Università degli Studi di Verona

Complessità

Riassunto dei principali argomenti

Matteo Danzi, Davide Bianchi

30 aprile 2018

Indice

1 Introduzione

1.1 Cos'è la complessità computazionale

Nella teoria della complessità ci si pone la seguente domanda:

> *Come scalano le risorse necessarie per risolvere un problema all'aumentare delle dimensioni del problema?*

La teoria della *complessità computazionale* è una parte dell'informatica teorica che si occupa principalmente di classificare i problemi in base alla quantità di *risorse computazionali* (come il tempo di calcolo e lo spazio di memoria) che essi richiedono per essere risolti. Tale quantità è detta anche *costo computazionale* del problema.

1.2 Problemi *facili* e *difficili*

Vediamo quattro esempi di problemi che classificheremo come facili o difficili:

1. (**Eulerian Cycle**) Esiste un modo per attraversare ogni arco di un grafo una e una sola volta?

    - Il problema si può vedere anche nella forma più piccola del problema dei *sette ponti di Königsberg*:
      A Königsberg ci sono 7 ponti, esiste un percorso che attraversa tutti i ponti una e una sola volta per poi tornare al punto di partenza?
      Se avessi n ponti e su ogni riva partissero 2 ponti avrei 2^n possibili percorsi.
    - La **soluzione di Eulero** dice che un grafo connesso non orientato ha un percorso che parte e inizia esattamente nello stesso vertice e attraversa ogni arco esattamente una volta se e solo se ogni vertice ha grado dispari (grado = numero di archi uscenti).
      Se ci sono esattamente due vertici v e u, di grado dispari, allora esiste un percorso che parte da u e attraversa ogni arco esattamente una volta e finisce in v.
    - Seguendo quindi la soluzione di Eulero, *quanto costa decidere* se un grafo G ha un tour Euleriano?

    ```
    odd-vertex-num = 0;
    For each vertex v of G
        if (deg(v) is odd)
            increment odd_vertex-num
    If(odd-vertex-num is neither 0 nor 2)
        output no Eulerian tour
    output Eulerian
    ```

    Questo algoritmo ha complessità: $O(|E| + |V|)$
    Il costo e l'algoritmo sono gli stessi se vogliamo *provare* che G *non* ha un tour Euleriano.

2. (**Hamiltonian Cycle**) Esiste un modo per attraversare ogni nodo di un grafo una e una sola volta?

    Esistono diverse soluzioni:

    - Provo tutte le possibilità ogni volta, costo: $O(2^n)$
    - Provo tutte le possibili permutazioni, costo: $O(n!)$
    - La soluzione migliore ad oggi è: $O(1.657^n)$

    Alla domanda: *Quanto costa decidere se un grafo ha un tour hamiltoniano?* Non sappiamo rispondere. Non sappiamo dire se il problema ha una soluzione non esponenziale. Per quanto ne sappiamo meglio di $O(1.657^n)$ non sappiamo fare.

    Non sappiamo nemmeno dire se Hamiltonian Cycle è più difficile di Eulerian Cycle.

3. N è un numero primo?

    Il migliore algoritmo conosciuto per decidere se N è un numero primo impiega $O((\log N)^{6+\epsilon})$

4. Quali sono i fattori primi di un numero?

   Ad oggi non conosciamo una procedure per fattorizzare un numero molto grande nei suoi divisori, che non sia provare tutte le possibilità.

1.3 Risolvere vs Verificare

La seguente tabella riassume in modo generico quanto detto nella sezione precedente riguardo alla difficoltà di risolvere problemi e verificare tali problemi su un istanza.

Tabella 1: Risolvere vs Verificare

Problema	**Risolvere**	**Verificare**
Eulerian Cycle	*facile*	*facile*
Hamiltonian Cycle	*difficile?*	*facile*
N è primo?	*facile*	*facile*
N ha un numero piccolo di fattori?	*difficile?*	*facile*

2 Problema computazionale

Un problema computazionale è una semplice relazione p che mappa l'insieme *infinito* di possibili input (domande o istanze) con un insieme *finito* (non vuoto) di output, cioè di risposte o soluzioni alle istanze.

$$\text{p} : \text{ istanze infinite } \mapsto \text{ soluzioni finite alle istanze}$$

Un problema computazionale non è una singola domanda, ma è una **famiglia di domande**:

- Una domanda per ogni possibile istanza
- Ogni domanda è dello stesso tipo (appartiene alla stessa classe)

Esempio 2.0.1. Il seguente esempio è un problema computazionale:

- Input: Qualsiasi grafo G
- Domanda: Il grafo G contiene un ciclo Euleriano?

Esempio 2.0.2. Il seguente esempio *non* è un problema computazionale:

- Domanda: È vero che il bianco vince sempre a scacchi, sotto l'ipotesi della giocata perfetta?

Non è un problema computazionale perché non ho un insieme infinito di possibili partite in input.

2.1 Risolvere un problema computazionale

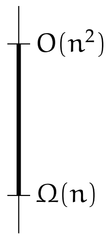
Risolvere un problema computazionale significa trovare un **algoritmo**, cioè una procedura che risolve il problema matematico in un numero finito di passi (di computazione elementare), che solitamente include la ripetizione di un operazione. È un procedimento deterministico che mappa l'input sull'output.

> Un algoritmo è una procedura *finita*, *definita*, *efficace* e con un input e un output.
>
> *Donald Knuth – The Art of Computer Programming*

2.2 Complessità di un problema computazionale

Misura della complessità. Come misuro la complessità di un problema computazionale? Come faccio a dire quanto è facile rispetto ad altri problemi?

- Do un **upper bound**: trovo un algoritmo qualsiasi che risolve il problema in modo da calcolare qual è il suo costo.
- Do un **lower bound**: trovo la minima quantità di risorse che ogni algoritmo utilizza per risolvere il problema. Tutti gli algoritmi sono *al minimo* complessi come il limite inferiore che abbiamo stabilito. Nessuno può fare di meglio.



2.3 Trattabilità di un problema.

La crescita della complessità di un problema è riducibile a 2 categorie fondamentali.

Crescita polinomiale. Un problema ha crescita polinomiale quando le risorse necessarie alla sua risoluzione sono limitate ad n^k, per qualche k. Se la taglia del problema aumenta, la sua complessità aumenta di un qualche fattore costante. Infatti, se la taglia dell'input va da n a $2n$ allora la complessità del problema si modifica in $(2n)^k = 2^k n^k$, ovvero aumenta di un fattore 2^k (costante). Raggruppiamo nella classe **P** i problemi di questo tipo.

Crescita esponenziale. Un problema ha crescita esponenziale la necessità di risorse necessarie alla sua risoluzione è proporzionale a c^n, per qualche costante $c > 1$. Se la taglia dell'input va da n a $2n$ c^n allora la richiesta di risorse si diventa $c^{2n} = c^n * c^n$, aumentando quindi di un fattore che cresce con l'aumentare di n. Raggruppiamo nella classe **Exp** i problemi di questo tipo.

