Logica e Modelli Computazionali

Problemi Difficili e Completi

Marco Console

Ingegneria Informatica e Automatica (Sapienza, Università di Roma)

La Quintessenza di una Classe di Complessità

- Una classe di complessità contiene solitamente un numero infinito di linguaggi e potrebbe contenere una o più sottoclassi di problemi interessanti
  - $\mathbf{NP}$ contiene $\mathbf{P}$ e $\mathbf{EXPTIME}$ contiene $\mathbf{NP}$ e $\mathbf{P}$

- Alcuni linguaggi di una classe potrebbero non presentare tutte le caratteristiche della classe
  - I linguaggi in $\mathbf{P}$ non sono "caratteristici" della classe $\mathbf{EXPTIME}$
  - Il linguaggio $PAL = \{s \in \Sigma^* \mid s \text{ è } palindroma\}$ è in $\mathbf{EXPTME}$ ma non presenta le stesse difficoltà di altri problemi all'interno della classe

- In generale, potrebbe non essere chiaro quali problemi sono davvero rappresentativi di una classe

- Le classi che abbiamo introdotto contengono **tutte** dei problemi speciali molto rappresentativi
  - Questi problemi presentano tutti i requisiti computazionali fondamentali di tale classe
  - È un sintomo della robustezza delle definizioni

Problemi Difficili e Completi – Intuizione

- Assumiamo una classe di linguaggi $\mathbb{C}$ qualunque

- **Intuizione 1**. Un problema $\mathcal{P}$ è **$\mathbb{C}$-hard** se una MT che risolve $\mathcal{P}$ può risolvere ogni problema nella classe $\mathbb{C}$ se utilizzata come subroutine da altre macchine
  - $\mathcal{P}$ è almeno tanto difficile quanto ogni altro problema in $\mathbb{C}$ (se non di più)

- **Intuizione 2**. Un problema $\mathcal{P}$ è **$\mathbb{C}$-complete** se è **$\mathbb{C}$-hard** e appartiene a $\mathbb{C}$
  - È tanto difficile quanto ogni altro problema in $\mathbb{C}$ ma è anche in $\mathbb{C}$
  - **Nota**. Non "esattamente difficile quanto" ogni altro problema in $\mathbb{C}$ perché la classe può contenere un numero (infinito) di problemi facili (non **$\mathbb{C}$-hard**)

- **Intuizione 3**. Una macchina per $\mathcal{P}$ risolve ogni problema in $\mathbb{C}$ se, per ogni problema $\mathcal{P}'$ di $\mathbb{C}$ **esiste una riduzione da $\mathcal{P}'$ a $\mathcal{P}$**
  - Ma quale tipo di riduzione?

Problemi Difficili e Riduzioni – Intuizione

- **Definizione**. Siano A e B due linguaggi sugli alfabete Σ_A e Σ_B. A è ***riducibile*** a B ($\boldsymbol{A \leq_m B}$), se esiste una **funzione computabile** $f: \Sigma^* \to \Sigma^*$ tale che, per ogni $x \in \Sigma^*$

$$x \in A \text{ se e solo se } f(x) \in B$$

- Potremmo essere tentati di utilizzare la nozione di riduzione di Karp per quella di $\mathbb{C}$-hardness

- **Definizione [Prova]**.Un linguaggio $\mathcal{L}$ è $\mathbb{C}$-hard se, per ogni $\mathcal{L}' \in \mathbb{C}$ abbiamo $\mathcal{L}' \leq_M \mathcal{L}$
  - Esiste una qualunque riduzione (funzione computable) da $\mathcal{L}'$ ad $\mathcal{L}$

- Questa definizione non cattura però la nostra intuizione e da vita a comportamenti indesiderati
  - Perchè?

Problemi Difficili e Riduzioni – Intuizione – Esempio

- Prendiamo di nuovo l'esempio di PAL e la classe $\mathbf{EXPTIME}$.
  - Intuitivamente, vorremmo affermare che PAL non è $\mathbf{EXPTIME}$-hard visto che $PAL \in \mathbf{P}$
  - Non è almeno tanto difficile quanto simulare una Macchina di Turing per k passi …

- **Proposizione.** PAL è $\mathbf{EXPTIME}$-complete **secondo la definizione di prova**
- **Dimostrazione**. Chiaramente $PAL \in \mathbf{EXPTIME}$ perché $PAL \in \mathbf{P} \subseteq \mathbf{EXPTIME}$. Definiamo ora una riduzione da $\mathcal{L}$ a PAL, per ogni problema $\mathcal{L} \in \mathbf{EXPTIME}$.
  - Sia $f: \Sigma_{\mathcal{L}} \to \Sigma_{PAL}$ tale che $\boldsymbol{f(x) = aa}$**, se** $\boldsymbol{x \in \mathcal{L}}$**,** $\boldsymbol{f(x) = ab}$ **altrimenti**.
  - Chiaramente $f(x) \in PAL$ se e solo se $x \in \mathcal{L}$. Procediamo a dimostrare che f è computabile
  - $\mathcal{L} \in \mathbf{EXPTIME}$ implica l'esistenza di una macchina M che decide $\mathcal{L}$
  - Per calcolare f, una macchina M_f con input x simula M su x
    - Se M accetta x allora $M_f(x) = aa$
    - Altrimenti $M_f(x) = ab$

Riduzioni Efficienti – Intuizione

- Per evitare i problemi evidenziati nel lucido precedente, dobbiamo limitare il potere delle riduzioni
  - In particolare, la complessità computazionale delle macchine necessarie per calcolarle!

- **Intuizione**. Un linguaggio $\mathcal{L}$ è **$\mathbb{C}$-hard** se, per ogni $\mathcal{L}' \in \mathbb{C}$ esiste una MT M tale che
  - M definisce una **riduzione** da $\mathcal{L}'$ a $\mathcal{L}$ ($M(x) \in \mathcal{L} \leftrightarrow x \in \mathcal{L}'$)
  - M ha **complessità strettamente inferiore** a quella necessaria per riconoscere i linguaggi di $\mathbb{C}$

- Utilizzando la nuova intuizione non possiamo più nascondere la complessità di $\mathbb{C}$ nella riduzione!
  - Perché non possiamo più risolvere il problema di $\mathbb{C}$ utilizzando solo la riduzione
  - Come accadeva nell'esempio precedente

Riduzioni Efficienti – Intuizione

- **Definizione 1**. Siano A e B due linguaggi sugli alfabeti Σ_A e Σ_B. A è ***riducibile*** a B ($\boldsymbol{A \leq_m B}$), se esiste una **funzione computabile** $f: \Sigma^* \to \Sigma^*$ tale che, per ogni $x \in \Sigma^*$ $x \in A$ se e solo se $f(x) \in B$

- **Intuizione**. Per definire problemi $\mathbf{EXPTIME}$-hard imponiamo riduzioni calcolabili in tempo polinomiale

- **Definizione 2**. Un linguaggio $\mathcal{L}$ è $\mathbf{EXPTIME}$-hard se, per ogni $\mathcal{L}' \in \mathbb{C}$ esiste una Macchina di Turing M tale che $M(x) \in \mathcal{L} \leftrightarrow x \in \mathcal{L}'$, per ogni $x \in \mathcal{L}$, e $\boldsymbol{M}$ **ha complessità temporale** $\boldsymbol{f = O(n^k)}$ **con** $\boldsymbol{k} \in \mathbb{N}$

- Utilizzando Definizione 2 la prova precedente del fatto che PAL è $\mathbf{EXPTIME}$-hard non è più valida
  - Non abbiamo più utilizzare la riduzione per risolvere i problemi in $\mathbf{EXPTIME}$
  - In questo modo, un problema $\mathbf{EXPTIME}$-hard cattura effettivamente l'intuizione desiderata

- **Teorema**. $H_b = \{ (e(M), e(\sigma), e(n)) \mid M \text{ termina dopo } n \text{ passi su input } \sigma \}$ è $\mathbf{EXPTIME}$-complete

NP-Completezza

Riduzioni per la Classe NP

- Come abbiamo discusso fino ad ora, prima di parlare di problemi difficili e completi per una classe, dobbiamo stabilire quali riduzioni ammettiamo nelle nostre dimostrazioni
  - Classi diverse potrebbero aver bisogno di riduzioni diverse, come discusso

- Per la classe **NP** convenzionalmente utilizziamo **riduzioni polinomiali**
  - Riduzioni calcolabili con Macchine di Turing la cui complessità temporale è $O(n^k)$ per $k \in \mathbb{N}$

- Il motivo per questa scelta è duplice
  - Congetturiamo che i problemi in **NP** siano strettamente più difficili di quelli in **P** **(ma non sappiamo dimostrarlo)**
  - Una macchina deterministica con complessità temporale $f = O(n^k)$ può essere simulata da una macchina non deterministica con la stessa complessità.

