Intrattabilità

Nicola Gulmini

1 Intrattabilità (Lez. 01)

Una TM ha complessità temporale $T(n)$, $n \in \mathbb{N}$, se con un input w tale che $\|w\| = n$ impiega **massimo** $T(n)$ passi a terminare la computazione. Ci può mettere anche meno.

Un problema (linguaggio) L è nella classe $\mathcal{P}$ se $\exists$ una TM che lo riconosce con complessità $T(n)$ polinomiale.

Definizione 1. Chiamiamo $\mathcal{P}$ *classe dei problemi efficienti.*

Se $T(n)$ **non è un polinomio** allora si parla di problema *esponenziale*, anche nei casi in cui $T(n)$ non lo è davvero: è sufficiente che non sia polinomiale.

Esempio 1. $T(n) = n^{\log_2 n}$ cresce più velocemente di un polinomio, ma più lentamente di un esponenziale.

1.1 Problemi legati all'analisi della complessità

- Una TM accetta o rifiuta, mentre un generico algoritmo può restituire un oggetto in output. Riformuliamo il problema che l'algoritmo risolve, in modo che diventi un problema decisionale.

  Esempio 2. Sia il problema: trovare lo spanning tree di peso minimo in un grafo G. Esso può essere riformulato nel seguente modo: dato un grafo G, esiste uno spanning tree con peso non superiore all'intero W?

  A noi interessa dimostrare che un certo problema è difficile, cioè non polinomiale. Un problema decisionale rappresenta un limite inferiore alla complessità, ma questo non costituisce un problema. Infatti, se la versione decisionale di un problema risulta non polinomiale, allora la sua versione non decisionale sarà a maggior ragione difficile.

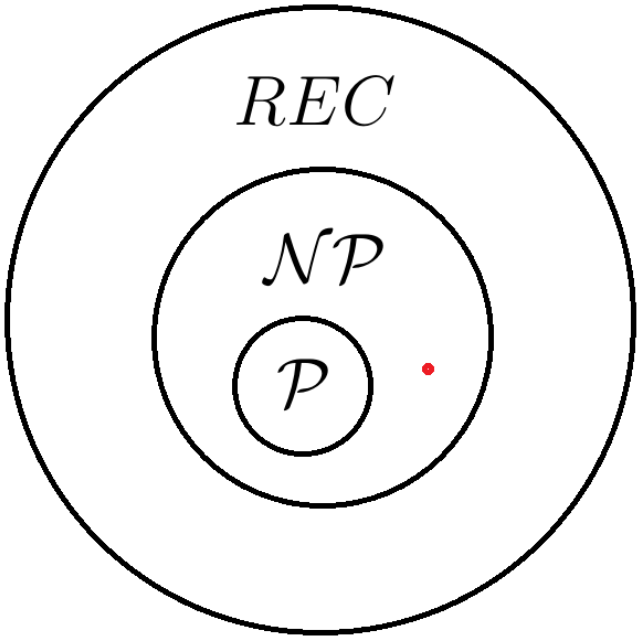
  (Se un problema in versione decisionale è $\mathcal{NP}$ allora non esiste un algoritmo polinomiale per la versione non decisionale.)

- Un algoritmo ha alfabeto potenzialmente illimitato mentre una TM ha alfabeto d'ingresso finito.

  La TM richiede la codifica di ciascun simbolo usato. Codificando i simboli in ingresso all'algoritmo si intrduce un fattore di crescita del tempo di computazione $T(n)$, che solitamente è pari al logaritmo del numero dei simboli, quindi trascurabile rispetto qualsiasi polinomio. La questione della codifica risulta pertanto irrilevante nello studio della classe $\mathcal{P}$.

Definizione 2. La classe $\mathcal{NP}$ è detta *classe dei problemi non deterministico polinomiali* (Nondeterministic Polynomial Time).