3 Le classi di problemi computazionali

Notazione e idee di base. Formalmente definiamo un problema come un elemento $\mathbb{A}$ di una relazione

$$\mathcal{R} \subseteq \mathcal{I}(\mathbb{A}) \times \mathtt{Sol}$$

dove:

- $\mathcal{I}(\mathbb{A})$ è l'insieme delle istanze del problema $\mathbb{A}$
- $\mathtt{Sol}$ è l'insieme delle soluzioni delle istanze di $\mathbb{A}$

Si può quindi dire che

$$\forall x \in \mathcal{I}(\mathbb{A}),\ \mathtt{Sol}(x) = \{\text{Soluzioni di } x\}$$

Non è restrittivo restringersi ai **problemi di tipo decisionale**, ovvero quei problemi che hanno come soluzione una risposta del tipo *si* o *no*, quindi i problemi del tipo

$$\mathbb{A} : \mathcal{I}(\mathbb{A}) \to \{yes, no\}$$

L'algoritmo $\mathcal{A}$ per un problema $\mathbb{A}$ è un algoritmo che dato il problema, $\forall x \in \mathcal{I}(\mathbb{A})$, $\mathcal{A}(x) = \mathbb{A}(x)$. Inoltre, dato un algoritmo $\mathcal{A}$, definiamo $T_{\mathcal{A}}(|x|)$ la sua **complessità**, cioè il *tempo che impiega* $\mathcal{A}$ sull'istanza di taglia $|x|$. Notare che $|x|$ è la taglia dell'istanza x.

3.1 Classe P

Intuitivamente la classe **P** è definita come la classe di problemi di **complessità polinomiale**. Introduciamo qui la definizione formale.

Definizione 3.1.1 (Classe P)**.** Definiamo la classe di problemi **P** come l'insieme dei problemi di complessità polinomiale, ovvero

$$\mathbf{P} = \left\{ \mathbb{A} \mid \exists \mathcal{A} \text{ t.c. } \exists c \text{ costante e } \forall x \in \mathcal{I}(\mathbb{A}),\ \ \mathcal{A}(x) = \mathbb{A}(x) \ \text{ e } \ T_{\mathcal{A}}(|x|) \leqslant |x|^c \right\}$$

Esempio 3.1.1 (Eulerian Cycle)**.** Un semplice esempio di problema appartenente alla classe **P** è il problema del tour euleriano. Per questo problema infatti abbiamo che è un problema computazionale di decisione:

- Input: grafo G
- Output: $yes \Leftrightarrow \exists$ Eulerian Cycle in G.

Come abbiamo già visto quindi:

$$\exists \mathcal{A} \text{ t.c. } T_{\mathcal{A}}(|G|) = O(|E| + |V|) = O(|G|)$$

Eulerian Cycle $\in$ **P** perché $\exists \mathcal{A}$ che impiega un tempo che è nell'ordine della taglia di G, in particolare $\exists c$ costante dove $c = 1$.

Esempio 3.1.2 (Hamiltonian Cycle)**.** Ci chiediamo allora se anche Hamiltonian Cycle $\in$ **P** ? La risposta è che non lo sappiamo dire. Quello che sappiamo per questo problema è che:

$$\exists \mathcal{A} \text{ t.c. } T_{\mathcal{A}}(|G|) = O(a^{|G|})$$

dove a è costante.

3.2 Classe Exp

Dal momento che non sappiamo se alcuni problemi stiano oppure no nella classe **P** (dal momento che non si conosce un algoritmo che li risolva in tempo polinomiale), si definisce la classe **Exp**, che racchiude tutte le istanze di questa tipologia di problemi di **complessità esponenziale**.

Definizione 3.2.1 (Classe Exp)**.** Definiamo la classe di problemi **Exp** come la classe di problemi di complessità esponenziale, ovvero

$$\mathbf{Exp} = \left\{ \mathbb{A} \mid \exists \mathcal{A} \text{ t.c. } \forall x \in \mathcal{I}(\mathbb{A}),\ \mathcal{A}(x) = \mathbb{A}(x) \ \text{ e } \ T_{\mathcal{A}}(|x|) \leqslant 2^{|x|^c} \right\}$$

Esempio 3.2.1 (Hamiltonian Cycle)**.** Ci chiediamo se Hamiltonian Cycle $\in$ **Exp** ? Se prendiamo l'algoritmo che prova tutte le combinazioni di archi cioè $\binom{|E|}{n}$ per vedere se formano un ciclo hamiltoniano. La complessità di quest'algoritmo è al massimo $2^{|E|^2}$.

Se invece prendiamo l'algoritmo che considera tutte le possibili permutazioni dei vertici del grafo abbiamo che la complessità è $n!$. Quindi il problema Hamiltonian Cycle $\notin$ **Exp**

Relazione tra P ed Exp. La domanda che sorge spontanea è **P** $\subseteq$ **Exp** ?

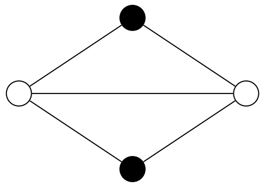
La risposta alla domanda è banalmente **si**, in quanto, dato un algoritmo $\mathcal{B}$ con complessità $T_{\mathcal{B}}(|x|)$, possiamo dire che

$$T_{\mathcal{B}}(|x|) = O(|x|^c) = O(2^{|x|^c}) \Rightarrow \mathbb{A} \in \mathbf{Exp}$$

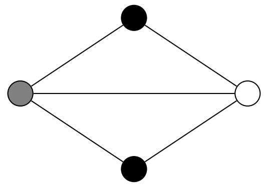
Problema K-Graph-Colouring. Analizziamo ora il problema della K-colorabilità di un grafo G:

- Input: G non orientato.
- Output: $yes \Leftrightarrow \exists$ colorazione *propria* dei vertici di G ovvero:

$$\exists f :\ v \mapsto \{0, \dots, k-1\} \quad \text{t.c.} \quad \forall (u, v) \in E(G) \quad f(u) \neq f(v)$$



(a) Grafo con colorazione *non* propria



(b) Grafo con colorazione propria

Problema 2-Graph-Colouring. Consiste nel trovare se esiste una 2 colorazione del grafo dato in input in modo tale che un arco non si trovi tra due vertici dello stesso colore. Questo problema corrisponde a dire se il grafo è **bipartito**, cioè se *posso suddividere il grafo in due classi diverse*.
Per vedere se è bipartito si effettua una **BFS**, cioè una visita in ampiezza, e si controlla se c'è un ciclo dispari. Se c'è allora non è bipartito e quindi nemmeno 2-colorabile.

È 2-colorabile $\Leftrightarrow$ è Bipartito $\Leftrightarrow$ non contiene un ciclo dispari. La visita BFS ha una complessità pari a $O(|E| + |V|)$, perciò il problema è risolvibile in tempo polinomiale, perciò possiamo concludere che 2-Graph-Colouring $\in$ **P**.

Problema 3-Graph Colouring Il problema 3-Graph Colouring $\in$ **P**? Non sappiamo rispondere a questa domanda, poiché non sappiamo se esiste un algoritmo che lo svolga in tempo polinomiale. Il problema 3-Graph Colouring $\in$ **Exp**? Se consideriamo l'algoritmo che prova tutte le possibili colorazioni abbiamo che:

$$3^n \text{ sono le colorazioni dei vertici, dove } n = |V(G)|$$

Bisogna vedere se ci sono archi monocolore e quindi la complessità diventa:

$$O(3^n \cdot |E|) = O(3^{2n}) = O((2^{\log_2 3})^{2n}) = O(2^{2n \log_2 3})$$

Perciò possiamo concludere che il problema 3-Graph Colouring $\in$ **Exp**.

3.3 Classe Time(n)

Definizione 3.3.1 (Classe Time(n))**.** Definiamo la classe **Time**(**n**) come l'insieme dei problemi di complessità lineare, ovvero

$$\mathbf{Time}(\mathbf{n}) = \left\{ \mathbb{A} \mid \exists \mathcal{B} \text{ per } \mathbb{A} \quad \text{t.c.} \quad \forall x \in \mathcal{I}(\mathbb{A}) \quad T_{\mathcal{B}}(|x|) = O(n) = O(f(|x|)) \right\}$$

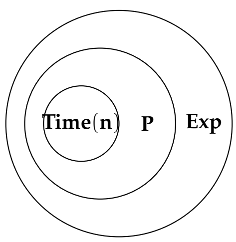
Teorema 3.3.1. $\forall \mathcal{B} \quad \text{t.c.} \quad \mathcal{B}(x) = \mathbb{A}(x) \quad T_{\mathcal{B}}(|x|) > |x|^c \quad \forall c$ *costante*

Teorema 3.3.2. *Qualsiasi* ***algoritmo di ordinamento*** *che usa confronti su* n *elementi ha tempo di esecuzione pari a*

$$\Omega(n \log n)$$

Possiamo dire quindi che:

- **Eulerian Cycle** $\in$ **Time**(**n**) perché esiste un problema che lo risolve in tempo lineare.
- **Sorting** $\notin$ **Time**(**n**) per il teorema 3.3.2.