La Nozione di Riduzione di Karp – Definizione

- **Definizione.** Una funzione $f: \Sigma^* \to \Sigma^*$ (con Σ un alfabeto di simboli) è ***computabile*** se esiste una **MdT** M_f tale che, per ogni $x \in \Sigma^*$, esiste una sequenza di configurazioni $C_1, \dots, C_n$ di M_f con $C_1 = (\epsilon, q_0, x)$, $C_n = \left(\epsilon, q_{yes}, f(x)\right)$ e $C_i \Rightarrow_M C_{i+1}, i = 1, \dots, n-1$

- **Definizione.** Diciamo che la macchina M_f **calcola** f
  - **Intuizione**. Per ogni $x \in \Sigma^*$, M_f termina in una configurazione in cui il suo nastro contiene solo $f(x)$, effettivamente calcolando $f: \Sigma^* \to \Sigma^*$

- **Definizione**. Siano A e B due linguaggi sugli alfabeti Σ_A e Σ_B (non necessariamente gli distinti). A è ***Karp-riducibile*** a B ($A \leq_m B$), se esiste una **funzione computabile** $f: \Sigma_A^* \to \Sigma_B^*$ tale che, per ogni $x \in \Sigma_A^*$, $x \in A$ se e solo se $f(x) \in B$
  - *Anche detto riducibile molti-a-uno*, oppure semplicemente ***riducibile***

- **Definizione**. La funzione f viene chiamata **Karp-Riduzione da A in B**
  - Anche riduzione molti a uno oppure semplicemente ***riduzione***

Riduzioni efficienti

- **Definizione**. Una funzione $f: \Sigma^* \to \Gamma^*$ è **computabile in tempo polinomiale** se esiste una macchina di Turing M tale che:
  - M calcola f
  - M ha complessità temporale $O(n^k)$ per un qualche $k \in N$

- **Definizione.** Siano A e B due linguaggi. Diremo che ***A* è *Karp-riducibile* in *tempo polinomiale* a *B*** (denotato con $\boldsymbol{A \leq_m^p B}$) se esiste una **funzione computabile in tempo polinomiale** $f: \Sigma^* \to \Sigma^*$ tale che, per ogni $x \in \Sigma^*$, $x \in A$ se e solo se $f(x) \in B$
  - Riducibile molti-a-uno in tempo polinomiale, oppure semplicemente riducibile in tempo polinomiale

- **Definizione.** La funzione f viene chiamata **Karp-riduzione in tempo polinomiale da A in B**
  - Riduzione molti-a-uno polinomiale, oppure semplicemente riduzione polinomiale

Linguaggi Difficili e Completi Per NP

- Utilizzando le riduzioni polinomiali siamo pronti a definire i problemi **NP**-hard

- **Definizione**: Un linguaggio L si dice **NP-hard sotto le riduzioni polinomiali** se *per ogni* $L' \in \mathbf{NP}$ è tale che $\boldsymbol{L' \leq_m^p L}$
  - Semplicemente **NP-hard** se questo non crea confusione

- **Definizione** : Un linguaggio L si dice **NP-complete sotto le riduzioni in tempo polinomiale** se: $L \in \mathbf{NP}$ **e** L è **NP-hard** sotto le riduzioni in tempo polinomiale

- **Domanda.** Esistono problemi **NP-completi**?

Linguaggi Difficili e Completi Per NP

- Fissiamo un alfabeto di variabili proposizionali $\mathbf{V}$ e un Encoding ragionevole $\boldsymbol{e}$ per le formule proposizionali su $\boldsymbol{V}$
  - Possiamo utilizzare i simboli stessi delle formule nella macchina

- **Definizione.** L_{SAT} è il linguaggio di stringhe definito come segue

$$\{e(\varphi) \mid \varphi \text{ è una formula proposizionale } \textbf{soddisfacibile} \text{ su } \boldsymbol{V}\}$$

- **Teorema [Cook-Levin]:** L_{SAT} è NP-hard (e quindi **NP-completo**).
- **Dimostrazione**. La dimostrazione (non banale) definisce una riduzione per ogni $L \in \mathbf{NP}$
- Se $L \in \mathbf{NP}$ allora esiste una MdT non-deterministica M_L che decide L in tempo polinomiale
- Definiamo una funzione e_M tale che, per ogni input x per M, $e_M(x)$ è una formula proposizionale e $e_M(x)$ è soddisfacibile se e solo se M accetta x
- Dimostriamo che e_M è una funzione calcolabile in tempo polinomiale (utilizzando M)
- Concludiamo che e_M è una riduzione polinomiale da L a L_{SAT}

Forma Normale Congiuntiva

- **Definizione.** Una formula della logica proposizionale φ è in Forma Normale Congiuntiva (CNF) se è nella forma $c_1 \wedge c_2 \wedge \cdots \wedge c_n$ dove ogni c_i è una formula della forma $(l_1 \vee l_2 \vee \cdots \vee l_k)$ e l_j è una variabile o una variabile negata.
  - **Ogni c_i è una clausola di φ**

- **Esempi**. Le seguenti formule sono in Forma Normale Congiuntiva
  - $(v_2) \wedge (v_1 \vee v_2) \wedge (v_1 \vee \neg v_3 \vee v_4) \wedge (v_4 \vee \neg v_4 \vee v_5 \vee v_6)$
  - $(v_1) \wedge (v_2) \wedge (\neg v_3)$
  - $(v_1 \vee v_2 \vee v_3) \wedge (v_1 \vee \neg v_3 \vee v_4)$

- **Esempi**. Le seguenti formule **NON** sono in Forma Normale Congiuntiva
  - $(v_2) \wedge \neg(v_1 \vee v_2) \wedge (v_1 \vee \neg v_3 \vee v_4) \wedge (v_4 \vee \neg v_4 \vee v_5 \vee v_6)$
  - $(v_1 \rightarrow v_2)$
  - $(v_1 \wedge v_2) \vee (\neg v_2 \wedge v_3)$

Linguaggi Difficili e Completi Per NP

- Fissiamo un alfabeto di variabili proposizionali $\mathbf{V}$ e un Encoding ragionevole $\boldsymbol{e}$ per le formule proposizionali su $\boldsymbol{V}$
  - Possiamo utilizzare i simboli stessi delle formule nella macchina

- **Definizione.** CNF è il linguaggio di stringhe definito come segue

$\{e(\varphi) \mid \varphi$ è una formula proposizionale in **forma normale congiuntiva soddisfacibile** su $\boldsymbol{V}\}$

- **Teorema:** CNF è NP-hard (e quindi **NP-completo**).
- **Dimostrazione**. Le formule prodotte dalla dimostrazione del teorema di Cook e Levin possono essere trasformate in formule in CNF in tempo polinomiale.
- Tale prova è stata fornita nello stesso articolo scientifico in cui è stata dimostrata l'NP-completezza di L_{SAT}

In **P** o **NP**-completo?

In **P** o **NP**-completo?

- La classe $\mathbf{NP}$ contiene una grande quantità di problemi computazionali naturali e di forte interesso pratico (molti dei problemi affrontati quotidianamente sono in $\mathbf{NP}$)

- Quando ci troviamo di fronte ad uno di questi problemi, la cosa più naturale da fare è capire se il problema sia risolvibile in tempo polinomiale (e quindi il problema è in **P**) oppure dimostrare che il problema è $\mathbf{NP}$-hard (e quindi **NP-completo**)
  - Tramite una riduzione da un altro problema che è già noto essere **NP-difficile**.

- Nel secondo caso, abbiamo appena dimostrato che **molto probabilmente** non esiste un algoritmo che risolva il problema in **tempo polinomiale**
  - Tale algoritmo esisterebbe solo se $\mathbf{P} = \mathbf{NP}$

3CNF

- **Definizione.** Una formula della logica proposizionale è kCNF se è nella forma $c_1 \wedge c_2 \wedge \cdots \wedge c_n$ dove ogni c_i è una formula della forma $(l_1 \vee l_2 \vee l_3)$ e l_j è una variabile o una variabile negata.

