# Lezione di Informatica Teorica: Oracoli e Gerarchia Polinomiale

Appunti da Trascrizione Automatica

30 giugno 2025

## Indice

# 1 Introduzione: Ripasso e Problemi Difficili

Ricapitoliamo il problema del **Vertex Cover (VC)**.

**Definizione 1** (Vertex Cover). *Dato un grafo* $G = (V, E)$*, un Vertex Cover* $VC$ *è un sottoinsieme di vertici* $V' \subseteq V$ *tale che per ogni arco* $(u, v) \in E$*, almeno uno tra* $u$ *e* $v$ *appartiene a* $V'$*.*

Il problema decisionale associato, che denotiamo $VC$, è:

**Definizione 2** (Problema Decisionale VC). $VC = \{\langle G, k\rangle \mid G$ *è un grafo e ammette un Vertex Cover di taglia al più* $k\}$.

Il problema $VC$ è **NP-completo**. Questo significa che un'istanza $\langle G, k\rangle$ appartiene a $VC$ se e solo se esiste un certificato conciso (un $V'$ di taglia $\leq k$) verificabile in tempo polinomiale.

## 1.1 Richiamo sulle Classi NP e Co-NP

**Definizione 3** (NP). *La classe NP (Non-deterministic Polynomial time) contiene i linguaggi per cui le istanze "sì" hanno un certificato conciso e verificabile in tempo polinomiale.*

**Definizione 4** (Co-NP). *La classe Co-NP contiene i linguaggi* $L$ *tali che il loro complemento* $\overline{L}$ *appartiene a NP. Intuitivamente, per i problemi in Co-NP, le istanze "no" hanno un certificato conciso e verificabile in tempo polinomiale.*

## 1.2 Il Problema Min-Cover

Introduciamo un problema decisionale leggermente diverso: **Min-Cover**.

**Definizione 5** (Min-Cover). $Min - Cover = \{\langle G, k\rangle \mid G$ *è un grafo e il Vertex Cover di taglia minima ha taglia esattamente* $k$

A differenza di $VC$, $Min - Cover$ richiede che la taglia sia *esattamente* $k$ e che sia la taglia *minima*.

### 1.2.1 Complessità di Min-Cover

* **Sta in NP?** Se abbiamo un certificato $V'$ di taglia $k$, possiamo verificare che sia un Vertex Cover. Ma come verifichiamo che sia *minimo*? Non possiamo, a meno di testare tutte le possibilità più piccole, che porterebbe a un tempo esponenziale. Sembra non essere in NP. * **Sta in Co-NP?** Per dimostrare che un'istanza $\langle G, k\rangle$ è un "no" (ovvero che la taglia minima non è $k$), dovremmo fornire un certificato conciso. Se la taglia minima fosse $< k$, potremmo fornire un $V'$ di taglia inferiore. Ma se la taglia minima fosse $> k$, non avremmo un modo semplice per certificarlo in Co-NP. Sembra non essere in Co-NP. * **Sta in PSPACE?** Sì. Possiamo generare tutti i possibili sottoinsiemi di $V$, verificare per ciascuno se è un Vertex Cover e calcolarne la taglia. Teniamo traccia della taglia minima trovata. Poiché i sottoinsiemi possono essere generati e testati uno alla volta riutilizzando lo stesso spazio, questo può essere fatto in spazio polinomiale.

### 1.2.2 Un Algoritmo per Min-Cover

Osserviamo che $Min - Cover$ può essere riformulato in termini di $VC$: $\langle G, k\rangle \in Min - Cover \iff \langle G, k\rangle \in VC$ AND $\langle G, k-1\rangle \notin VC$.

Questo significa che per decidere $Min - Cover$, possiamo fare due domande a un decisore per $VC$. Consideriamo il seguente pseudocodice:

```python
# `check_VC` è una subroutine che decide il problema VC
def solve_MinCover(G, k):
    # Domanda 1: Esiste un VC di taglia <= k?
    result1 = check_VC(G, k)

    # Domanda 2: Esiste un VC di taglia <= k-1?
    result2 = check_VC(G, k - 1)

    # La taglia minima è esattamente k se result1 è vero E result2 è falso
    return result1 and (not result2)
```

Questo algoritmo deterministico in tempo polinomiale necessita di un "aiuto" esterno da una procedura capace di risolvere $VC$.

## 2 Macchine ad Oracolo

Per formalizzare il concetto di "aiuto" o "subroutine esterna", introduciamo il modello delle **macchine ad oracolo**.

