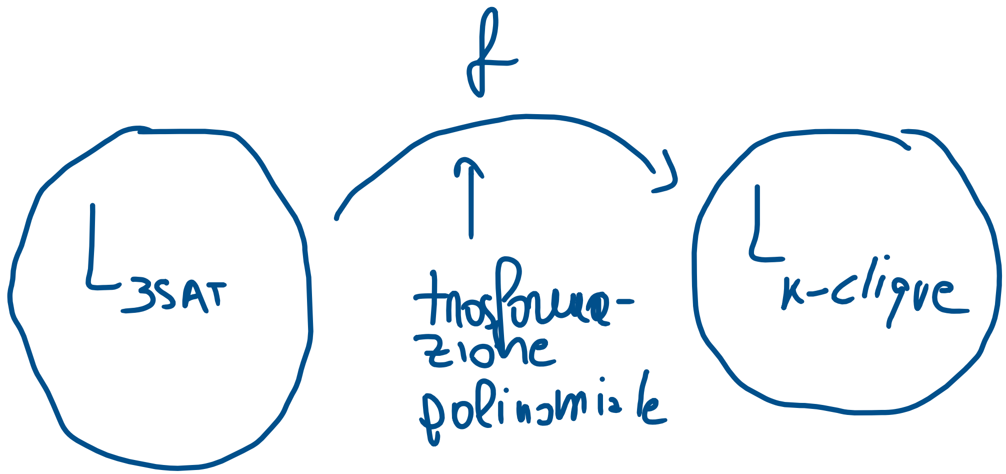
- E' possibile trasformare una clausola nella congiunzione di clausole aventi esattamente tre letterali (3CNF).
- Teorema. Per ogni formula φ esiste una formula booleana φ' in 3CNF ad essa equivalente ($\varphi \equiv \varphi'$).
- (la formula φ' potrebbe avere dimensione esponenziale nella dimensione di φ)

Altri problemi NP-complete: L_{3SAT}

- Teorema. Il linguaggio L_{3SAT} è $\mathrm{NP}-\mathrm{completo}$.
- La prova di NP$-$completezza è simile alla prova data per il linguaggio L_{SAT}.
- Si può provare che una computazione di una MdT non deterministica polinomiale può essere codificata in una formula booleana φ di 3SAT
  - φ è polinomiale nell'input
  - φ è soddisfacibile se e solo se la computazione è accettante.
- NON è possible usare la normalizzazione della formula per ridurre L_{SAT} a L_{3SAT}

Altri problemi NP-complete: k-clique

- **Teorema.** Il linguaggio $L_{HamPath}$ è $\mathrm{NP-completo}$.
- La prova di $\mathrm{NP-completezza\ si\ ottiene\ per\ riduzione\ del\ problema\ di}\ L_{3SAT}\ \mathrm{a}\ L_{k-clique}$.



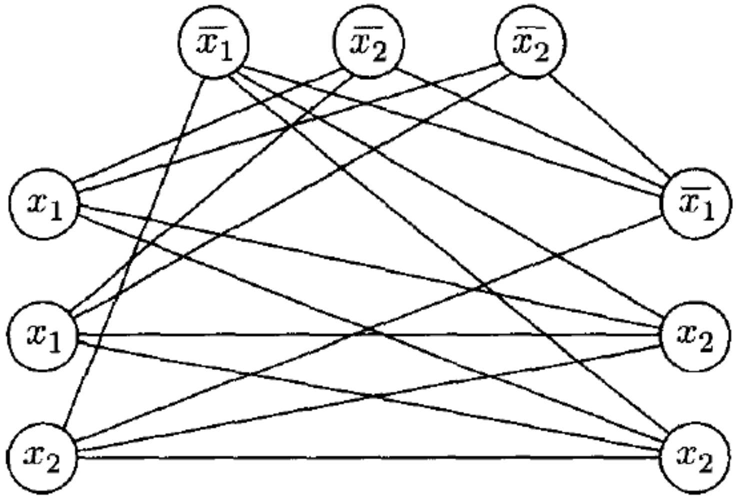
Riduzione da 3-SAT a k-clique.