- **Esempi**. Le seguenti formule sono in Forma Normale Congiuntiva
  - $(v_1 \vee v_2 \vee v_3) \wedge (v_1 \vee \neg v_3 \vee v_4)$

- **Esempi**. Le seguenti formule **NON** sono in Forma Normale Congiuntiva
  - $(v_2) \wedge (v_1 \vee v_2) \wedge (v_1 \vee \neg v_3 \vee v_4) \wedge (v_4 \vee \neg v_4 \vee v_5 \vee v_6)$
  - $(v_1) \wedge (v_2) \wedge (\neg v_3)$
  - $(v_2) \wedge \neg(v_1 \vee v_2) \wedge (v_1 \vee \neg v_3 \vee v_4) \wedge (v_4 \vee \neg v_4 \vee v_5 \vee v_6)$
  - $(v_1 \rightarrow v_2)$
  - $(v_1 \wedge v_2) \vee (\neg v_2 \wedge v_3)$

3CNF rimane NP-completo - I

- **Definizione.** $\mathbf{3}CNF$ è il linguaggio di stringhe definito come segue

  $\{e(\varphi) \mid \varphi$ è una formula proposizionale in 3CNF **soddisfacibile** su $\boldsymbol{V}\}$

- **Teorema**. **CNF $\leq_m^p$ 3CNF**
- **Dimostrazione**. Dimostriamo l'esistenza di una funzione computabile in tempo polinomiale f che, data $\phi \in CNF$, restituisce una **$f(\phi)$ in $3CNF$ tale che ϕ è soddisfacibile se e solo se lo è $f(\phi)$**

- Considera ogni clausola $c_i = l_{i,1} \vee \cdots \vee l_{i,p_i}$ di ϕ. Abbiamo 4 possibili cai
  - 1) $p_i = 1$, 2) $p_i = 2$, 3) $p_i = 3$, 4) $p_i > 3$

- Nel caso 1), introduciamo due nuove variabili $x'_{i,1}$ e $x'_{i,2}$ e $f(\phi)$ produce le clausole:

  $$(l_{i,1} \vee x'_{i,1} \vee x'_{i,2}) \wedge (l_{i,1} \vee \neg x'_{i,1} \vee x'_{i,2}) \wedge (l_{i,1} \vee x'_{i,1} \vee \neg x'_{i,2}) \wedge (l_{i,1} \vee \neg x'_{i,1} \vee \neg x'_{i,2})$$

- Per costruzione, se un'assegnazione T rende vera $f(\phi)$, allora il letterale $l_{i,1}$ deve necessariamente essere vero (ovvero, se $l_{i,1} = x$, allora $T(x) = 1$; altrimenti, $T(x) = 0$)

3CNF rimane NP-completo - II

- Nel caso 2), si introduce una nuova variabile $x'_{i,1}$ e $f(\phi)$ produce le seguenti clausole:
$$\boldsymbol{(l_{i,1} \vee l_{i,2} \vee x'_{i,1}) \wedge (l_{i,1} \vee l_{i,2} \vee \neg x'_{i,1})}$$
- Per costruzione, se un'assegnazione T rende vera $f(\phi)$, allora uno dei due letterali tra $l_{i,1}$ e $l_{i,2}$ deve necessariamente essere vero
- Nel caso 3), $f(\phi)$ copia semplicemente la clausola c_i così com'è
- Nel caso 4), si riduce la dimensione della clausola $c_i = (l_{i,1} \vee \cdots \vee l_{i,p_i})$ come segue:
$\boldsymbol{(l_{i,1} \vee l_{i,2} \vee x'_i) \wedge (\neg x'_i \vee l_{i,3} \vee \cdots \vee l_{i,p_i})}$, dove x'_i è una nuova variabile
- Quindi $f(\phi)$ aggiunge la clausola $\boldsymbol{(l_{i,1} \vee l_{i,2} \vee x'_i)}$ di tre letterali e passa poi a ridurre (se necessario) la rimanente clausula $\boldsymbol{(\neg x'_i \vee l_{i,3} \vee \cdots \vee l_{i,p_i})}$

3CNF rimane NP-completo - III

Esempio: Se $c_4 = (x_1 \vee \neg x_3 \vee x_7 \vee \neg x_8 \vee \neg x_6)$, allora $f(\phi)$ prima sostituisce c_4 con:

$$(\boldsymbol{x_1 \vee \neg x_3 \vee x'_{4,1}}) \wedge (\neg x'_{4,1} \vee x_7 \vee \neg x_8 \vee \neg x_6)$$

Poi sostituisce $(\neg x'_{4,1} \vee x_7 \vee \neg x_8 \vee \neg x_6)$ con: $(\boldsymbol{\neg x'_{4,1} \vee x_7 \vee x'_{4,2}}) \wedge (\boldsymbol{\neg x'_{4,2} \vee \neg x_8 \vee \neg x_6})$

- Chiaramente, f costruisce una $3CNF$ in tempo polinomiale
- Dobbiamo dimostrare che $f(\varphi) \in 3CNF$ se e solo se $\varphi \in CNF$.
  - Per farlo possiamo dimostrare il seguente risultato

- **Lemma.** Sia c una clasola di $\varphi \in CNF$ e sia I una interpretazione per le variabili in φ. Allora $I \vDash c$ se e solo se $I' \vDash f(c)$, dove I' è l'interpretazione per le variabili di $f(\varphi)$ definita come segue
  - $I'(v) = I(v)$ per ogni variabile v in φ
  - $I'(v) = 0$ per ogni variabile v di $f(\varphi)$ che non compare in φ

3CNF rimane NP-completo – Esempio Riduzione

- Consideriamo $\varphi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_1) \wedge (x_2 \vee x_3)$

- La formula $f(\varphi)$ è definita come segue
  - $(x_1 \vee \neg x_2 \vee \neg x_3)$ rimane invariato
    - $\boldsymbol{(x_1 \vee \neg x_2 \vee \neg x_3)}$
  - $(x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4)$ viene spezzato in due clausole
    - $\boldsymbol{(x_1 \vee \neg x_2 \vee V_1) \wedge (\neg V_1 \vee \neg x_3 \vee x_4)}$
  - (x_1) subisce un padding con 2 altre variabili
    - $\boldsymbol{(x_1 \vee V_2 \vee V_3) \wedge (x_1 \vee \neg V_2 \vee V_3) \wedge (x_1 \vee V_2 \vee \neg V_3) \wedge (x_1 \vee \neg V_2 \vee \neg V_3)}$
  - $(x_2 \vee x_3)$ subisce un padding con 1 altra variabile
    - $\boldsymbol{(x_2 \vee x_3 \vee V_4) \wedge (x_2 \vee x_3 \vee \neg V_4)}$

- La formula $f(\varphi)$ è la congiunzione delle **clausole in verde**

IndSet è NP-completo - I

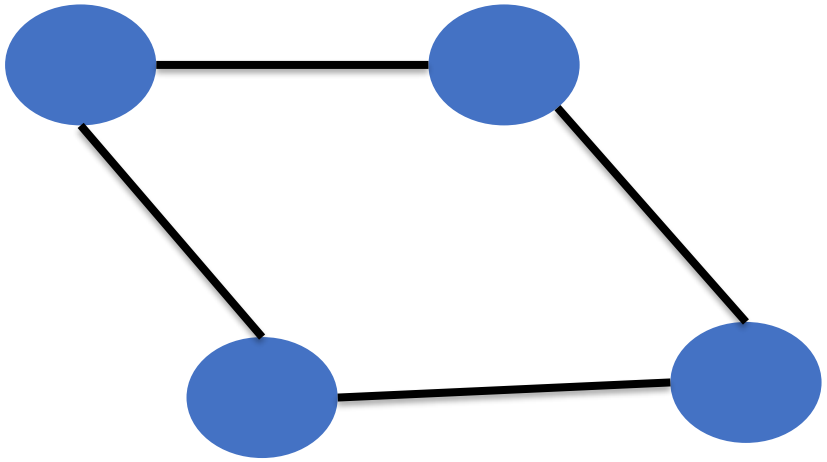
- **Definizione** Un **independent set** I di un grafo $G = (V, E)$ è un insieme $\mathcal{I}$ di nodi tale che
  - $\mathcal{I} \subseteq V$;
  - Per ogni coppia di nodi $(i, j) \in V^2$, abbiamo che $(i, j) \notin E$