**Definizione 6** (Macchina ad Oracolo)**.** *Una macchina ad oracolo $M^L$ è una Macchina di Turing deterministica o non deterministica estesa con i seguenti componenti:*

- ***Oracle Tape (Nastro dell'Oracolo):*** *Un nastro di sola scrittura utilizzato per formulare le query all'oracolo.*
- ***Query State ($q_?$):*** *Uno stato speciale in cui la macchina può entrare dopo aver scritto una query sull'Oracle Tape.*
- ***Answer States ($q_{yes}$, $q_{no}$):*** *Due stati speciali in cui la macchina transita in un singolo passo dopo essere entrata in $q_?$. La transizione dipende dalla risposta dell'oracolo: a $q_{yes}$ se la stringa sulla query tape è un'istanza "sì" del linguaggio L, a $q_{no}$ se è un'istanza "no".*

*L'oracolo L è un linguaggio (un problema decisionale). Quando la macchina entra in $q_?$, la stringa sul nastro dell'oracolo viene valutata istantaneamente dall'oracolo L. Dopo la transizione a $q_{yes}$ o $q_{no}$, il contenuto dell'Oracle Tape viene automaticamente cancellato. Ogni query all'oracolo conta come un singolo passo di computazione per la macchina chiamante.*

La notazione $M^L$ indica che la macchina $M$ ha accesso a un oracolo per il linguaggio $L$.

## 3 Classi di Complessità con Oracolo

Le macchine ad oracolo permettono di definire nuove classi di complessità.

**Definizione 7** ($P^C$)**.** $P^C = \{L \mid L$ *può essere deciso in tempo polinomiale da una macchina ad oracolo deterministica* $M^{L'}$ *dove* $C\}$.

**Definizione 8** ($NP^C$)**.** $NP^C = \{L \mid L$ *può essere deciso in tempo polinomiale da una macchina ad oracolo non deterministica* $C\}$.

### 3.1 Min-Cover in $P^{NP}$

Basandoci sull'algoritmo precedente per $Min - Cover$: Il problema $VC$ è in NP. L'algoritmo `solve_MinCover` è deterministico e fa un numero costante (2) di query a un oracolo per $VC$ (che è un linguaggio in NP). Quindi, $Min - Cover \in P^{NP}$.

### 3.2 Relazioni tra le Classi con Oracolo

**Proposizione 1** ($NP \subseteq P^{NP}$)**.** *Sia $L \in NP$. Per dimostrare che $L \in P^{NP}$, costruiamo una macchina $M$ che utilizza un oracolo per $L$. $M$ prende l'input $x$, lo scrive sul nastro dell'oracolo, effettua una query. Se l'oracolo risponde "sì", $M$ accetta; altrimenti, $M$ rifiuta. Questa macchina è deterministica e opera in tempo polinomiale (solo 1 query + tempo per copiare input). Quindi, $L \in P^{NP}$.*

**Proposizione 2** ($CoNP \subseteq P^{NP}$)**.** *Sia $L \in CoNP$. Ciò implica che $\overline{L} \in NP$. Costruiamo una macchina $M$ che utilizza un oracolo per $\overline{L}$. $M$ prende l'input $x$, lo scrive sul nastro dell'oracolo, effettua una query. Se l'oracolo per $\overline{L}$ risponde "sì" (cioè $x \in \overline{L}$), $M$ rifiuta; altrimenti (cioè $x \notin \overline{L}$, quindi $x \in L$), $M$ accetta. Questa macchina è deterministica e opera in tempo polinomiale. Quindi, $L \in P^{NP}$.*

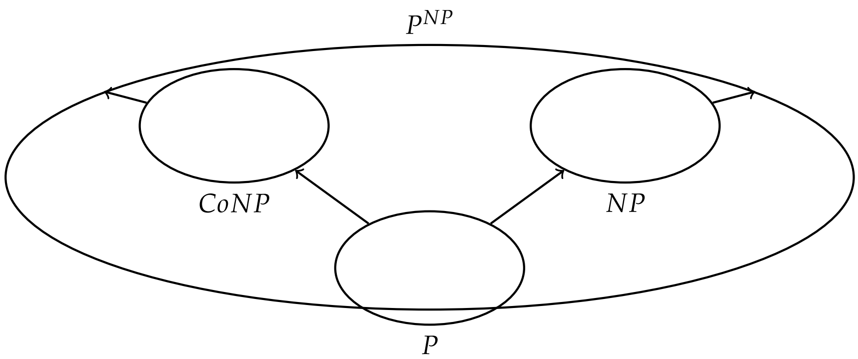


Figura 1: Relazioni iniziali tra $P$, $NP$, $CoNP$ e $P^{NP}$

## 4 Gerarchia Polinomiale

La nozione di classi ad oracolo può essere generalizzata, definendo livelli ricorsivamente. Questo porta alla **Gerarchia Polinomiale**. La gerarchia polinomiale è un insieme di classi di complessità denotate da $\Sigma_i^P$, $\Pi_i^P$, e $\Delta_i^P$ per $i \geq 0$.