Possiamo riassumere quindi che:

- **Eulerian Cycle** $\in$ **P**, **Eulerian Cycle** $\in$ **Time**$(\mathbf{n})$.
- **Hamiltonian Cycle** $\in$ **Exp**
- **Hamiltonian Cycle** $\in$ **P** ? non lo sappiamo dire.
- **K-Colouring** $\in$ **Exp**
- **K-Colouring** $\in$ **P**?
  per $k \geqslant 3$ non lo sappiamo dire
  per $k = 2$ sì.

Inoltre, con la definizione della classe **Time**$(\mathbf{n})$ si può dire che:

$$\mathbf{P} = \bigcup_{k \geqslant 0} \mathbf{Time}(\mathbf{n}^k)$$

$$\mathbf{Exp} = \bigcup_{k \geqslant 0} \mathbf{Time}(\mathbf{2}^{\mathbf{n}^k})$$

3.4 Classe NP

La classe **NP** (*non deterministic polinomial time*) è la classe di problemi tali che se la soluzione per un'istanza del problema è *yes*, allora è facile verificarlo.

Definizione 3.4.1. (Classe NP)

$$\begin{aligned}\mathbf{NP} = \{\mathbb{A} \quad | \quad & \exists \mathcal{B}(\overset{x}{\cdot}, \overset{w}{\cdot}) \quad \text{t.c.} \quad T_{\mathcal{B}}(|x| + |w|) = O((|x| + |w|)^c) \\ & \forall x \in \mathcal{I}(\mathbb{A}) \quad \mathbb{A}(x) = yes \Leftrightarrow \exists w \text{ t.c.} \quad |w| = O(|x|^d) \text{ e } \mathcal{B}(x, w) = yes\}\end{aligned}$$

dove:

- $\mathcal{B}(\overset{x}{\cdot}, \overset{w}{\cdot})$ è detto **verificatore** per $\mathbb{A}$. Se la risposta di $\mathbb{A}$ esiste, allora $\mathcal{B}$ dice *yes*. Il verificatore impiega **tempo polinomiale** nella taglia dell'istanza per rispondere.
- x è l'istanza
- w è il certificato.

Hamiltonian Cycle $\in$ NP ? Per vedere se il problema Hamiltonian cycle appartiene alla classe **NP** dobbiamo costruire un verificatore $\mathcal{B}$ che agisca in tempo polinomiale.

```
VerifyHamCycle(G = (V,E), C=x_1,...,x_n)
    if r != |V|: return no                          } O(|w|)
    for each v∈ V                                   }
        if v non appare in C: return no             } O(|V| · |C|)
    for i=1 to n-1                                  }
        if (x_i, x_{i+1}) ∉ E: return no            } O(|C|)
    if (x_1, x_n) ∉ E: return no                    } O(1)
    return yes
```

Il tempo di esecuzione del verificatore è polinomiale e quindi posso dire che Hamiltonian Cycle $\in$ **NP** .

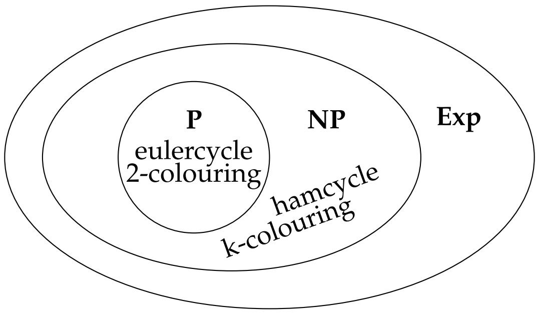
K-Colouring $\in$ NP ? Per vederlo costruisco il verificatore:

```
VerifyK-Colouring(G = (V,E), f(v_1),...,f(v_n))
    for each E(u,v)
        if f(u) = f(v): return no          } O(|E|)
    for i=1 to n
        if f(v_i) >= K: return no          } O(|V|)
    return yes
```

Il tempo di esecuzione del verificatore è polinomiale e quindi posso dire che K-Colouring $\in$ **NP** .

P $\subseteq$ NP ? Vogliamo capire in che classe è **NP**. Se include la classe **P** allora significa che un problema che appartiene a quest'ultima, se lo sappiamo risolvere, lo sappiamo anche verificare. Infatti se $\mathbb{A} \in$ **P** dobbiamo dimostrare che esiste un verificatore. Tale verificatore per $\mathbb{A}$ sarà: $\mathcal{B}'(x, w) = \mathcal{B}(x)$ privo di certificato. Dobbiamo dimostrare che se l'istanza è *yes* allora $\mathcal{B}(x) = yes$ altrimenti $\mathcal{B}(x) = no$.

NP $\subseteq$ Exp ? Vogliamo capire in che classe è **NP**
Possiamo supporre che **P** $\subseteq$ **NP** $\subseteq$ **Exp**.



4 Riduzione alla Karp tra problemi di decisione

Definizione 4.0.2 (Riduzione alla Karp)**.** Un problema di decisione $\mathbb{A}$ si riduce alla Karp al problema $\mathbb{B}$: $\mathbb{A} \leqslant_K \mathbb{B}$ se esiste un algoritmo polinomiale $\mathcal{A}$ tale che

$$\forall x \in \mathcal{I}(\mathbb{A}),\ \mathbb{B}(\mathcal{A}(x)) = yes \ \Leftrightarrow\ \mathbb{A}(x) = yes$$

Proposizione 4.0.1. Se $\mathbb{A} \leqslant_K \mathbb{B}$ e $\mathbb{B} \in$ **P** $\Rightarrow$ $\mathbb{A} \in$ **P**

Proposizione 4.0.2. Se $\mathbb{A} \leqslant_K \mathbb{B}$ e $\mathbb{B} \notin$ **P** $\Rightarrow$ $\mathbb{A} \notin$ **P**

Come effettivamente svolgiamo le trasformazioni?

4.1 Problema SAT

Definizione 4.1.1 (SAT)**.** Il problema di soddisfacibilità di una formula booleana è definito nel seguente modo:

- Input: formula booleana : $\phi(x_1, \dots, x_n) = C_1 \wedge C_2 \wedge \dots \wedge C_n$
  Dove:
  - $C_i = l_{i1} \vee l_{i2} \vee \dots \vee l_{ik}$ (clausola)
  - $l_{ij} = x_k$ oppure $\bar{x}_k$ (letterale)
- Output: *yes* $\Leftrightarrow$ $\exists a_1 \dots a_n \in T, F^n$ t.c. $\phi(a_1, \dots, a_n) = T$

Esempio 4.1.1. $\phi(x_1, x_2, x_3) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee \bar{x}_3)$
Assegnamento che soddisfa la formula booleana $\phi(x_1, x_2, x_3)$:

$$x_1 = T \quad x_2 = F \quad x_3 = F$$
$$a_1 = T \quad a_2 = F \quad a_3 = F$$

SAT $\in$ NP ? Ci chiediamo se il problema SAT sta nella classe **NP**. Vediamo dunque se esiste un certificato e un verificatore che attesta, dato una formula booleana, se essa è soddisfacibile in tempo polinomiale.