- **Definizione.** $IndSet$ è il linguaggio di stringhe definito come segue

  $\{e(G, k)$ | Esiste un independent set I di G con $|I| = k\}$

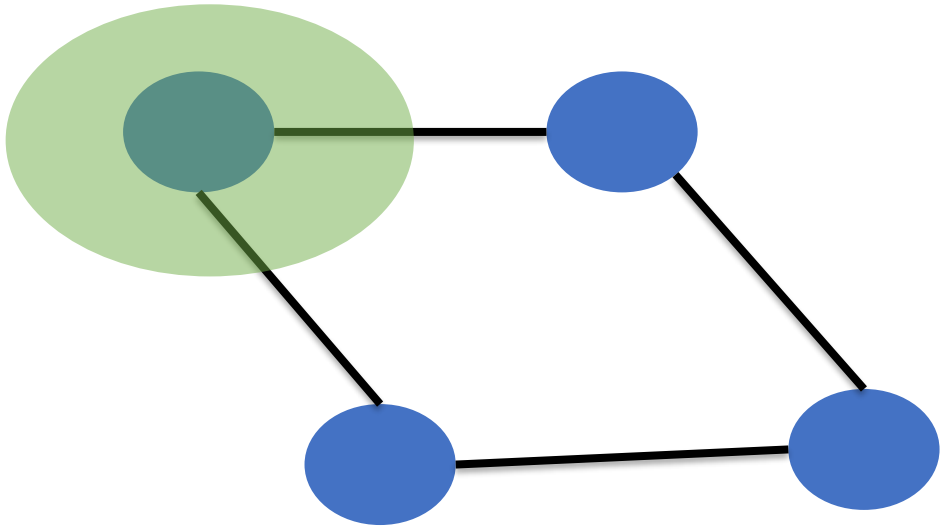
- **Proposizione.** IndSet è **NP**-completo
- **Dimostrazione**. IndSet $\in$ **NP** è ovvio. Dimostriamo che $\mathbf{3CNF} \leq_m^p$ **IndSet**, e che quindi IndSet è anche **NP**-hard