**Definizione 9** (Gerarchia Polinomiale)**.** *Le classi della Gerarchia Polinomiale sono definite come segue:*

- ***Livello 0:*** $\Sigma_0^P = P$ $\Pi_0^P = P$ $\Delta_0^P = P$
- ***Livello $i+1$ (per $i \geq 0$):*** $\Sigma_{i+1}^P = NP^{\Sigma_i^P}$ $\Pi_{i+1}^P = co\Sigma_{i+1}^P$ $\Delta_{i+1}^P = P^{\Sigma_i^P}$

### 4.0.1 Esempi dei Primi Livelli:

- $\Sigma_1^P = NP^{\Sigma_0^P} = NP^P = NP$ (poiché un oracolo in P non aggiunge potere computazionale a una macchina NP, che già può simulare P).
- $\Pi_1^P = co\Sigma_1^P = coNP$.
- $\Delta_1^P = P^{\Sigma_0^P} = P^P = P$.
- $\Delta_2^P = P^{\Sigma_1^P} = P^{NP}$. (Il problema $Min - Cover$ visto prima si colloca precisamente in $\Delta_2^P$).
- $\Sigma_2^P = NP^{\Sigma_1^P} = NP^{NP}$.
- $\Pi_2^P = co\Sigma_2^P = coNP^{NP}$.

## 4.1 Relazioni di Inclusione nella Gerarchia Polinomiale

**Proposizione 3** (Inclusioni Fondamentali)**.** *Per ogni $i \geq 0$:*

1. $\Sigma_i^P \subseteq \Delta_{i+1}^P$*: Ogni linguaggio in $\Sigma_i^P$ può essere deciso da una macchina deterministica in tempo polinomiale che fa una sola query a un oracolo per un linguaggio in $\Sigma_i^P$ (la query è l'input stesso).*
2. $\Pi_i^P \subseteq \Delta_{i+1}^P$*: Similmente, ogni linguaggio in $\Pi_i^P$ (il cui complemento è in $\Sigma_i^P$) può essere deciso da una macchina deterministica in tempo polinomiale con un oracolo in $\Sigma_i^P$ (query il complemento dell'input e inverte la risposta).*
3. $\Delta_i^P \subseteq \Sigma_i^P$*: Una macchina deterministica in tempo polinomiale con oracolo $\Sigma_{i-1}^P$ è meno potente di una macchina non deterministica in tempo polinomiale con lo stesso oracolo.*
4. $\Delta_i^P \subseteq \Pi_i^P$*: Simile al punto 3, una macchina deterministica è meno potente di una macchina non deterministica che accetta se il complemento rifiuta.*

La gerarchia polinomiale contiene infiniti livelli, e la questione se questa gerarchia collassi (cioè se ci sia un livello $k$ tale che $\Sigma_k^P = PSPACE$) o se sia infinita è un problema aperto. Si ritiene che sia infinita. L'intera gerarchia polinomiale è contenuta in $PSPACE$. Al di sopra di $PSPACE$ troviamo $EXPTIME$, $NEXPTIME$, e infine la classe $R$ (Ricorsivamente enumerabile o Decidibile).

## 4.2 Problemi Completi nella Gerarchia Polinomiale

Le formule booleane quantificate (QBF) sono problemi naturali che si collocano ai vari livelli della gerarchia.

**Definizione 10** (SAT Quantificato)**.**

- ***Existential SAT ($SAT_\exists$):*** $\{\langle\Phi\rangle \mid \exists X_1, \ldots, X_n$ *tale che* $\Phi(X_1, \ldots, X_n)$ *è vera*$\}$ *Questo è il problema $SAT$ standard ed è NP-completo ($\Sigma_1^P$-completo).*
- ***Alternating SAT ($\Sigma_2^P$-complete):*** $\{\langle\Phi\rangle \mid \exists X_1, \ldots, X_n \forall Y_1, \ldots, Y_m$ *tale che* $\Phi(X, Y)$ *è vera*$\}$ *Questo problema è $\Sigma_2^P$-completo. Significa: "Esiste un'assegnazione per $X$ tale che per ogni assegnazione di $Y$, la formula $\Phi$ è vera".*
- ***Alternating SAT ($\Sigma_3^P$-complete):*** $\{\langle\Phi\rangle \mid \exists X \forall Y \exists Z$ *tale che* $\Phi(X, Y, Z)$ *è vera*$\}$ *Questo problema è $\Sigma_3^P$-completo. Significa: "Esiste un $X$ tale che per ogni $Y$, esiste un $Z$ tale che $\Phi$ è vera".*

L'alternanza dei quantificatori ("esiste", "per ogni", "esiste", ...) cattura la complessità dei giochi strategici (come gli scacchi): "Esiste una mossa che posso fare, tale che, qualunque mossa faccia il mio avversario, esiste una mossa che posso fare io, tale che... e alla fine vinco."