Figure 1: Il problema indicato in rosso **non** ha soluzione polinomiale, ma se $\mathcal{P} = \mathcal{NP}$ allora la distinzione tra i due insiemi collassa e tale problema ha sicuramente una soluzione polinomiale.



Un problema/linguaggio $L \in \mathcal{NP}$ se $\exists$ un polinomio $T(n)$ tale che $L = L(M)$ per una **NTM** (macchina di Turing non deterministica) M con complessità temporale $T(n)$. Questo perché una TM simula una NTM in un tempo esponenziale (vedi NTM e dimostrazione dell'equivalenza dei linguaggi accettati, in cui si esamina un numero esponenziale di strade in parallelo).

Vale $\mathcal{P} \subset \mathcal{NP}$ perché ogni TM è anche una NTM. Tipicamente si assume $\mathcal{P} \neq \mathcal{NP}$ ma ad oggi non esiste una dimostrazione: si tratta di una congettura[1].

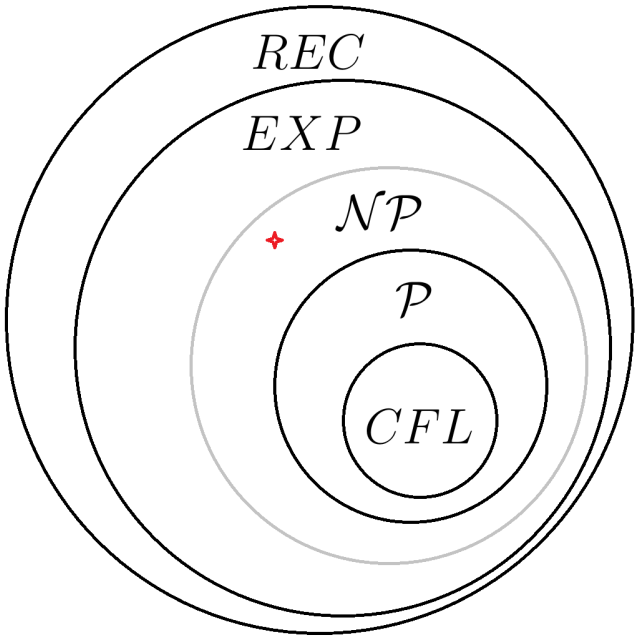
Le nostre ipotesi saranno le seguenti:

- $\mathcal{P} \neq \mathcal{NP}$;
- una NTM computa in un tempo **sempre** $T(n)$. Se ci mette meno, per esempio $T(n) - m$, $m \in \mathbb{N}$ si introducano m passi di computazione inutili per raggiungere la complessità desiderata. Inoltre, si introduce anche un *clock* simulato da un nastro aggiuntivo che gestisce la computazione ed eventualmente ferma la macchina.

Una possibile rappresentazione può essere quella in figura (1).

[1]Teoricamente potrebbe essere anche $\mathcal{P} = \mathcal{NP}$, ma ci sono validi motivi per credere che non sia così. La cosa, comunque, è basata sulla costruzione effettuata nella dimostrazione del teorema di equivalenza $L(TM) = L(NTM)$ perciò ci si riconduce al problema di determinare l'ottimalità di tale costruzione.

Figure 2: La classe dei problemi $\mathcal{NP}$ è indicata in grigio per ovvi motivi. La classe dei problemi EXP contiene tutti i linguaggi accettati da TM in tempo esponenziale. I problemi di interesse sono situati dove indicato dalla stella rossa.



2 Intrattabilità (Lez. 02)

Ricapitolando, tutti i $CFL \in \mathcal{P}$, mentre $\exists$ linguaggi $\notin \mathcal{P}$, dunque vale la rappresentazione in figura (2).

2.1 Algoritmi polinomiali non deterministici

Definizione 3 (Circuito hamiltoniano)**.** Un circuito hamiltoniano è un ordinamento dei nodi di un grafo che ne costituisce un ciclo.