IndSet è NP-completo



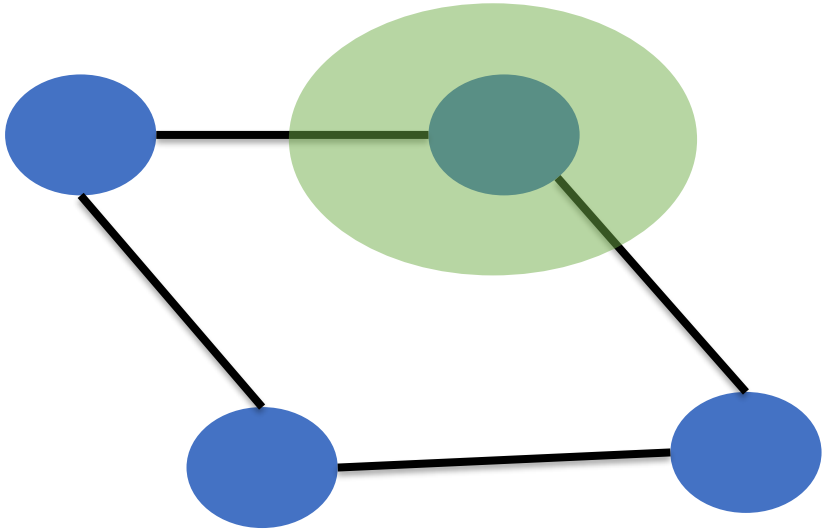
IndSet è NP-completo

È un indipendent set



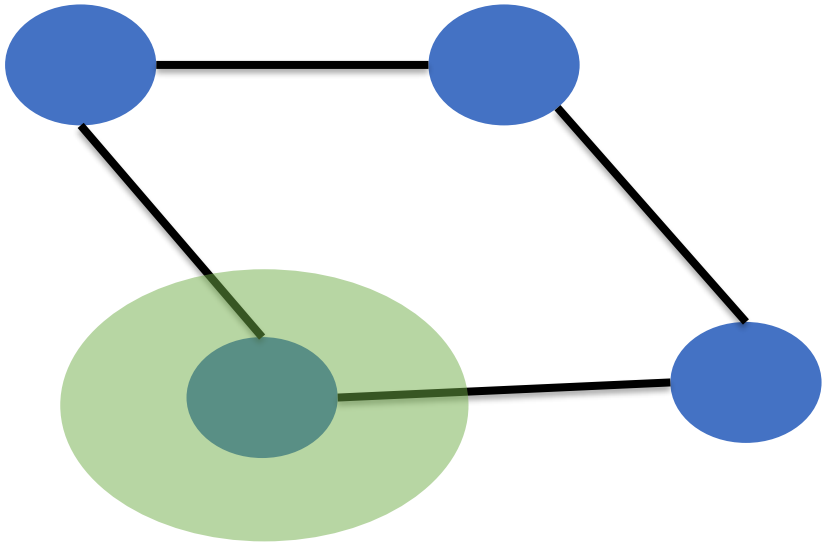
IndSet è NP-completo

È un indipendent set



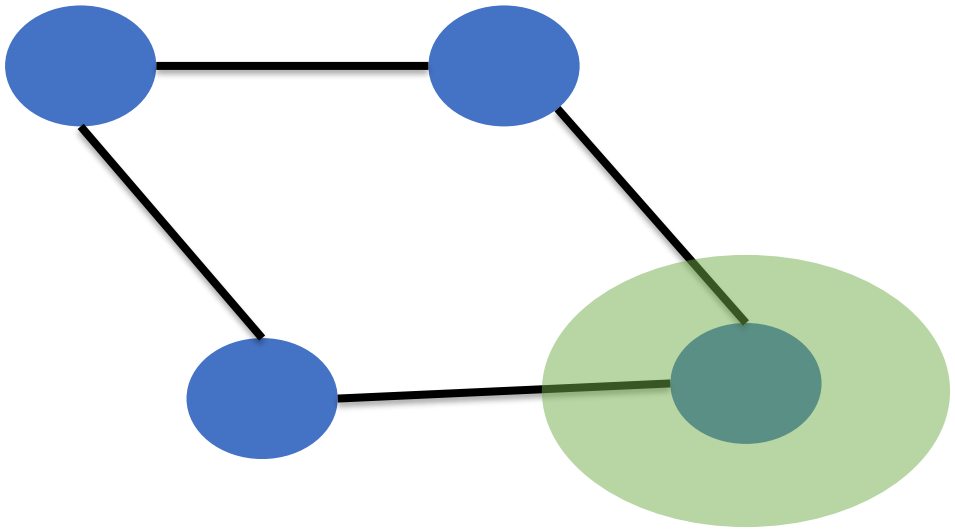
IndSet è NP-completo

È un indipendent set



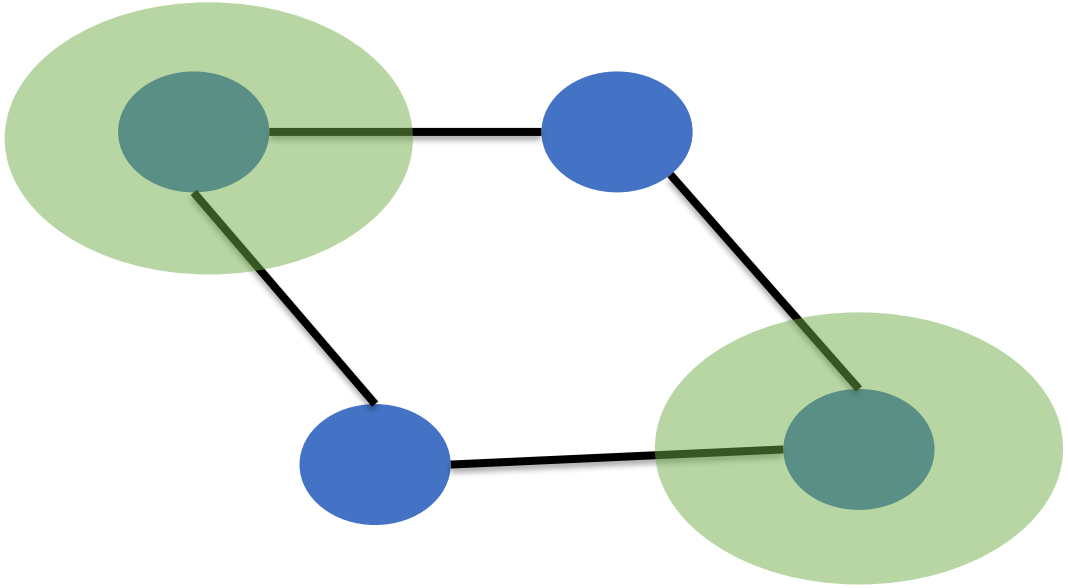
IndSet è NP-completo

È un indipendent set



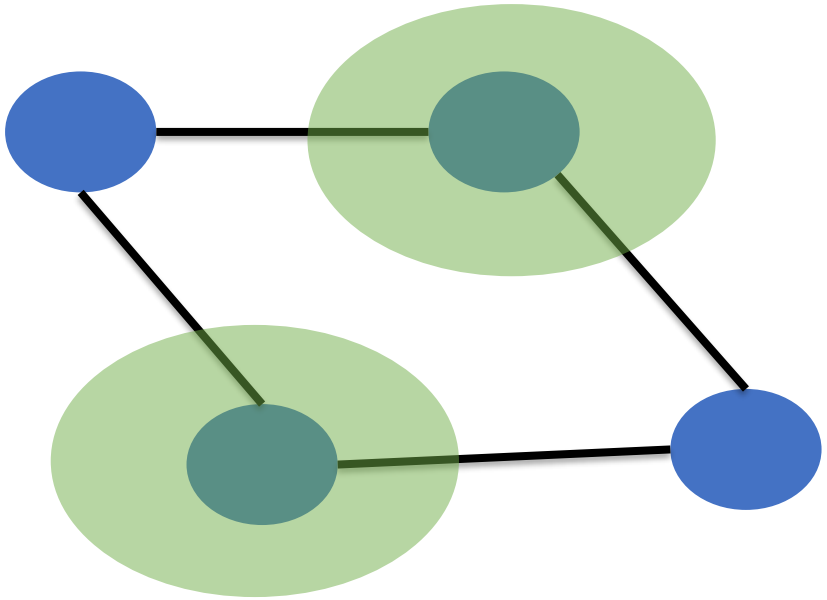
IndSet è NP-completo

È un indipendent set



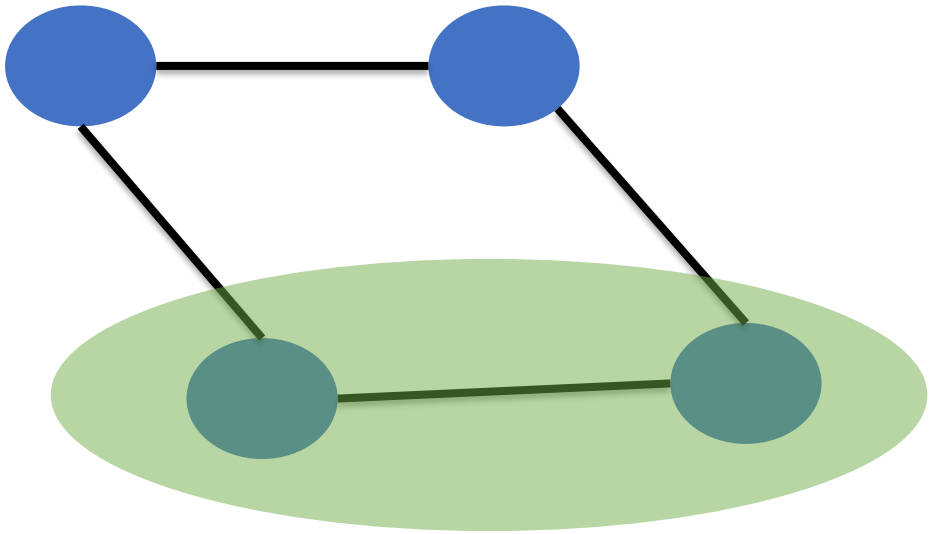
IndSet è NP-completo

È un indipendent set



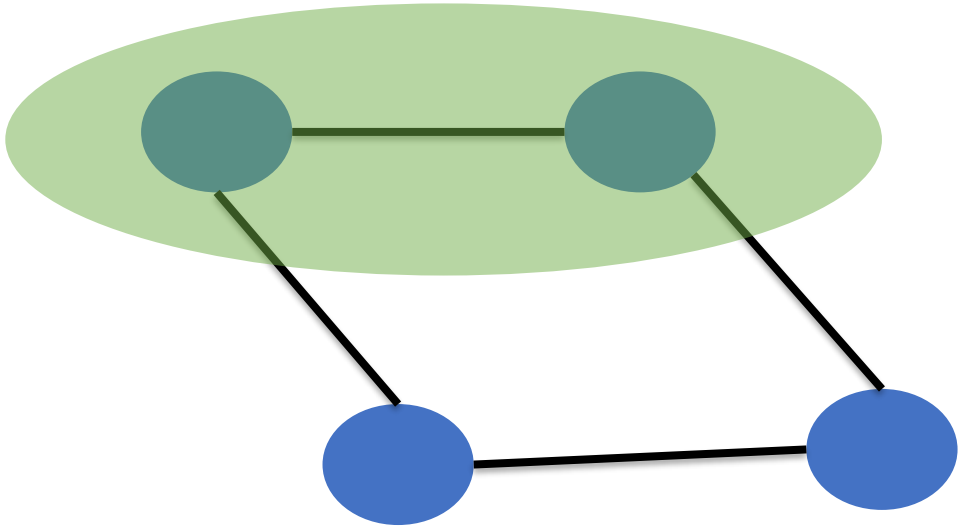
IndSet è NP-completo

È un indipendent set



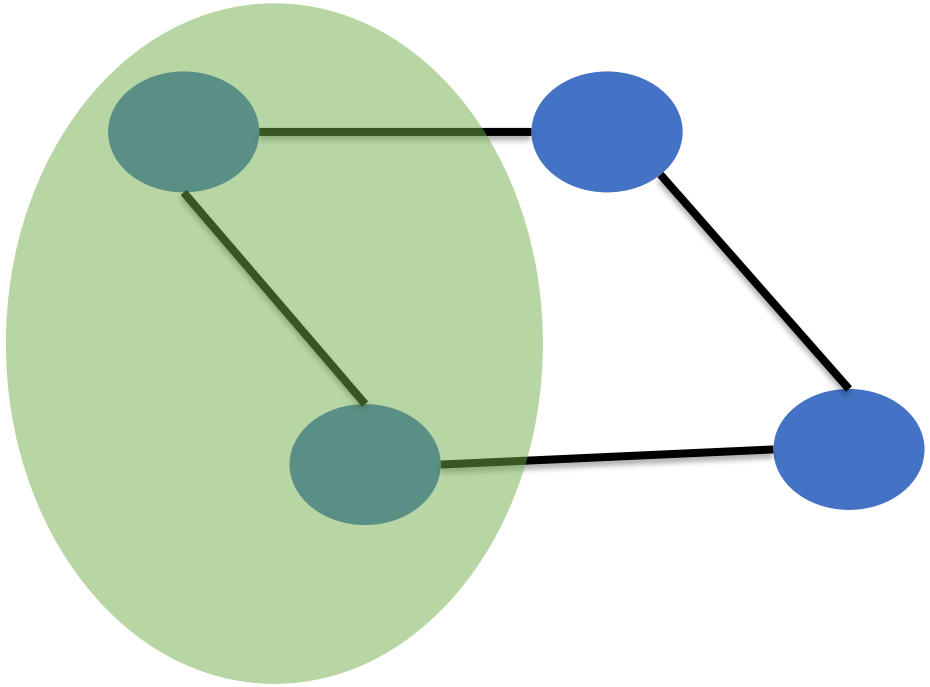
IndSet è NP-completo

È un indipendent set



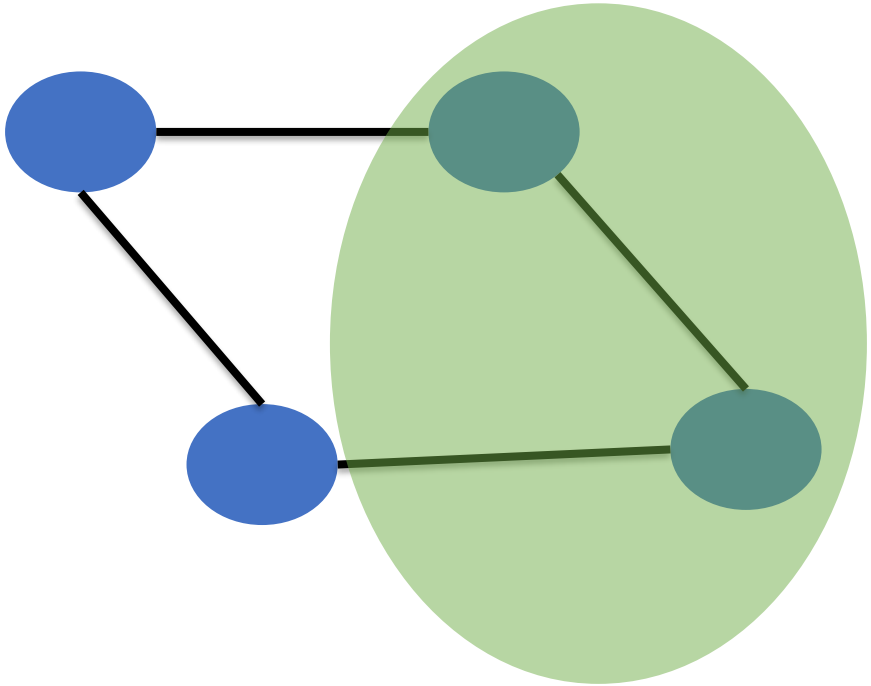
IndSet è NP-completo

È un indipendent set



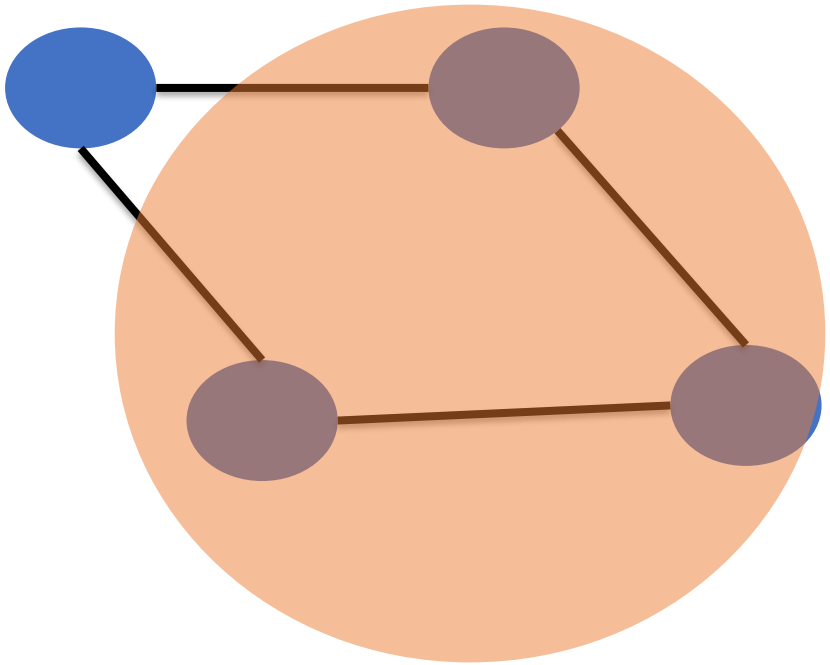
IndSet è NP-completo

È un indipendent set

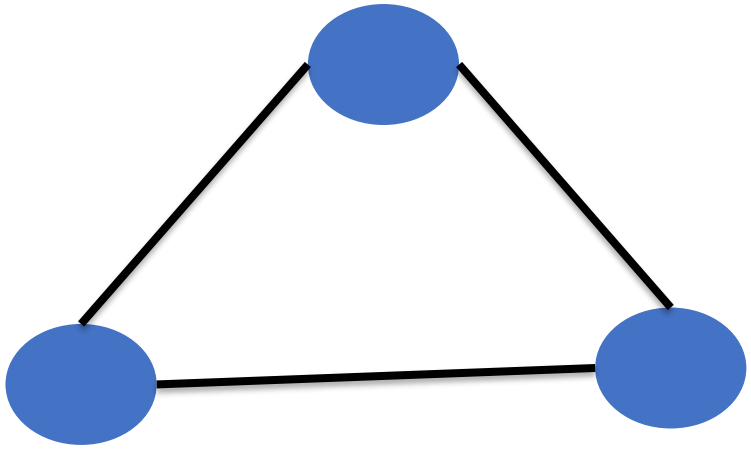


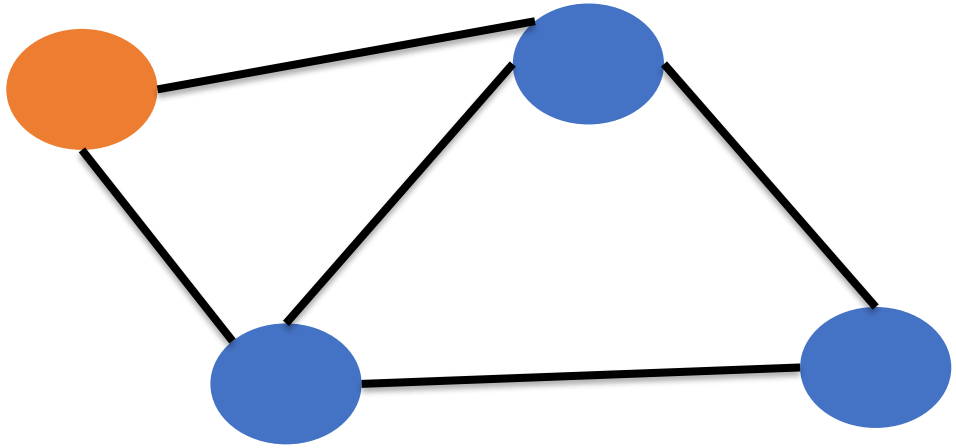
IndSet è NP-completo

Non è un Indiendent set



IndSet è NP-completo





- Un grafo della forma $(a, b), (b, c), (c, d)$ è detto un **triangolo**