- Si può notare facilmente che il certificato è un assegnamento per la formula booleana, dunque è polinomialmente correlato alla grandezza delle variabili della formula, sarà al massimo n.
- Il verificatore viene costruito analizzando la formula booleana, controllando ogni letterale di ciascuna clausola. Ho quindi $m \times n \times n$ controlli, dove m = numero di clausole, n = numero di letterali. Il verificatore è quindi polinomiale.

Possiamo concludere che il problema SAT $\in$ **NP**. Questa affermazione si può tradurre con: *data una formula booleana di cui sappiamo essere soddisfacibile, allora è facile (polytime) costruire un verificatore che attesta che essa è SAT.*

Problema K-SAT: è il problema SAT in cui l'input ha come restrizione il vincolo che ogni clausola ha esattamente k letterali.

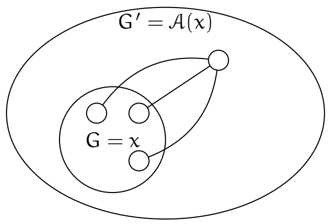
Esempio 4.1.2 (3-SAT). $\phi(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$

4.2 Alcuni esempi di riduzioni tra problemi

K-colouring $\leqslant_K$ (K+1)-colouring Vediamo se il problema (K+1)-colouring non è più facile del problema K-colouring. Dobbiamo in sostanza dimostrare che decidere se possiamo colorare un grafo con $k+1$ colori non è più facile che decidere se possiamo colorare un grafo con k colori.
N.B.: da notare che i due grafi non sono necessariamente uguali, parliamo di qualsiasi grafo che appartiene al problema.

$$\mathcal{A} : \quad x \in \mathcal{I}(K - COL) \quad \mapsto \quad \mathcal{A}(x) \in \mathcal{I}((K+1) - COL)$$
$$K - COL(x) = yes \quad \Leftrightarrow \quad (K+1) - COL(\mathcal{A}(x)) = yes$$

Prendiamo quindi il grafo G':



per cui

$$G = (V, E)$$
$$G' = (V \cup \{v'\}, E \cup (v, u') \mid v \in V)$$

in tempo lineare e quindi sotto il polinomiale riesco a costruire il grafo G'.
Se G è K-colorabile allora G' è (K+1)-colorabile. Mi basta assegnare a v' il colore k (il k+1-esimo colore) e mantenere la colorazione di G.
Se G non è K-colorabile allora G' non è K+1-colorabile. Equivale a dire che se G' è K+1-colorabile allora G è k-colorabile. Quindi se v' ha un colore $f(v') = x$ allora ogni $v \in V(G)$ ha un colore $f(v') \neq x$, al più usano k colori.

Da questa dimostrazione ricaviamo anche che 2-col $\leqslant_K$ 3-col $\leqslant_K$ 4-col $\leqslant_K$ 5-col

SAT $\leqslant_K$ 3-SAT Vogliamo dimostrare che data una formula booleana ϕ CNF esiste una trasformazione polytime che mi porta a una formula booleana ϕ' 3CNF (ogni clausola ha esattamente 3 letterali). E inoltre che ϕ è soddisfacibile se e solo se ϕ' è soddisfacibile.

Possiamo iniziare dicendo che $(x_1 \vee x_2) \equiv (x_1 \vee x_1 \vee x_2)$. Le clausole più piccole possono essere espanse. Seguendo questa intuizione arriviamo a dire che:

$$\begin{aligned}
&(l_1 \vee l_2 \vee l_3 \vee \cdots \vee l_k) \quad \rightsquigarrow \\
&(l_1 \vee l_2 \vee z_1) \wedge (\bar{z}_1 \vee l_3 \vee z_2) \wedge (\bar{z}_2 \vee l_4 \vee z_3) \wedge (\bar{z}_3 \vee l_5 \vee z_4) \wedge \cdots \wedge (\bar{z}_{k-1} \vee l_{k+1} \vee z_k)
\end{aligned}$$

Dimostriamo che se ϕ non è soddisfacibile allora non lo è neanche ϕ'.

- Prendiamo $\phi = (x_1, \dots, x_n)$. Per questa formula prendiamo un assegnamento $a_1, \dots, a_n$ che non la rende soddisfacibile, quello in cui ogni letterale viene assegnato a F.
- Prendiamo dunque $\phi' = (x_1, \dots, x_n, z_1, \dots z_r)$. Per questa formula prendiamo lo stesso assegnamento di ϕ e vediamo cosa succede con i letterali z:

$$(\underset{F}{l_1} \vee \underset{F}{l_2} \vee \underset{V}{z_1}) \wedge (\underset{F}{\bar{z}_1} \vee \underset{F}{l_3} \vee \underset{V}{z_2}) \wedge (\underset{F}{\bar{z}_2} \vee \underset{F}{l_4} \vee \underset{V}{z_3}) \wedge (\underset{F}{\bar{z}_3} \vee \underset{F}{l_5} \vee \underset{V}{z_4}) \wedge \cdots \wedge (\underset{F}{\bar{z}_{k-1}} \vee \underset{F}{l_{k+1}} \vee \underset{F}{z_k})$$

risulta che l'ultimo letterale z_k è falso, e quindi ϕ' non è soddisfacibile.

K-COL $\leqslant_K$ K-SAT Vogliamo dimostrare che il problema di colorare un grafo con k colori è riducibile al problema di soddisfacibilità di una formula booleana k-CNF.

Cerchiamo un modo per esprimere in modo logico il fatto che due nodi adiacenti non abbiano lo stesso colore. Supponiamo che il nodo v abbia colore i e il nodo u abbia colore i con $i = 0, 1, \dots, k-1$. Per ogni $v \in V$: $x_0^{(v)}\ x_1^{(v)}\ x_2^{(v)} \dots x_{k-1}^{(v)}$ dove $x_i^{(v)} = T$ se il vertice v ha colore i.

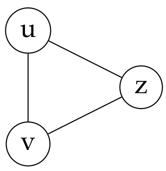
Ci chiediamo quindi quand'è che la formula è K-colorabile?

$$\forall v \in V \begin{cases} x_0^{(v)} \vee x_1^{(v)} \vee x_2^{(v)} \vee \cdots \vee x_{k-1}^{(v)} & \text{ogni vertice ha un colore} \\ \\ \overline{x_i^{(v)} \wedge x_j^{(v)}} = \overline{x_i^{(v)}} \vee \overline{x_j^{(v)}} \quad \forall i, j & \end{cases}$$

$$\forall e = (u, v) \in E \ \text{ i due vertici non devono avere lo stesso colore}$$

$$\forall i \quad \overline{x_i^{(v)} \wedge x_i^{(u)}} = \overline{x_i^{(v)}} \vee \overline{x_i^{(u)}}$$

Esempio 4.2.1. Prendiamo per esempio il seguente grafo:



La formula booleana corrispondente sarà:

$$\begin{aligned}
\text{Ogni vertice ha un colore}&\begin{cases}
(x_0^{(u)} \vee x_1^{(u)} \vee x_2^{(u)}) \wedge \overbrace{(\overline{x_0^{(u)}} \vee \overline{x_1^{(u)}}) \wedge (\overline{x_0^{(u)}} \vee \overline{x_2^{(u)}}) \wedge (\overline{x_1^{(u)}} \vee \overline{x_2^{(u)}})}^{\text{Un vertice non può avere 2 colori}} \wedge \\
(x_0^{(v)} \vee x_1^{(v)} \vee x_2^{(v)}) \wedge (\overline{x_0^{(v)}} \vee \overline{x_1^{(v)}}) \wedge (\overline{x_0^{(v)}} \vee \overline{x_2^{(v)}}) \wedge (\overline{x_1^{(v)}} \vee \overline{x_2^{(v)}}) \wedge \\
(x_0^{(z)} \vee x_1^{(z)} \vee x_2^{(z)}) \wedge (\overline{x_0^{(z)}} \vee \overline{x_1^{(z)}}) \wedge (\overline{x_0^{(z)}} \vee \overline{x_2^{(z)}}) \wedge (\overline{x_1^{(z)}} \vee \overline{x_2^{(z)}}) \wedge
\end{cases}\\
\text{Ogni arco ha colori diversi}&\begin{cases}
(\overline{x_0^{(v)}} \vee \overline{x_0^{(u)}}) \wedge (\overline{x_1^{(v)}} \vee \overline{x_1^{(u)}}) \wedge (\overline{x_2^{(v)}} \vee \overline{x_2^{(u)}}) \wedge \\
(\overline{x_0^{(v)}} \vee \overline{x_0^{(z)}}) \wedge (\overline{x_1^{(v)}} \vee \overline{x_1^{(z)}}) \wedge (\overline{x_2^{(v)}} \vee \overline{x_2^{(z)}}) \wedge \\
(\overline{x_0^{(z)}} \vee \overline{x_0^{(u)}}) \wedge (\overline{x_1^{(z)}} \vee \overline{x_1^{(u)}}) \wedge (\overline{x_2^{(z)}} \vee \overline{x_2^{(u)}})
\end{cases}
\end{aligned}$$

La trasformazione è polinomiale? La complessità della trasformazione è:

$$|V| \cdot \left(K + 2\binom{k}{2}\right) + |E|K \cdot 2 \quad \leqslant \quad (|E| + |V|)K^2$$

Quindi è polinomiale.

4.3 Problema NAE-K-SAT

NAE-K-SAT (Not All Equivalent-K-SAT):

- Input: ϕ K-CNF $\quad \phi : \{T, F\}^n \mapsto \{T, F\}$
- Output: $yes \Leftrightarrow \exists \underline{a} \in \{T, F\}^n$ t.c. $\phi(\underline{a}) = T$ e, in ogni clausola $C_i = l_1^{(i)} \vee l_2^{(i)} \vee \cdots \vee l_k^{(i)}$ con $\underline{a}$, almeno un $l_j^{(i)}$ è vero e almeno un $l_j^{(i)}$ è falso.

Esempio 4.3.1.

$$\phi(x_1, x_2, x_3) = (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$

$x_1 = F \quad x_2 = F \quad x_3 = F$ *non* è NAE-K-SAT
$x_1 = F \quad x_2 = T \quad x_3 = F$ è NAE-K-SAT

Proposizione 4.3.1. Se $\underline{a}$ è un assegnamento che soddisfa ϕ (è NAE), allora anche il negato $\overline{\underline{a}}$ soddisfa ϕ (è NAE).

3-SAT $\leqslant_K$ NAE-4-SAT Vogliamo dimostrare che data una qualsiasi formula ϕ 3-CNF la trasformo in una formula ψ 4-CNF in tempo polinomiale.

$$\phi \text{ 3-CNF} \longmapsto \psi \text{ 4-CNF}$$

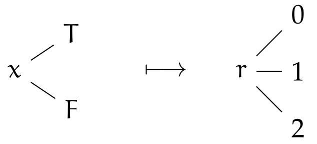
$$\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n \quad C_i = l_1^{(i)} \vee l_2^{(i)} \vee l_3^{(i)} \quad i = 1 \ldots n$$
$$\psi = C_1' \wedge C_2' \wedge \cdots \wedge C_n' \quad C_i' = l_1^{(i)} \vee l_2^{(i)} \vee l_3^{(i)} \vee z \quad i = 1 \ldots n$$

Per creare ψ espando le variabili e ne aggiungo sempre una. La trasformazione da ϕ a ψ è polinomiale nella taglia della formula ϕ, perché la scorro tutta per creare ψ.
Ora dobbiamo dimostrare che se ϕ è soddisfacibile allora anche ψ è soddisfacibile:

- ϕ è soddisfacibile $\Rightarrow \exists \underline{a} \in \{T, F\}^n$ t.c. $\phi(\underline{a}) = T$.
- Se prendiamo l'assegnamento $\underline{b} = \underline{a} \quad z = F \quad \psi(\underline{b}) = T$ e ogni clausola ha un letterale a FALSE.
- Vogliamo dimostrare che se esiste un assegnamento $\underline{b}$ che soddisfa ψ allora esiste un assegnamento $\underline{a}$ che soddisfa ϕ.
- Se secondo $\underline{b}$ $z = F$ allora, la parte rimanente di $\underline{b}$ soddisfa ψ
- Se secondo $\underline{b}$ $z = T$ allora, lo nego e torno al primo caso. Perciò se ψ è nae-soddisfatta con $z = F$ allora ϕ è soddisfatta.

NAE-3-SAT $\leqslant_K$ 3-COL Vogliamo dimostrare che data la formula ϕ 3-CNF esiste una trasformazione polinomiale che la rende un grafo G tale che ϕ è NAE-soddisfacibile se e solo se il grafo G è 3-colorabile.

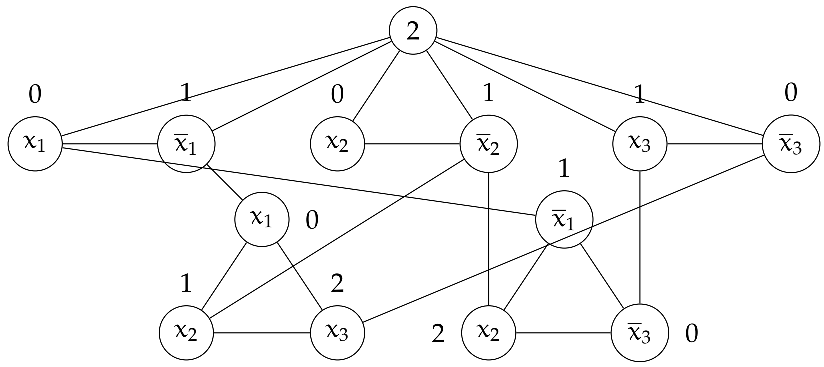
Mappo variabili (letterali) che possono valere T o F, su vertici (elementi del grafo) che hanno colore $0, 1, 2$.



Partendo dalla formula $\phi(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3)$ costruiamo il grafo nel seguente modo:

- Creo un nodo per ogni letterale e per il suo negato, poi aggiungo un vertice perché per ogni vertice x uso la stessa coppia di colori.
- Per ogni clausola metto un triangolo che corrisponde ai letterali della clausola
- Se ho una 3-colorazione ho un assegnamento corrispondente per la clausola che mi mette un letterale T e uno F.
- Ora aggiungo gli archi, collego i letterali che hanno valori di verità opposti.

Se associamo $0 \mapsto T$, $1 \mapsto F$, e 2 libero, abbiamo il seguente risultato:



Perciò la trasformazione garantisce che se $\exists \underline{a}$ t.c. $\phi(\underline{a})$ è nae-soddisfatta allora esiste una 3-colorazione per il grafo G che associa ai valori di verità i colori in modo tale da rendere G 3-colorabile. È facile vedere anche l'implicazione nel verso opposto.