In altre parole, una permutazione dei nodi di un grafo tale che esiste un ciclo che li comprenda tutti e li visiti una e una sola volta.

Esempio 3 (TSP)**.** Il problema del commesso viaggiatore (*traveling salesman problem*) ha come input un grafo G con pesi interi sugli archi e un limite di peso W. Si chiede che G abbia un circuito hamiltoniano di peso $\leq W$.

In un grafo di m nodi, il numero di cicli distinti cresce come[2] $O(m!) \in O(m^m)$: una crescita più rapida di qualsiasi esponenziale del tipo 2^{cm} per c costante. Qualunque algoritmo deterministico deve esaminare almeno un numero esponenziale di cicli e calcolare il peso di ciasuno di esso.

Un algoritmo non deterministico sceglie (non deterministicamente) un ordinamento dei nodi del grafo, verifica se essi costituiscono un ciclo, ne calcola il peso e lo confronta con W, il che richiede un tempo polinomiale: il problema è nella classe $\mathcal{NP}$.

Da questo esempio si apprezza la struttura tipica di un problema $\in \mathcal{NP}$:

1. parte non deterministica in cui si posiziona in uno spazio di ricerca esponenziale;
2. verifiche polinomiali.

[2]Essendo una permutazione dei nodi, un algoritmo che la cerca è sicuramente O(numero di permutazioni possibili) $\in O(m!)$.

2.2 Riduzioni polinomiali

La riduzione, strumento che useremo spesso in questo capitolo, consiste in un algoritmo che mappa istanze di un problema, in istanze di un altro problema, con precisi criteri. É richiesto, però, che tale trasformazione **avvenga in tempo polinomiale**: il passaggio non deve avvenire con troppo *sforzo* computazionale, altrimenti non avrebbe senso se la riduzione dovesse coinvolgere problemi polinomiali.

Teorema 1. Se $P_1 \leq P_2$ e $P_1 \notin \mathcal{P} \Rightarrow P_2 \notin \mathcal{P}$.

Proof. Se P_2 fosse polinomiale, allora potrei risolvere in tempo polinomiale anche P_1, il che è assurdo per l'ipotesi $P_1 \notin \mathcal{P}$. □

Definizione 4 (NP-completezza)**.** Un linguaggio L è NP-completo se:

- $L \in \mathcal{NP}$;
- per ogni linguaggio $L' \in \mathcal{NP}$ vale $L' \leq L$.

I problemi NP-completi sono i più difficili tra quelli in $\mathcal{NP}$. Se $\mathcal{P} \neq \mathcal{NP}$, allora i problemi NP-completi sono in $\mathcal{NP}/\mathcal{P}$.

Teorema 2. Se P_1 è NP-completo, $P_2 \in \mathcal{NP}$ e $P_1 \leq P_2 \Rightarrow P_2$ è NP-completo.

Proof. **La riduzione polinomiale è transitiva**: $\forall L \in \mathcal{NP}$ vale $L \leq P_1$ e $P_1 \leq P_2$, quindi $L \leq P_2$, che è la definizione di NP-completezza. □

Teorema 3. Se un problema NP-completo è in $\mathcal{P}$, allora $\mathcal{P} = \mathcal{NP}$.

Proof. Se P è NP-completo ed è in $\mathcal{P}$, allora $\forall L \in \mathcal{NP}$ vale $L \leq P$, il che significa che possiamo risolvere qualsiasi problema in $\mathcal{NP}$ in tempo polinomiale. □

Se $\mathcal{P} \neq \mathcal{NP}$ allora la dimostrazione di NP-completezza di un problema equivale alla dimostrazione che tale problema $\notin \mathcal{P}$.

Definizione 5 (NP-hard)**.** Un linguaggio L è NP-difficile se, $\forall L' \in \mathcal{NP}$ vale $L' \leq L$.