- **Osservazione 1**: Dato un triangolo G, ogni Independent Set di G può contenere al massimo un nodo

- **Osservazione 2**: Dato un triangolo G' in cui G è un sottografo, ogni Independent Set di G' può contenere al massimo un nodo da G

IndSet è NP-completo - II

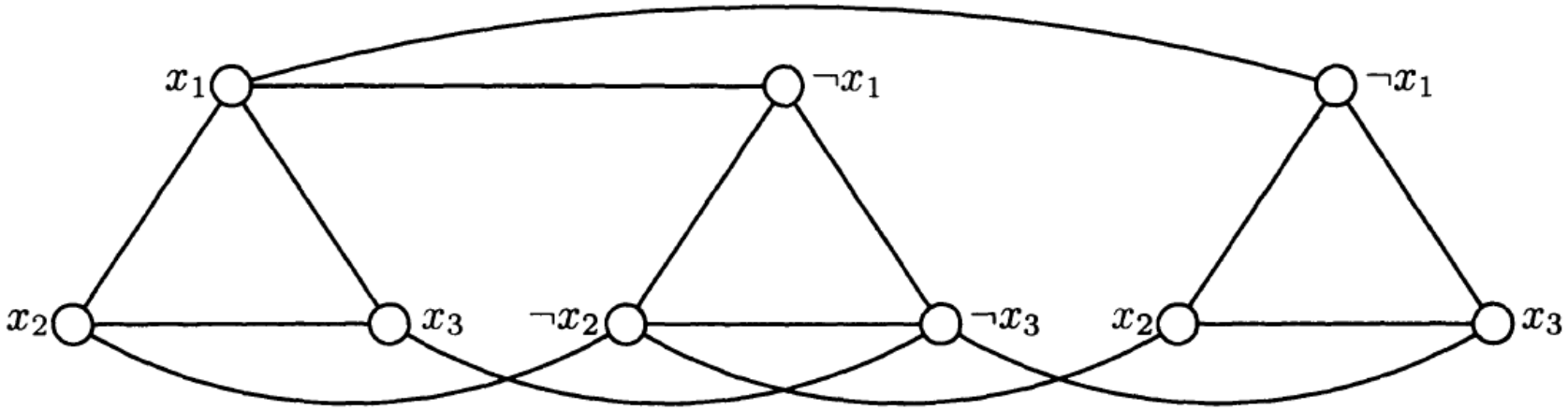
- Data una formula ϕ in 3CNF **contenente m clausole**, f costruisce un grafo $G(\phi)$ sui nodi $V(\phi) = \{v_{i,j} \mid 1 \leq i \leq m,\ 1 \leq j \leq 3\}$ e un valore k_ϕ nel seguente modo:
  - **Ogni clausola di ϕ corrisponde ad un triangolo in $G(\phi)$:** per ogni clausola $c_i = (l_{i,1} \vee l_{i,2} \vee l_{i,3})$ di ϕ, f aggiunge in $G(\phi)$ i seguenti archi: $(v_{i,1}, v_{i,2})$, $(v_{i,1}, v_{i,3})$, $(v_{i,2}, v_{i,1})$, $(v_{i,2}, v_{i,3})$, $(v_{i,3}, v_{i,1})$ e $(v_{i,3}, v_{i,2})$
  - **Due nodi in triangoli diversi sono connessi se e solo se i corrispondenti letterali l, l' sono tali che $l = \neg l'$** (cioè se e solo se corrispondono a letterali opposti): $G(\phi)$ contiene anche i seguenti archi:

$$\{(v_{i,j}, v_{p,k}) \mid i \neq p \wedge l_{i,j} = \neg l_{p,k}\} \cup \{(v_{p,k}, v_{i,j}) \mid i \neq p \wedge l_{i,j} = \neg l_{p,k}\}$$