- Una formula in 3CNF con k clausole.

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \quad \cdots \quad \wedge (a_k \vee b_k \vee c_k).$$

- Un nodo per ogni letterale
- Un arco per ogni coppia di letterali appartenenti a clausole diverse tranne I letterali sulla stessa variabile con segno opposto.

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$



Riduzione da 3-SAT a k-clique.

- Se esiste una K-clique.
- Contiene un nodo per ciascuna delle k clausole (non ci sono archi tra nodi della stessa clausula)
- Ogni nodo scelto corrisponde a letterale valutato a 1 (due letterali sulla stessa variabile con segno opposto non hanno arco)
- La k-clique rappresenta una valutazione della formula in 3CNF

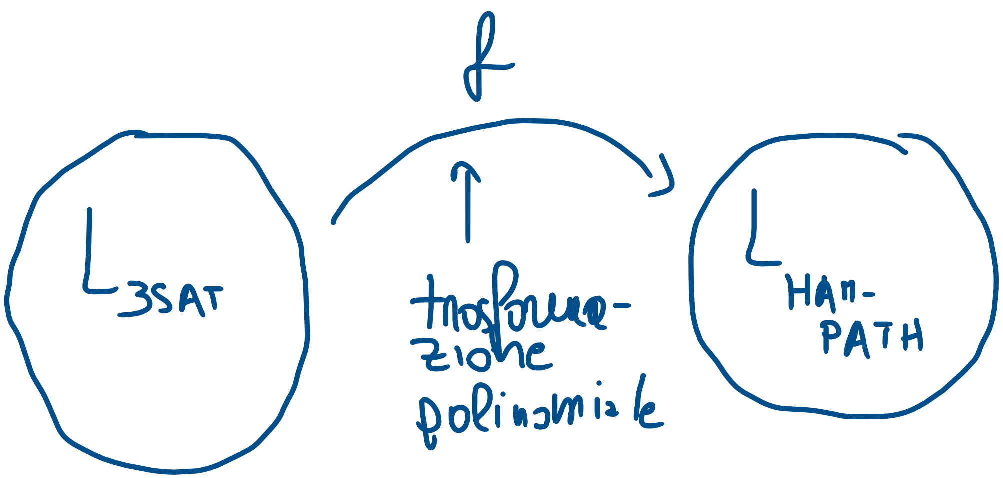
Viceversa.

Se la formula è soddisfacibile

- Scelgo un letterale che valuta a 1 per ogni clausola
- I rispettivi nodi formano una K-clique.

Altri problemi NP-complete: cammino Hamiltoniano.

- Teorema. Il linguaggio $L_{HamPath}$ è $\mathrm{NP}-\mathrm{completo}$.
- La prova di $\mathrm{NP}-$completezza si ottiene per riduzione del problema di L_{3SAT} a $L_{k-clique}$.