Cosa cambia dalla NP-completezza? Che in questo caso non è richiesto che $L \in \mathcal{NP}$! In pratica L potrebbe essere *più difficile*: alcuni problemi NP-hard richiederebbero tempo esponenziale anche se $\mathcal{P} = \mathcal{NP}$.

2.3 Soddisfacibilità delle espressioni booleane (no dim, solo idea alla base)

Decidere se un'espressione booleana è soddisfacibile è un problema NP-completo. É il primo problema NP-completo che vediamo, dunque dobbiamo **esplicitamente ridurre ciascun problema** di $\mathcal{NP}$ a questo.

Definizione 6 (Espressione booleana: sintassi)**.** Una BE (*boolean expression*) è costruita a partire dai seguenti simboli:

- un insieme **illimitato** (per esempio $\{x, y, z, x_1, y_1, \dots\}$) di variabili sui valori booleani 1 (vero) e 0 (falso);
- operatori binari $\wedge$ (AND logico) e $\vee$ (OR logico);
- operatore unario $\neg$ (NOT logico);

- parentesi tonde (per forzare le precedenze).

Definizione 7 (Espressione booleana: semantica)**.** Siano E_1 e E_2 BE. Una BE E è definita ricorsivamente come:

- $E = x$, per una qualsiasi variabile booleana x;
- $E = E_1 \vee E_2$, $E = E_1 \wedge E_2$;
- $E = \neg E_1$;
- $E = (E_1)$.

La precedenza degli operatori è la seguente, in ordine decrescente: $\neg$, $\wedge$, $\vee$.

Definizione 8 (Assegnamento di valori di verità)**.** Un assegnamento di valori di verità T per una BE E assegna il valore $T(x)$ (vero o falso) a ciascuna variabile x in E.

Definizione 9 (Valore di una BE rispetto a T)**.** Il valore $E(T)$ di E rispetto a T è il risultato della valutazione di E con ciascuna variabile x sostituita da $T(x)$.

Definizione 10 (Soddisfacibilità)**.** T soddisfa E se $E(T) = 1$. E è soddisfacibile se esiste almeno un T che soddisfa E.

Esempio 4. $x \wedge \neg(y \vee z)$ è soddisfacibile: $T(x) = 1, T(y) = 0, T(z) = 0$.

Esempio 5. $E = x \wedge (\neg x \vee y) \wedge \neg y$ **non** è soddisfacibile: deve essere $T(x) = 1$ e $T(y) = 0$ ma allora $(\neg x \vee y)$ è falso, e con lui tutta E.

Definizione 11 (SAT)**.** Il problema della soddisfacibilità è definito come segue:

- l'input è una BE E;
- l'output è una risposta *yes* se E è soddisfacibile, oppure *no* altrimenti.

3 Intrattabilità (Lez. 03)

Abbiamo introdotto il problema SAT che, se visto come linguaggio, consiste nell'insieme L_{SAT} che contiene tutte le BE soddisfacibili.

3.1 Codifica delle espressioni booleane

Rinominiamo le variabili nel seguente modo:

$$x, y, \ldots \to x_1, x_2, \ldots, x_i \to x1, x10, \ldots, xi|_2$$

(cioè il simbolo x seguito dalla rappresentazione binaria del suo indice) per poter limitare l'alfabeto[3] a $\{\wedge, \vee, \neg, (,), 0, 1, x\}$.

Esempio 6.

$$x\neg(y \vee z) \to x_1\neg(x_2 \vee x_3) \to x1\neg(x10 \vee x11)$$

Teorema 4 (Cook)**.** SAT è NP-completo.

Idea di dimostrazione. La dimostrazione si divide in due parti:

1. è necessario dimostrare prima che SAT $\in \mathcal{NP}$. Per fare questo è sufficiente dimostrare che esiste una NTM che risolve SAT in tempo polinomiale:

   - prima verifica che la sintassi è corretta, in tempo polinomiale;
   - genera non deterministicamente la tabella di verità della data BE (questo è lo spazio esponenziale, si veda la struttura tipica dei problemi NP-completi);
   - controlla in tempo polinomiale deterministico che $E(T) = 1$.