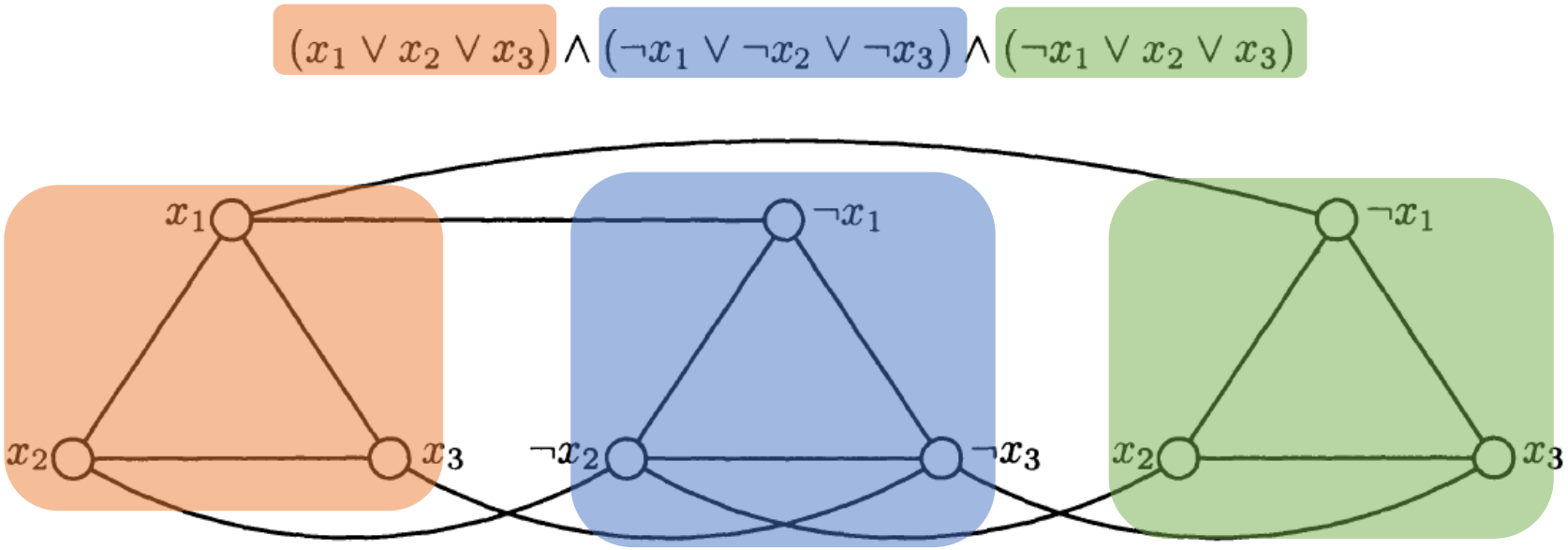
  - Definiamo $k(\phi) = m$ (numero di clausola)

IndSet è NP-completo – Esempio Effetto Riduzione

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$$



IndSet è NP-completo – Esempio Effetto Riduzione



IndSet è NP-completo - III

- La funzione f trasforma d_ϕ in d_G, d_k, è una funzione computabile in tempo polinomiale.
  - Semplicemente, scorre la formula e costruisce le componenti (gadget) della riduzioen
- **Lemma**. $f(x) \in IndSet$ se e solo se $x \in 3CNF$.
- **Assumiamo $f(x) \in IndSet$, dimostriamo $x \in 3CNF$.**
  - Se $f(x) \in IndSet$, allora $G(\phi)$ ha un I.S. $\mathcal{I}$ tale che $|\mathcal{I}| = m$ pari al numero di clausole in x
  - Come abbiamo osservato, all'interno di un "triangolo", un I.S. può contenere un solo nodo
  1. Affinché un $\mathcal{I}$ abbia dimensione m, quindi, $\mathcal{I}$ deve contenere un nodo per ogni triangolo di $G(\phi)$
     - Non ci sono altri nodi disponibili all'interno di G
  2. Inoltre $\mathcal{I}$ non può contenere nodi che rappresentano x e $\neg x$ (per qualunque $x \in Var(\phi)$)
     1. Tali nodi sono collegati in $G(\phi)$ e non sarebbe un Independent Set
  - Consideriamo ora $J: Var(\phi) \rightarrow \{0,1\}$ tale che
    - $J(v) = 1$ se un nodo che rappresenta il letterale v (v positivo) è in $\mathcal{I}$,
    - $J(v) = 0$ altrimenti, $\forall v \in Var(\phi)$
  - J è una interpretazione per ϕ (non assegna la stessa variabile a due valori di verità a causa di 2)
  - Inoltre J soddisfa ϕ (soddisfa un letterale per clausola a causa di 1)

IndSet è NP-completo - III

- La funzione f trasforma d_ϕ in d_G, d_k, è una funzione computabile in tempo polinomiale.
  - Semplicemente, scorre la formula e costruisce le componenti (gadget) della riduzioen
- **Lemma**. $f(x) \in IndSet$ se e solo se $x \in 3CNF$.
- **Assumiamo $x \in 3CNF$ dimostriamo che $f(x) \in IndSet$**
  - Se $x \in 3CNF$ allora esiste un assegnamento J che soddisfa ϕ
  1. Chiaramente, J soddisfa almeno 1 letterale per ogni clausola di ϕ
  2. Sia $\mathcal{I}$ l'insieme dei nodi di $G(\phi)$ tale che, per ogni clausola di ϕ, $\mathcal{I}$ contiene esattamente 1 nodo che rappresenta un letterale soddisfatto da J
  - Procediamo a dimostrare che $\mathcal{I}$ è un Independent Set di dimensione m
  - La dimensione di $\mathcal{I}$ è m per costruzione (ci sono m clausole nella formula)
  - Per ogni triangolo di $G(\phi)$, $\mathcal{I}$ contiene esattamente 1 nodo (il letterale soddisfatto da J **Punto 1**)
  - Inoltre, $\mathcal{I}$ non contiene nodi che rappresentano un variabile e la sua negazione (**Punto 2**)
  - Visto che gli archi di $G(\phi)$ provengono dai triangoli o dai letterali inversi, concludiamo che $\mathcal{I}$ è un IS

VertexCover è NP-completo - I

- **Definizione** Un **Vertex Cover** C di un grafo $G = \langle V, E \rangle$ è un insieme di nodi tale che per ogni arco $(a, b) \in E$, $a \in C$ oppure $b \in C$

- **Definizione.** VC è il linguaggio di stringhe definito come segue
  $\{e(G, k)$ | Esiste un Vertex Cover C di G con $|C| = k\}$

- **Proposizione.** VC è **NP**-completo
- **Dimostrazione**. VC $\in$ **NP** è ovvio. Dimostriamo che $IndSet \leq_m^p$ **VC**, e che quindi VC è anche **NP**-hard. Per dimostrare la proprietà desiderata passiamo per il seguente lemma

- **Lemma.** Sia $G = \langle V, E \rangle$ un grafo. $\mathcal{I}$ è un Independent Set di G se e solo se $V \setminus \mathcal{I}$ è un Vertex Cover di G

VertexCover è NP-completo - I

- **Lemma.** Sia $G = \langle V, E \rangle$ un grafo. $\mathcal{I}$ è un Independent Set di G se e solo se $V \setminus \mathcal{I}$ è un Vertex Cover di G
- **Dimostrazione.** Dividiamo la dimostrazione in due casi.
- **Assumiamo che $\mathcal{I}$ sia un Independent Set di G .** Non esiste nessun arco $\mathrm{e} \in E$ tale che $\mathrm{e} = (a, b)$ e $a, b \in \mathcal{I}$, altrimenti $\mathcal{I}$ non sarebbe un IS a causa di e. Quindi $V \setminus S$ contiene a oppure b per ogni $(a, b) \in E$. Concludiamo che tale insieme è un Vertex Cover.
- **Assumiamo che $\mathrm{V} \setminus \mathcal{I}$ sia un Vertex Cover di G .** Allora, per ogni arco di $\mathrm{e} \in E$ tale che $\mathrm{e} = (a, b)$ abbiamo $a \in \mathrm{V} \setminus \mathcal{I}$ oppure $\mathrm{b} \in \mathrm{V} \setminus \mathcal{I}$. In questo caso, $\mathcal{I}$ non può contenere alcuna coppia $a, b \in V$ tale che $(a, b) \in E$. Concludiamo che tale $\mathcal{I}$ è un Independent Set di G