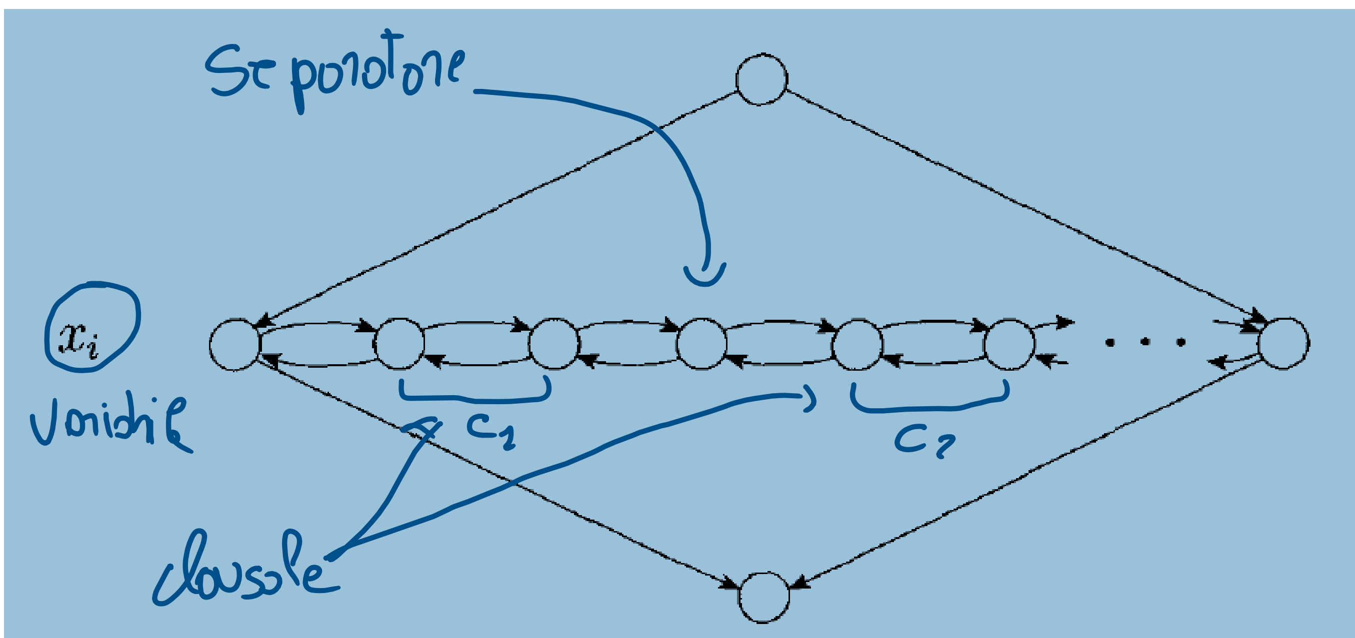


Riduzione da 3-SAT a cammino Hamiltoniano

- Una formula in 3CNF con k clausole.

$$\phi = \overbrace{(a_1 \vee b_1 \vee c_1)}^{C_1} \wedge \overbrace{(a_2 \vee b_2 \vee c_2)}^{C_2} \wedge \quad \cdots \quad \wedge \overbrace{(a_k \vee b_k \vee c_k)}^{C_k}.$$

- Variabili $x_1, \dots, x_l$
- Per ogni variabile un gadget



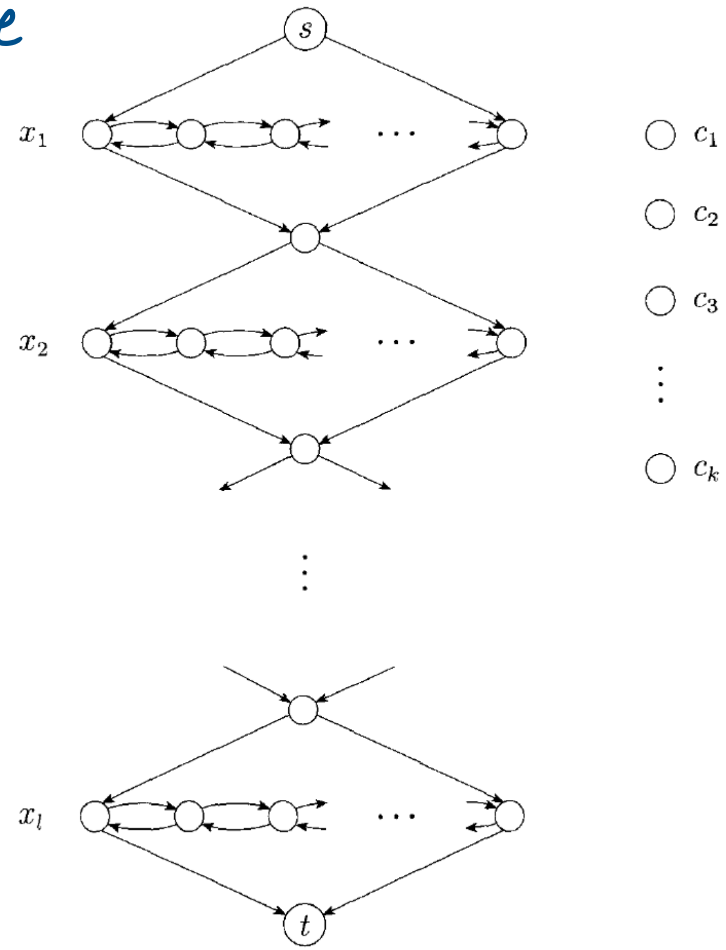
Riduzione da 3-SAT a cammino Hamiltoniano