2. É necessario dimostrare che $\forall L' \in \mathcal{NP}$ vale $L' \leq$ SAT.

   La riduzione mappa qualsiasi istanza $w \in L'$ in un'istanza $enc(E)$ del problema SAT, dove $enc(E)$ è la codifica prima citata di una BE. Si fissano:

   - $L \in \mathcal{NP}$, $w \in L$ tale che $\|w\| = n$;
   - NTM M tale che $L(M) = L$. M processa w in un numero $p(n)$ di passi, con un nastro semi-infinito (sul quale non può scrivere blank B) e computazioni lunghe $p(n) + 1$ per ogni input (come accennato nell'introduzione di un nastro che simula il clock della TM).

   Sia quindi

   $$\alpha_0 \vdash \alpha_1 \vdash \cdots \vdash \alpha_{p(n)}$$

   una computazione lunga $p(n)+1$, che può essere messa in una matrice $(p(n)+1)\times(p(n)+1)$, si veda la figura (3). L'idea è quella di verificare la validità della computazione facendo operazioni di $\wedge$ e $\vee$ tra le celle della matrice, che sono $(p(n) + 1)^2$, quindi in numero polinomiale. In particolare la $\vee$ è il punto di partenza per la riduzione.

□

[3] Il libro (e quindi anche le slide) dice che una BE con m posizioni ha un numero $O(m)$ di variabili, che è una frase un po' ambigua...

Figure 3: Esiste una matrice di questo tipo per ognuna delle esponenziali computazioni possibili.

ID	0	1	$\cdots$	$j-1$	j	$j+1$	$\cdots$	$p(n)$
α_0	X_{00}	X_{01}						$X_{0,p(n)}$
α_1	X_{10}	X_{11}						$X_{1,p(n)}$
$\vdots$								
α_i				$X_{i,j-1}$	$X_{i,j}$	$X_{i,j+1}$	$\cdots$	
α_{i+1}				$X_{i+1,j-1}$	$X_{i+1,j}$	$X_{i+1,j+1}$	$\cdots$	
$\vdots$								
$\alpha_{p(n)}$	$X_{p(n),0}$	$X_{p(n),1}$						$X_{p(n),p(n)}$

4 Intrattabilità (Lez. 04)

Il teorema di Cook è utile perché tutte le dimostrazioni di NP-completezza ne fanno uso. Per semplificare la cosa, definiamo una versione ristretta di SAT.

Definizione 12 (Letterale)**.** Un letterale è una variabile x oppure una variabile negata $\bar{x} = \neg x$.

Definizione 13 (Clausola)**.** Una clausola è una disgiunzione (OR) di letterali.

Definizione 14 (CNF, conjunctive normal form)**.** Una BE in forma normale congiuntiva è una congiunzione (AND) di clausole.

Utilizziamo $+$ al posto di $\vee$ e $\times$ oppure nulla a posto di $\wedge$.

Esempio 7. Per esempio:

- $(x \vee \bar{y}) \wedge (\bar{x} \vee z)$ diventa $(x + \bar{y})(\bar{x} + z)$;
- $(x + y\bar{z})(\bar{x} + y + z)$ non è in CNF per colpa di $y\bar{z}$;
- xyz è in CNF in quanto congiunzione di clausole.

Definizione 15 (k-CNF)**.** Una BE è in forma normale k-congiuntiva se:

- è una CNF;
- ogni clausola ha esattamente k letterali.

Esempio 8. $(x + \bar{y})(\bar{x} + z)$ è in 2-CNF.

Introduciamo due nuovi problemi di decisione:

- CSAT: una BE CNF è soddisfacibile?
- kSAT: una BE k-CNF è soddisfacibile?