# 5 Problemi Funzionali e Classi Funzionali

Finora abbiamo parlato di problemi decisionali (risposta sì/no). Ora introduciamo i problemi funzionali, che richiedono il calcolo di un valore.

**Definizione 11** (Functional Min-Cover (FMin-Cover))**.** $FMin-Cover(G) = \min\{|V'| \mid V' \subseteq V \text{ è un Vertex Cover di } G\}$.

Questo problema richiede di calcolare la taglia del Vertex Cover minimo, non solo di decidere se esiste un VC di una certa taglia.

## 5.1 La Classe FP

**Definizione 12** (FP (Functional Polynomial Time))**.** *FP è la classe delle funzioni che possono essere calcolate da trasduttori deterministici in tempo polinomiale. Un trasduttore è una Macchina di Turing con un nastro di output aggiuntivo.*

Le riduzioni polinomiali che usiamo spesso sono funzioni che appartengono a FP.

## 5.2 FMin-Cover e le Classi con Oracolo

Consideriamo nuovamente il problema $FMin-Cover$. Vogliamo calcolare la taglia del VC minimo. Abbiamo a disposizione un oracolo per $VC$ (il problema decisionale, che è NP-completo).

### 5.2.1 Algoritmo per FMin-Cover usando un Oracolo VC:

**1. Ricerca Lineare:**

1. Si inizia con $k = 1$.
2. Si chiede all'oracolo $VC$: "Il grafo $G$ ha un Vertex Cover di taglia $\leq k$?" (query $\langle G, k \rangle$).
3. Se l'oracolo risponde "no", si incrementa $k$ e si ripete.
4. Se l'oracolo risponde "sì", allora $k$ è la taglia minima. Si restituisce $k$.

Il valore massimo di $k$ da testare è $|V|$ (poiché l'insieme di tutti i vertici $V$ è sempre un Vertex Cover). Quindi, si effettuano al più $|V|$ query all'oracolo. Poiché $|V|$ è polinomiale nella dimensione dell'input, questa procedura è in tempo polinomiale. Quindi, $FMin-Cover \in FP^{NP}$.

**2. Ricerca Binaria (più efficiente):**

1. Si stabilisce un intervallo di ricerca per la taglia minima, ad esempio $[0, |V|]$.
2. Si seleziona il punto medio $m$ dell'intervallo.
3. Si chiede all'oracolo $VC$: "Il grafo $G$ ha un Vertex Cover di taglia $\leq m$?" (query $\langle G, m \rangle$).

4. Se l'oracolo risponde "sì", significa che la taglia minima è $\leq m$. Si restringe l'intervallo a $[0, m]$.
5. Se l'oracolo risponde "no", significa che la taglia minima è $> m$. Si restringe l'intervallo a $[m+1, |V|]$.
6. Si ripete fino a quando l'intervallo si riduce a un singolo valore.

Con la ricerca binaria, il numero di query all'oracolo è $O(\log |V|)$. Questa osservazione porta a una notazione più specifica per la classe di complessità: $FMin - Cover \in FP^{NP[O(\log n)]}$. Questa notazione indica che la macchina deterministica in tempo polinomiale effettua solo un numero logaritmico di query all'oracolo in NP.

## 5.3 Perché FMin-Cover non è in FP (se P $\neq$ NP)?

La domanda finale è: $FMin - Cover \in FP$? Se $FMin - Cover$ fosse in $FP$, significherebbe che esisterebbe un algoritmo deterministico in tempo polinomiale capace di **calcolare** la taglia del Vertex Cover minimo. Se avessimo tale algoritmo, potremmo risolvere il problema decisionale $VC$ (che è NP-completo) in tempo polinomiale:

- Input: $\langle G, k \rangle$.
- Calcola $size = FMin - Cover(G)$ usando l'algoritmo FP ipotizzato.
- Se $size \leq k$, rispondi "sì"; altrimenti, rispondi "no".
- Questo significherebbe che $VC \in P$. Ma poiché $VC$ è NP-completo, questo implicherebbe $P = NP$.

Poiché la maggior parte dei ricercatori ritiene che $P \neq NP$, si conclude che $FMin - Cover$ (e la maggior parte dei problemi di ottimizzazione NP-difficili) non è in FP.

Questo mostra la relazione fondamentale tra i problemi decisionali e i loro equivalenti funzionali/di ottimizzazione: la difficoltà di calcolare la soluzione ottimale è direttamente legata alla difficoltà di decidere una proprietà della soluzione.