4.4 Transitività della riduzione alla Karp

La riduzione $\leqslant_K$ è transitiva, ciò implica che:

$$\mathbb{A} \leqslant_K \mathbb{B} \text{ e } \mathbb{B} \leqslant_K \mathbb{C} \Rightarrow \mathbb{A} \leqslant_K \mathbb{C}$$

in particolare abbiamo che:

$$\begin{array}{ll} \mathbb{A} \leqslant_K \mathbb{B} & \exists \mathcal{A} \text{ polytime } x \in \mathcal{I}(\mathbb{A}),\ \mathcal{A}(x) \in \mathcal{I}(\mathbb{B}) \quad \mathbb{A}(x) = yes \Leftrightarrow \mathbb{B}(\mathcal{A}(x)) = yes \\ \mathbb{B} \leqslant_K \mathbb{C} & \exists \mathcal{B} \text{ polytime } y \in \mathcal{I}(\mathbb{B}),\ \mathcal{B}(y) \in \mathcal{I}(\mathbb{C}) \quad \mathbb{B}(y) = yes \Leftrightarrow \mathbb{C}(\mathcal{B}(y)) = yes \end{array}$$

Perciò

$$\forall x \in \mathcal{I}(\mathbb{A}),\ \mathcal{B}(\mathcal{A}(x)) \in \mathcal{I}(\mathbb{C}) \quad \mathbb{A}(x) = yes \Leftrightarrow \mathbb{C}(\mathcal{B}(\mathcal{A}(x))) = yes \quad \Rightarrow \quad \mathbb{C}(x) = \mathcal{B}(\mathcal{A}(x))$$

4.5 Problema Reachability

- Input: Grafo G diretto, due nodi s e t.
- Output: $yes \Leftrightarrow$ esiste un cammino che va da s a t.

Quanto costa risolvere Reachablity?
Una possibile soluzione potrebbe essere applicare BFS partendo da s. Se si trova t, allora ritorno yes, altrimenti no. Questo procedimento richiede $O(|V| + |E|)$. Quindi *Reachability* $\in$ **P**

5 Riduzione alla Turing tra problemi di decisione

Definizione 5.0.1 (Riduzione alla Turing)**.** $\mathbb{A} \leqslant_T \mathbb{B}$ se esiste un algoritmo con complessità polinomiale $\mathcal{A}$ che data un'istanza $x \in \mathcal{I}(\mathbb{A})$ utilizzando chiamate ad un *oracolo* per $\mathbb{B}$ che hanno costo $O(1)$, $\mathcal{A}(x) = \mathbb{A}(x)$.

6 Classe di problemi NP-Completi

Definizione 6.0.2. (Classe NPC) Un problema $\mathbb{A}$ è NP-completo (NPC) se

- $\mathbb{A} \in$ **NP**
- $\mathbb{A}$ è *NP-hard*, cioè se $\forall \mathbb{B} \in$ **NP** $\quad \mathbb{B} \leqslant_K \mathbb{A}$

6.1 Circuito Booleano

Definizione 6.1.1 (Circuito Booleano)**.** Un circuito booleano è un grafo aciclico orientato (DAG) C_n con n input e ha le seguenti caratteristiche:

- $\exists n$ vertici che hanno *in-degree* = 0
- $\exists 1$ vertice che ha *out-degree* = 0
- Ogni altro vertice ha *in-degree* = 1 o 2 ed è etichettato con and, or, not.
- La taglia di C_n è il numero di vertici.

Esempio 6.1.1. Per $n = 4$ abbiamo $C_4(x_1, x_2, x_3, x_4)$:

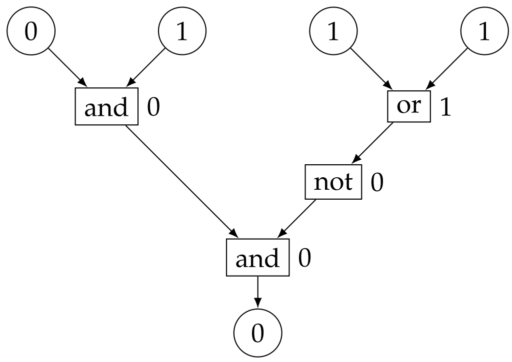


Figura 2: Esempio di circuito booleano con 4 input, il nodo finale di output è detto nodo *sink*.

6.2 Problema Circuit-SAT

- Input: Circuito booleano C_n
- Output: $yes \Leftrightarrow \exists \underline{x}$ t.c. $C(x) = 1$ (il circuito booleano è soddisfacibile).

Definiamo una famiglia di circuiti $C_{n \geqslant 0}$ (per ogni numero di input) di complessità $T(n)$ tale che la taglia di C_n è $O(T(n))$.

Vogliamo mappare il verificatore di ogni problema in **NP** in un circuito:

$$\mathbb{A} \qquad \longmapsto \qquad V(\cdot, \cdot)$$

$$\mathbb{A}(x) = yes \Leftrightarrow \exists w \text{ t.c. } V(x, w) = yes$$

Dove $V(x, w)$ è un circuito che prende x in input e che mi dice se esiste un certificato w tale che rende soddisfatto il circuito.

Teorema 6.2.1. *Se $\mathbb{A} \in \mathsf{TIME}(f(n))$ allora esiste una famiglia di circuiti $C_{n \geqslant 0}$ di complessità $T(n) = O(f(n)^2)$ tale che $\forall \underline{x} \in \mathcal{I}(\mathbb{A})$ e $n = |x|$ $C_n(x) = \mathbb{A}(x)$ e C_n è costruibile in tempo polinomiale.*

Corollario 6.2.1. *Se $\mathbb{A} \in$ **P** ($f(n)$ è un polinomio in $\mathsf{TIME}(f(n))$) allora esiste una famiglia di circuiti di complessità polinomiale ($T(n) = n^k$) tale che $\forall \underline{x} \in \mathcal{I}(\mathbb{A})$ e $n = |x|$ $C_n(\underline{x}) = \mathbb{A}(x)$ e C_n è costruibile in tempo polinomiale in $|x| = n$.*

Circuit SAT è NP-completo Dimostriamo prima a parole che Circuit-SAT $\in$ **NP**. Forniamo il verificatore $V(x, w)$ verifica se un'istanza soddisfa il problema. Il certificato w è l'assegnamento che soddisfa il circuito, mentre il verificatore scorre ogni nodo e ne valuta il valore, ritorna yes se il nodo finale (sink) è a 1, altrimenti no.

Ora dimostriamo che Circuit-SAT è NP-hard, ovvero che $\forall \mathbb{A} \in$ **NP** $\mathbb{A} \leqslant_K$ Circuit-SAT. Dobbiamo mostrare dunque che esiste tale trasformazione polinomiale:

$$\underline{x} \in \mathcal{I}(\mathbb{A}) \longmapsto C \in \mathcal{I}(\text{Circuit-SAT})$$

e vale anche che:

$$\mathbb{A} = yes \Leftrightarrow \exists w \text{ t.c. } C(w) = 1 (C \text{ è soddisfacibile})$$

Sia $\mathbb{A} \in$ **NP** allora $\exists V_{\mathbb{A}}(x, w)$ per le istanze $x \in \mathcal{I}(\mathbb{A})$, tale che $V_{\mathbb{A}}$ ha complessità $O(p(|x|)) = |w|$ (polinomiale). Allora per il teorema 6.2.1 sappiamo che esiste una famiglia di circuiti C_m che fa esattamente ciò che fa il verificatore $V_{\mathbb{A}}$:

$$C_m = V_{\mathbb{A}} \quad m = |x| + p(|x|)$$

perciò, se consideriamo $C'_x(x) = C_m(x, w)$

$$\mathbb{A}(x) = yes \Leftrightarrow \exists w \text{ t.c. } V_{\mathbb{A}}(x, w) = yes \Leftrightarrow \exists w \text{ t.c. } C_m(x, w) = 1 \Leftrightarrow \exists w \text{ t.c. } C'_x(x) = 1$$

SAT è NP-completo Vogliamo dimostrare che dato un circuito booleano soddisfacibile esiste una riduzione che lo trasforma in tempo polinomiale in una formula booleana soddisfacibile.

$$\begin{gathered} \text{Circuit-SAT} \leqslant_K \text{SAT} \\ \forall C \in \mathcal{I}(\text{Circuit-SAT}) \longmapsto \phi(\dots) \\ C \text{ è soddisfacibile} \Leftrightarrow \phi \text{ è soddisfacibile} \end{gathered}$$

Osservazione 6.2.1. Ogni funzione di gate ($and, or, not, \dots$) può essere espressa con una formula booleana CNF ϕ:

$$\begin{array}{ll} c = a \text{ and } b & (\bar{c} \vee a) \wedge (\bar{c} \vee b) \wedge (c \vee \bar{a} \vee \bar{b}) \\ c = a \text{ or } b & (\bar{c} \vee a \vee b) \wedge (c \vee \bar{b}) \wedge (c \vee \bar{a}) \\ c = \text{not } a & (\bar{c} \vee \bar{a}) \wedge (c \vee a) \end{array}$$

Quindi un circuito booleano è soddisfatto quando ogni formula è soddisfatta e il nodo sink è soddisfatto (= 1).