Teorema 5. CSAT è NP-completo.

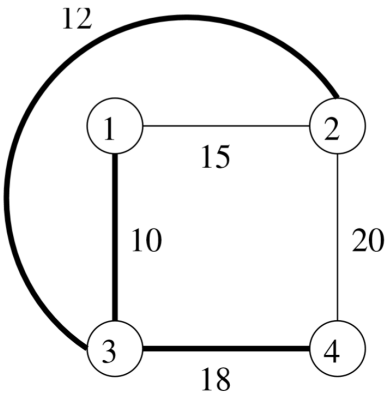
Proof. CSAT $\in \mathcal{NP}$, SAT $\leq$ CSAT. □

Teorema 6. 3SAT (k-SAT con $k = 3$) è NP-completo.

Proof. 3SAT $\in \mathcal{NP}$, CSAT $\leq$ 3SAT. □

Ora vediamo un altro problema NP-completo.

Figure 4: $I = \{1, 4\}$ è un insieme indipendente e massimale: ogni altra coppia di nodi diversa da $1, 4$ è connessa da un lato.



4.1 Insieme indipendente

Definizione 16 (Insieme indipendente (IS))**.** In un grafo G, un sottoinsieme I di nodi è detto indipendente se nessuna coppia di nodi in I è connessa da un lato di G.

Definizione 17 (Massimale)**.** Un insieme indipendente è massimale se ogni altro insieme indipendente di G ha un numero di nodi minore o uguale.

Vediamo il problema decisionale: dato un grafo G, ha un insieme indipendente di almeno $k \in \mathbb{N}$ nodi? Con k piccolo il problema è facile, ma avvicinandosi al massimale il problema diventa difficile.

Teorema 7. IS è NP-completo.

Proof. La dimostrazione è divisa in due parti, come sempre: la prima mostra una NTM che accetta il linguaggio che contiene la codifica delle istanze del problema; la seconda la riduzione da 3SAT:

1. IS $\in \mathcal{NP}$: consideriamo una NTM che sceglie arbitrariamente k nodi usando il non determinismo e che verifica in tempo polinomiale che l'insieme scelto sia indipendente.

2. 3SAT $\leq$ IS. Sia $E = e_1 \wedge e_2 \wedge \cdots \wedge e_m$ una BE in 3-CNF, dove ciascun $e_{i \in \{1,m\}}$ è una clausola. Costruiamo G con $3m$ nodi. Ciascun nodo è identificato da una coppia $[i, j]$ con $1 \leq i \leq m$ e $j \in \{1, 2, 3\}$. Il nodo $[i, j]$ rappresenta il j-esimo letterale della i-esima clausola. Si veda la figura (5) per un esempio. La costruzione del grafo G avviene nel seguente modo:

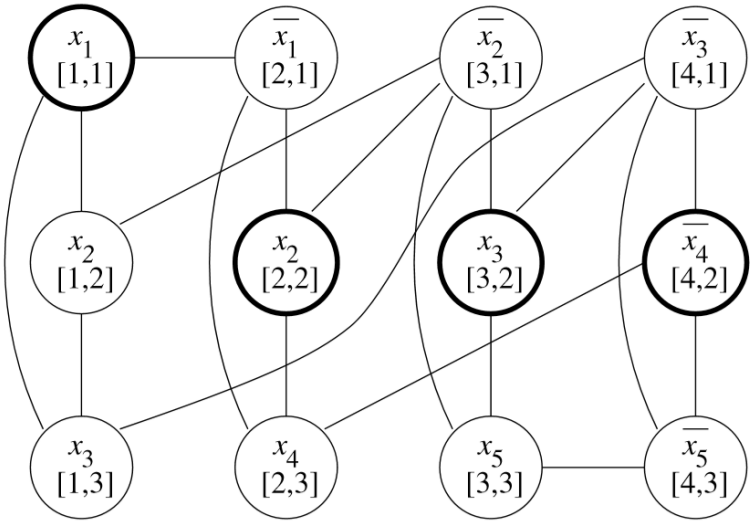
   - un lato per ciascuna coppia di nodi nella stessa colonna: si può scegliere al massimo un nodo per clausola;
   - un lato per ciascuno coppia di nodi $[i_1, j_1]$, $[i_2, j_2]$, se questi rappresentano i letterali x e $\bar{x}$: non si possono scegliere letterali complementari per lo stesso insieme indipendente[4].