- Una formula in 3CNF con k clausole.

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k).$$

- Variabili $x_1, \dots, x_l$

Variabili



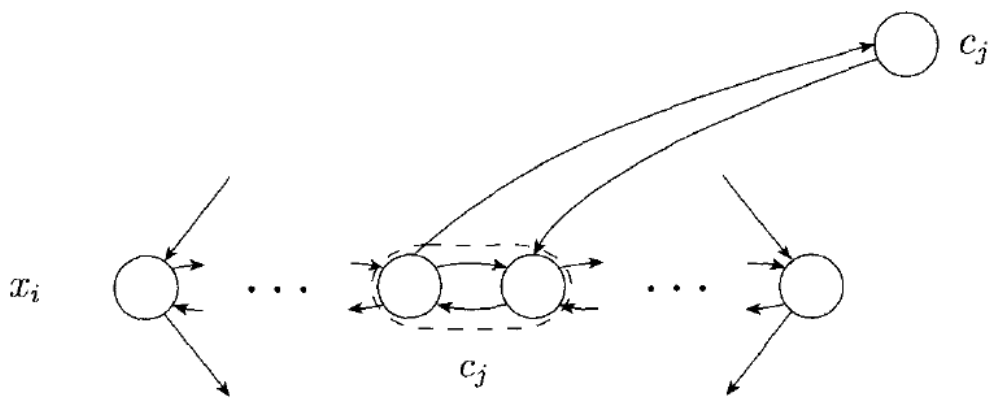
clausole

Riduzione da 3-SAT a cammino Hamiltoniano

- Una formula in 3CNF con k clausole.

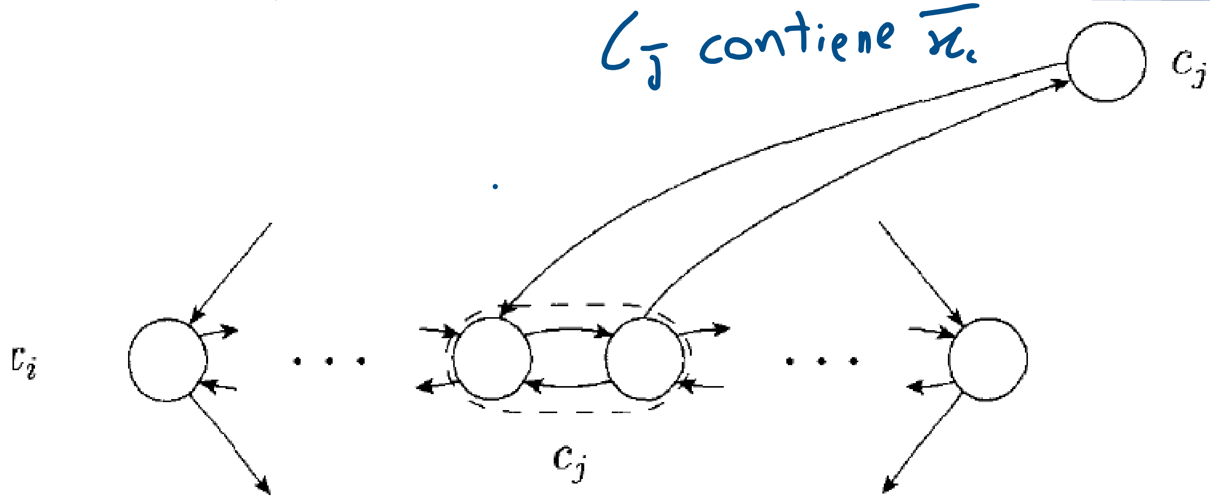
$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \quad \cdots \quad \wedge (a_k \vee b_k \vee c_k).$$