VertexCover è NP-completo - I

- **Proposizione.** VC è **NP**-completo
- **Dimostrazione**. VC ∈ **NP** è ovvio. Dimostriamo che $IndSet \leq_m^p \mathbf{VC}$, e che quindi VC è anche **NP**-hard. Per dimostrare la proprietà desiderata passiamo per il seguente lemma
- **Lemma 1.** Sia $G = \langle V, E \rangle$ un grafo. $\mathcal{I}$ è un Independent Set di G se e solo se $V \setminus \mathcal{I}$ è un Vertex Cover di G
- Definiamo $f(G, k) = (G, |V| - k)$.
  - La computabilità in tempo polinomiale è ovvia
- Dimostriamo la (semplicissima) correttezza applicando Lemma 1

- Se $(G, |V| - k) \in VC$ allora esiste un Vertex Cover C=$V \setminus \mathcal{I}$ di dimensione $|V| - k$ (Lemma 1)
  - Possiamo concludere che $(G, k) \in IndSet$ essendo $\mathcal{I}$ un Independent Set di dimensione k
- Se $(G, k) \in IndeSet$ allora esiste un Independent set $\mathcal{I}$ di dimensione k.
  - Possiamo concludere che $(G, |V| - k) \in VC$ perché $V \setminus \mathcal{I}$ (la cui dimensione è k) è un vertex cover (Lemma 1)

SetCover è NP-completo - I

- **Definizione** Un **Set Cover** C di una famiglia di insiemi U è una famiglia $C \subseteq U$ tale che la sua unione è uguale all'unione U

- **Esempio**: $U = \{1, 2, 3, 4, 5\}$, $C = \{\{1, 2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}$, $k = 2$. In questo caso, abbiamo $\{1, 2, 3\} \cup \{4, 5\} = U$
- **Esempio**: $U = \{1, 2, 3, 4, 5\}$, $C = \{\{1, 2, 3\}, \{2, 4\}, \{3, 4\}, \{5\}\}$, $k = 2$. In questo caso, non esistono due insiemi in C la cui unione fa U

- **Definizione.** SC è il linguaggio di stringhe definito come segue
  $\{e(U, k)$ | Esiste un Set Cover C di U con $|C| = k\}$

- **Proposizione.** SC è **NP**-completo
- **Dimostrazione**. SC $\in$ **NP** è ovvio. Possiamo dimostrare che $VC \leq_m^p$ **SC**.

Hamiltonian Path è NP-completo

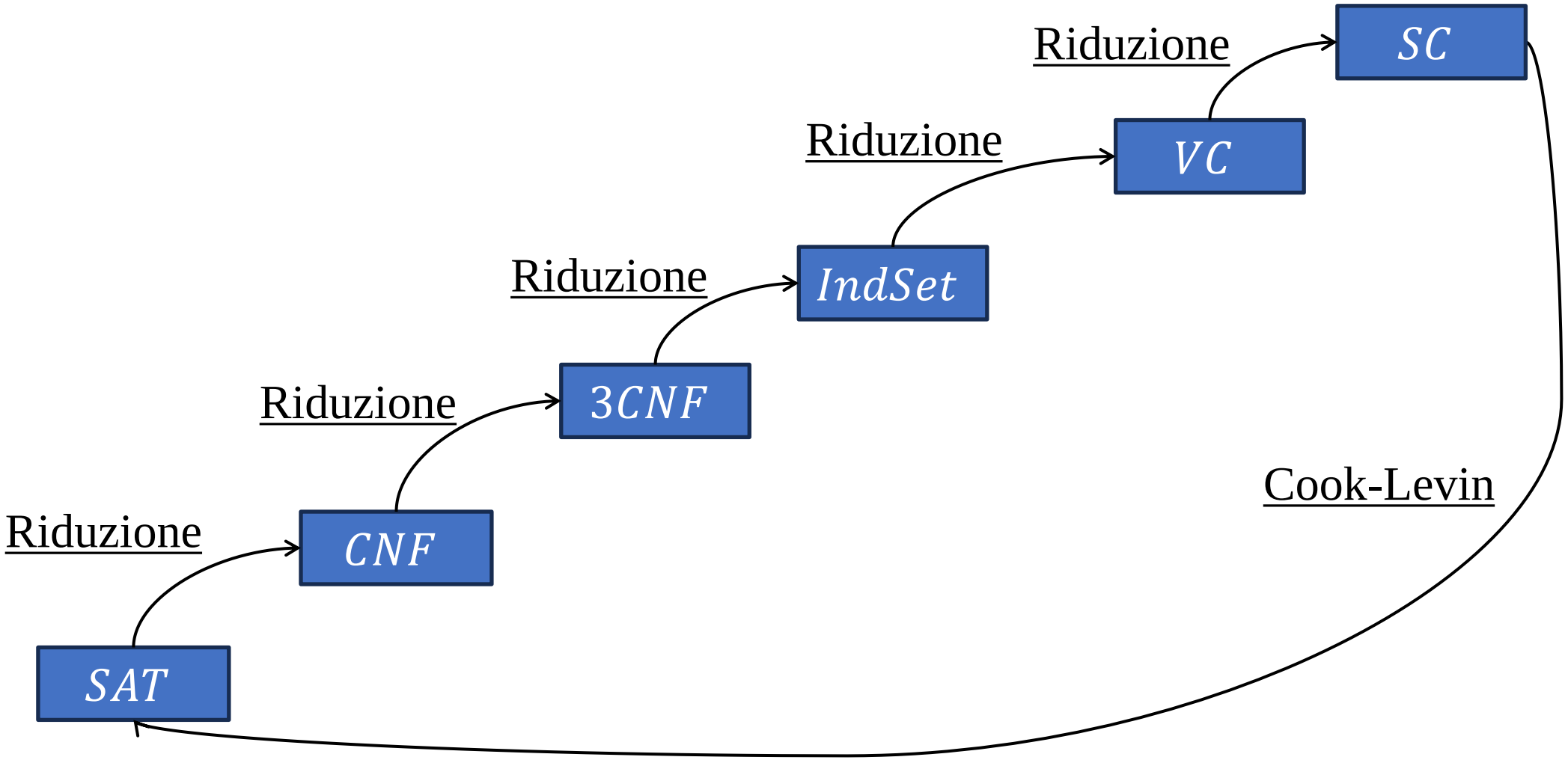
- **Definizione** Un **Set Cover** C di una famiglia di insiemi U è una famiglia $C \subseteq U$ tale che la sua unione è uguale all'unione U

- **Esempio**: $U = \{1, 2, 3, 4, 5\}$, $C = \{\{1, 2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}$, $k = 2$. In questo caso, abbiamo $\{1, 2, 3\} \cup \{4, 5\} = U$
- **Esempio**: $U = \{1, 2, 3, 4, 5\}$, $C = \{\{1, 2, 3\}, \{2, 4\}, \{3, 4\}, \{5\}\}$, $k = 2$. In questo caso, non esistono due insiemi in C la cui unione fa U

- **Definizione.** SC è il linguaggio di stringhe definito come segue
  $\{e(U, k)$ | Esiste un Set Cover C di U con $|C| = k\}$

- **Proposizione.** SC è **NP**-completo
- **Dimostrazione**. SC $\in$ **NP** è ovvio. Possiamo dimostrare che $VC \leq_m^p$ **SC**.

Linguaggi NP-Complete Visti Fino a Ora



Hamiltonian Path è NP-Completo

- **Definizione.** Un grafo è una coppia $\langle V, E \rangle$ dove V è un insieme (nodi) e E è un insieme di coppie non ordinate di elementi di V (archi)

- **Definizione.** Un cammino (path) p in un grafo G è una sequenza di archi $e_1, e_2, \dots, e_n$ di G tale che $e_i \cap e_{i+1} \neq \emptyset$, per $i = 1, \dots, n$ e ogni nodo compare in al più un arco della sequenza.

- **Definizione.** Un cammino (path) p in un grafo G è Hamiltoniano se ogni nodo del grafo G compare in almeno (esattamente) un arco della sequenza

- Fissiamo un Encoding ragionevole $\boldsymbol{e}$ per i grafi non orientati
  - Simboli per i nodi e coppie per gli archi oppure matrice di incidenza

- **Definizione.** Il linguaggio di stringhe P_H è definito come segue

$$\{e(G) \mid G \; contiene \; un \; cammino \; hamiltoniano\}$$

- **Proposizione.** P_H è in $\mathbf{NP - completo}$