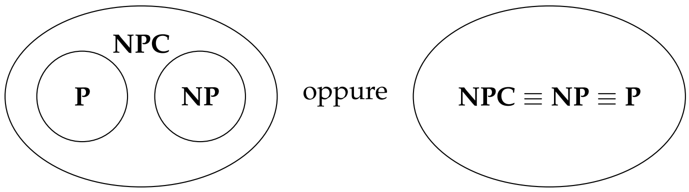
Perciò se ogni funzione di gate sottoforma di circuito booleano rappresenta ogni clausola della formula CNF ϕ, allora possiamo mettere in and tutte le clausole e dire che il circuito C è soddisfatto se e solo se ϕ è soddisfatta.
Con questo e con la dimostrazione che Circuit-SAT è NP-completo possiamo dire che

$$\forall \mathbb{B} \in \mathbf{NP} \quad \mathbb{B} \leqslant_K \text{Circuit-SAT} \leqslant_K \text{SAT}$$

Perciò, per la proprietà transitiva della riduzione alla Karp tra problemi di decisione, deduciamo che SAT è NP-completo.

6.3 Relazione tra P, NP, e NP-completo

Distinguiamo principalmente due casi che rappresentano le relazioni tra le classi di problemi **P**, **NP** e NP-completo:



Teorema 6.3.1. *Se* $\mathbf{NPC} \cap \mathbf{P} \neq \emptyset$ *e* $\mathbb{A} \in \mathbf{NP}$ t.c. $\mathbb{A}$ ***non è banale****, ovvero*

$$\exists x \in \mathcal{I}(\mathbb{A}) \quad \text{t.c. } \mathbb{A}(x) = yes$$
$$\exists y \in \mathcal{I}(\mathbb{A}) \quad \text{t.c. } \mathbb{A}(y) = no$$

Allora $\mathbb{A} \in \mathbf{NPC}$

Dimostrazione. Se $\mathbf{NPC} \cap \mathbf{P} \neq \emptyset \quad \exists \mathbb{B}$ Np-hard t.c. $\mathbb{B} \in \mathbf{P} \wedge \forall \mathbb{C} \in \mathbf{NP} \;\; \mathbb{C} \leqslant_K \mathbb{B}$. Perciò deduciamo che $\mathbb{C} \in \mathbf{P}$, quindi ogni problema che è in **NP** è anche in **P** e viceversa. Quindi $\mathbf{P} \equiv \mathbf{NP}$.

Dobbiamo quindi dimostrare che ogni problema in **NP** si riduce polinomialmente ad $\mathbb{A}$
Prendiamo come esempio il seguente problema *bit*:

- Input: Bit b
- Output: $yes \Leftrightarrow b = 1$

Sia $\mathbb{D}$ un problema $\mathbb{D} \in \mathbf{NP}$ e quindi $\mathbb{D} \in \mathbf{P}$ (c'è un risolutore polinomiale per $\mathbb{D}$). Dobbiamo trovare una trasformazione $f(x)$ tale che riduce il problema $\mathbb{D}$ al problema *bit*:

$$f(x) = \begin{cases} 1 & \text{se } \mathbb{D}(x) = yes \\ 0 & \text{altrimenti} \end{cases}$$

dove $x \in \mathcal{I}(\mathbb{D})$.
Sappiamo quindi risolvere $f(x)$ in tempo polinomiale perché sappiamo risolvere $\mathbb{D}$ in tempo polinomiale poiché $\mathbb{D} \in \mathbf{NP} \wedge \mathbb{D} \in \mathbf{P}$. Quindi siano x e y

$$x_{yes} \in \mathcal{I}(\mathbb{A}) \quad \text{t.c. } \mathbb{A}(x_{yes}) = yes$$
$$x_{no} \in \mathcal{I}(\mathbb{A}) \quad \text{t.c. } \mathbb{A}(x_{no}) = no$$

allora la trasformazione $f(x)$ sarà:

$$f(x) = \begin{cases} x_{yes} & \text{se } \mathbb{D}(x) = yes \\ x_{no} & \text{se } \mathbb{D}(x) = no \end{cases}$$

□

7 Classe di problemi CO-NP

Definizione 7.0.1. (Classe **CO-NP**) L'insieme dei problemi **CO-NP** è definito nel seguente modo:

$$\mathbf{CO\text{-}NP} = \{\mathbb{A} \mid \overline{\mathbb{A}} \in \mathbf{NP}\}$$

Sono quei problemi per cui è "facile" verificare le istanze no.

Di seguito forniamo un paio di esempi di problemi:

Esempio 7.0.1. Problema:

- Input: Grafo G
- Output: yes se G non è colorabile con 7 colori.

Questo problema è il complemento del problema 7-COL. Quest'ultimo appartiene alla classe **NP** quindi il problema in esempio è in **CO-NP**.

Esempio 7.0.2. Problema:

- Input: formula booleana ϕ
- Output: yes se $\forall \underline{a}\ \phi(a) = T$

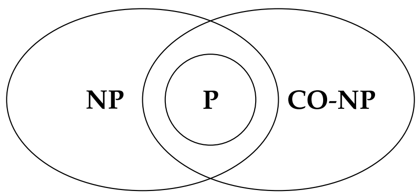
Per questo problema è facile vedere che esiste un'istanza no poiché basta che ci sia almeno una clausola con tutti i letterali a false. Quindi appartiene a **CO-NP**.

7.1 Relazione tra P,NP e CO-NP

Teorema 7.1.1. *Se* $\exists \mathbb{A}$ t.c. $\mathbb{A} \in \mathbf{NPC} \cap \mathbf{CO\text{-}NP}$ *allora* $\mathbf{NP} \equiv \mathbf{CO\text{-}NP}$.