   Si pone infine $k = m$. Tutto avviene in tempo polinomiale (quadratico) nella lunghezza della rappresentazione di E. E è soddisfacibile $\iff$ G ha un insieme indipendente con m elementi.

□

[4]Ogni colonna rappresenta una clausola. Almeno un letterale per ogni clausola deve essere vero. Si prende un nodo per ogni colonna e non si possono prendere quelli che sono connessi.

Figure 5: $E = (x_1 + x_2 + x_3)(\bar{x}_1 + x_2 + x_4)(\bar{x}_2 + x_3 + x_5)(\bar{x}_3 + \bar{x}_4 + \bar{x}_5)$



5 Intrattabilità (Lez. 05 - ultima del corso)

Importante: il non determinismo delle NTM è implementato con l'OR, nel senso *scegli questa strada oppure questa, oppure questa...*

Vediamo ora un altro paio di problemi NP-completi.

5.1 Copertura

In un grafo G, un sottoinsieme C di nodi è una copertura per nodi se ogni lato di G ha almeno un estremo in C. Una copertura è detta minimale se ogni altra copertura per nodi di G ha un numero di nodi maggiore o uguale. Il problema decisionale può essere formulato come segue: il grafo G ha una copertura per nodi con non più di $k \in \mathbb{N}$ nodi?

Teorema 8. NC (node cover) è NP-completo

Proof.

1. NC $\in \mathcal{NP}$. Si considera una NTM che sceglie arbitrariamente k nodi usando il non determinismo. Verifica in tempo polinomiale che l'insieme scelto sia una copertura per nodi.

2. IS $\leq$ NC. Sia G, k una istanza di IS, e sia n il numero di nodi di G. Trasformiamo in una istanza di NC formata da G e $n - k$ in tempo polinomiale. G ha un insieme dipendente con k elementi $\iff$ G ha una copertura per nodi con $n - k$ elementi (dimostrazione nelle slide, vista molto velocemente).

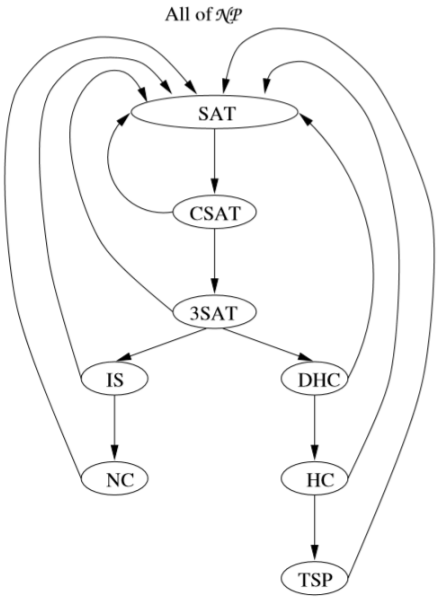
□

5.2 Circuito hamiltoniano

- DHC: dato un grafo, dire se ha un circuito hamiltoniano orientato. Questo è un problema NP-completo e si dimostra per riduzione da 3SAT;

- HC: lo stesso problema ma con un circuito hamiltoniano non orientato, è NP-completo e la riduzione si opera da DHC.

In figura (5.2) si trova un riepilogo delle riduzioni viste o accennate.


All of $\mathcal{NP}$
SAT
CSAT
3SAT
IS
DHC
NC
HC
TSP