- Variabili $x_1, \dots, x_\ell$



c_j contiene x_i

c_j contiene $\overline{x_i}$



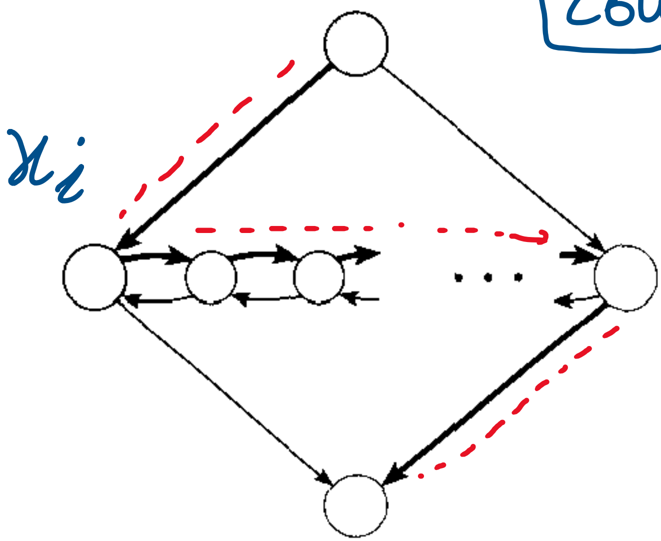
Riduzione da 3-SAT a cammino Hamiltoniano

- Una formula in 3CNF con k clausole.

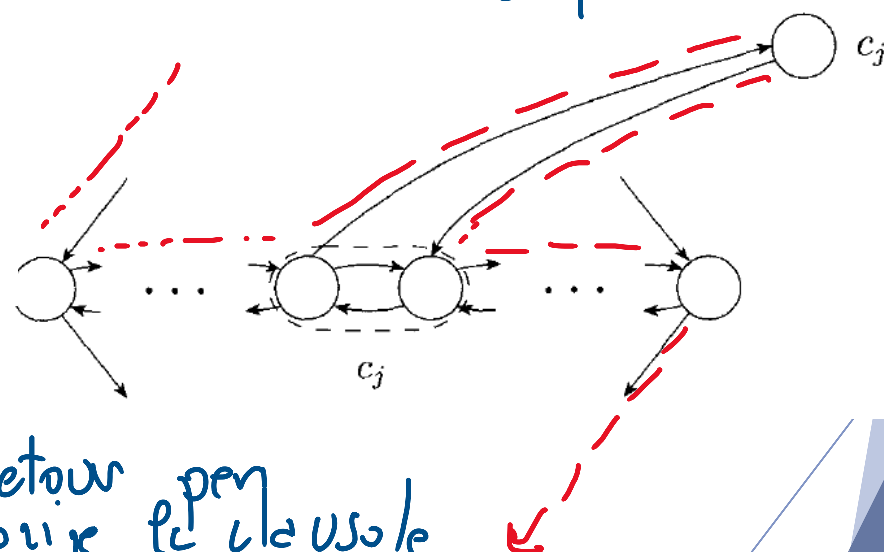
$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \quad \cdots \quad \wedge (a_k \vee b_k \vee c_k).$$

- Variabili $x_1, \ldots, x_\ell$

$\boxed{\text{Cammino con } x_i = 1}$



x_i positiva in C_j



detour per coprire le clausole

Riduzione da 3-SAT a cammino Hamiltoniano

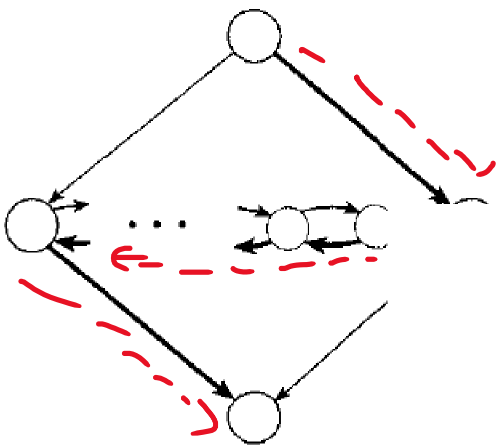
- Una formula in 3CNF con k clausole.

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \quad \cdots \quad \wedge (a_k \vee b_k \vee c_k).$$

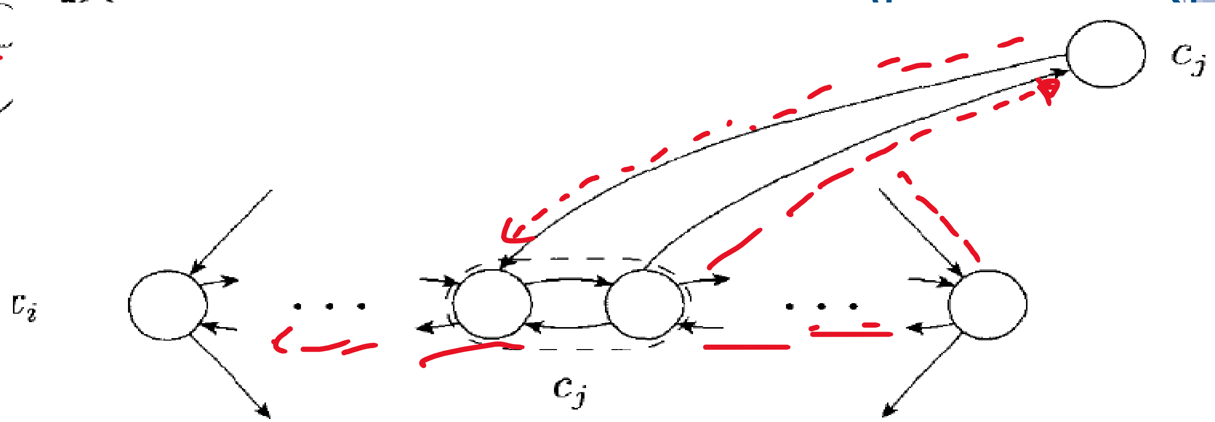
- Variabili $x_1, \dots, x_\ell$

$\boxed{\text{Cammino con } x_i = 0}$

x_i



x_i negata in C_j



detour per coprire le clausole

Riduzione da 3-SAT a cammino Hamiltoniano

- Una formula in 3CNF con k clausole.

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \quad \cdots \quad \wedge (a_k \vee b_k \vee c_k).$$

- Variabili $x_1, \dots, x_\ell$

La formula è soddisfacibile.

- L'assegnazione dei valori di verità determina un cammino nella struttura secondo la convenzione indicate.
- Per ogni clausola deve essere fatto un detour
- Il cammino è un cammino Hamiltoniano.

Esiste un cammino Hamiltoniano

- Il verso di percorrenza del cammino sulle strutture associate alle variabili indica il valore di verità delle variabili
- Tutii I nodi clausula sono visitati e questo garantisce che almeno un letterale in ogni clausola sia soddisfatto.