$\mathbf{CO\text{-}NP} \subseteq \mathbf{NP}$. Supponiamo che $\mathbb{A} \in \mathbf{NPC}$ allora $\mathbb{A} \in \mathbf{NP}$ e $\forall \mathbb{C} \in \mathbf{NP} \quad \mathbb{C} \leqslant_K \mathbb{A}$.
Se prendiamo il problema $\mathbb{B} \in \mathbf{CO\text{-}NP} \quad \overline{\mathbb{B}} \in \mathbf{NP}$.
Allora esiste una riduzione alla Karp $\overline{\mathbb{B}} \leqslant_K \mathbb{A}$ che mappa le istanze yes di $\mathbb{B}$ alle istanze no di $\mathbb{A}$ ed esiste anche una riduzione $\mathbb{B} \leqslant_K \overline{\mathbb{A}}$ che è duale alla precedente.
Poiché $\mathbb{A} \in \mathbf{CO\text{-}NP}$ allora $\overline{\mathbb{A}} \in \mathbf{NP}$. Quindi $\mathbb{B}$ si riduce polinomialmente ad un problema in **NP**. Quindi $\mathbb{B} \in \mathbf{NP}$. Quindi per estensione $\mathbf{CO\text{-}NP} \subseteq \mathbf{NP}$. □

$\mathbf{NP} \subseteq \mathbf{CO\text{-}NP}$. Sia $\mathbb{C} \in \mathbf{NP} \quad \mathbb{C} \leqslant_K \mathbb{A} \quad \overline{\mathbb{C}} \leqslant_K \overline{\mathbb{A}}$. Poiché $\mathbb{A} \in \mathbf{CO\text{-}NP}$ allora $\overline{\mathbb{A}} \in \mathbf{NP}$. Quindi $\overline{\mathbb{C}} \in \mathbf{NP} \Rightarrow \mathbb{C} \in \mathbf{CO\text{-}NP} \Rightarrow \mathbf{NP} \subseteq \mathbf{CO\text{-}NP}$. □



Cosa succede se P ≡ CO-NP ? Se abbiamo l'equivalenza di queste due classi di problemi si ha che:

$$\begin{array}{ll} \mathbb{A}(x) \in \mathbf{NP} & \mathcal{A}(x) = \exists w\ B(x,w) \in \mathbf{P} \\ \mathbb{A}(x) \in \mathbf{CO\text{-}NP} & \mathcal{A}(x) = \forall w\ B(x,w) \in \mathbf{P} \end{array}$$

- Se $\mathbf{NP} \neq \mathbf{CO\text{-}NP} \Rightarrow \mathbf{P} \neq \mathbf{NP}$
- Se $\mathbf{P} = \mathbf{NP}$ siccome $\mathbf{P} = \mathbf{CO\text{-}NP} \quad \forall \mathbb{A} \in \mathbf{NP}, \mathbb{A} \in \mathbf{P} \Rightarrow \overline{\mathbb{A}} \in \mathbf{P} = \mathbf{NP} \quad \Rightarrow \mathbf{NP} = \mathbf{CO\text{-}NP}$

Definizione 7.1.1 (Hardness del problema $\mathbb{A}$ nella classe **CO-NP**)**.** $\mathbb{A}$ è **CO-NP-completo** se $\mathbb{A} \in$ **CO-NP** e $\forall \mathbb{B} \in$ **CO-NP** $\mathbb{B} \leqslant_K \mathbb{A}$.

Teorema 7.1.2. *Se $\mathbb{A}$ è* **NP-*completo*** *allora $\overline{\mathbb{A}}$ è* **CO-NP-*completo*** *e viceversa.*

Dimostrazione. Se $\mathbb{A}$ è **NP-completo**, allora

- $\mathbb{A} \in$ **NP**
- $\forall \mathbb{B} \in$ **NP** $\mathbb{B} \leqslant_K \mathbb{A}$

Dalla prima deduciamo che $\Rightarrow \quad \overline{\mathbb{A}} \in$ **CO-NP**
Dalla seconda invece, se $\mathbb{C} \in$ **CO-NP**, $\quad \overline{\mathbb{C}} \in$ **CO-NP** $\quad \Rightarrow \quad \overline{\mathbb{C}} \leqslant_K \mathbb{A} \quad \Rightarrow \mathbb{C} \leqslant_K \overline{\mathbb{A}}$
$\Rightarrow \forall \mathbb{C} \in$ **CO-NP** $\quad \Rightarrow \mathbb{C} \leqslant_K \overline{\mathbb{A}}$
Da queste due deduzioni abbiamo quindi la definizione di **CO-NP-completo** per $\overline{\mathbb{A}}$ □

1. Se vogliamo dimostrare che è **CO-NP-completo** possiamo dimostrare che *il complemento* è **NP-completo**.
2. Per dimostrare che $\mathbb{A}$ è **NP-completo**
   (a) $\mathbb{A} \in$ **NP**
   (b) $\forall \mathbb{B} \quad \mathbb{B} \leqslant_K \mathbb{A}$

7.2 Problema Minimo circuito booleano

- Input: Circuito booleano C_n (con n input)
- Output: $yes \Leftrightarrow \nexists$ circuito C' t.c. $\forall x\ C'(x) = C(x)$ con $|C'| < |C|$

Consideriamo l'algoritmo $\mathcal{A}$

$$\mathcal{A}(x) = \forall w_1 \exists w_2 \quad B(x, w_1, w_2) = yes \quad \text{con } B \in \mathbf{P} \text{ e } |w_i| = O(p_i(|x|))$$

Se minimo circuito booleano $\in$ **NP** allora: $\forall w_1 \exists w_2 \quad B(x, w_1, w_2) \equiv \exists w' \quad B'(x, w')$.
Se minimo circuito booleano $\in$ **CO-NP** allora: $\forall w'' \quad B''(x, w'')$.

8 Gerarchia Polinomiale

Definizione 8.0.1 (Classe di problemi $\Pi_i P$)**.**
$\Pi_i \mathbf{P} = \{\mathcal{A}(x) = \forall w_1 \exists w_2 \forall w_3 \exists w_4 \dots Q_i w_i \quad B(x, w_1, \dots, w_i) \quad \text{dove } |w_i| = O(p_i(|x|)) \text{ e } B \in \mathbf{P}\}$

Definizione 8.0.2 (Classe di problemi $\Sigma_i P$)**.**
$\Sigma_i \mathbf{P} = \{\mathcal{A}(x) = \exists w_1 \forall w_2 \exists w_3 \forall w_4 \dots Q_i w_i \quad B(x, w_1, \dots, w_i) \quad \text{dove } |w_i| = O(p_i(|x|)) \text{ e } B \in \mathbf{P}\}$

Dalla definizione di queste classi di problemi deduciamo che:

$$\Pi_0 \mathbf{P} = \Sigma_0 \mathbf{P} = \mathbf{P} \qquad \mathcal{A}(x) = \mathcal{B}(x) \text{ non ho quantificatori}$$
$$\Pi_1 \mathbf{P} = \mathbf{NP}$$
$$\Sigma_1 \mathbf{P} = \mathbf{CO\text{-}NP}$$
$$\text{Minimo circuito booleano } \in \Pi_2 \mathbf{P}$$

Osservazione 8.0.1. $\mathcal{A}(x) \in \Pi_i \mathbf{P} \quad \Leftrightarrow \quad \overline{\mathcal{A}(x)} \in \Sigma_i \mathbf{P}$.

Osservazione 8.0.2. $\Pi_i \mathbf{P} \subseteq \Sigma_{i+1} \mathbf{P}$ e $\Sigma_i \mathbf{P} \subseteq \Pi_{i+1} \mathbf{P}$.
Infatti se aggiungo un quantificatore all'inizio, ho che

$$\mathcal{A}(x) \in \Pi_i \mathbf{P}$$
$$\mathcal{A}(x) = \forall w_1 \exists w_2 \dots Q_i w_i \quad B(x, w_1, w_2, \dots, w_i)$$
$$\Sigma_{i+1} \mathbf{P} = \exists w^* \forall w_1 \exists w_2 \dots Q_i w_i \quad B'(x, w^*, w_1, w_2, \dots, w_i)$$

Perciò $B'(\dots) = B(\dots)$

Osservazione 8.0.3. Per lo stesso motivo dell'osservazione precedente vale che:
$\Pi_i\mathbf{P} \subseteq \Pi_{i+1}\mathbf{P}$ e $\Sigma_i\mathbf{P} \subseteq \Sigma_{i+1}\mathbf{P}$.

Osservazione 8.0.4. Se $\mathbf{P} \equiv \mathbf{NP} \quad \Rightarrow \quad \forall i\ \Sigma_i\mathbf{P} = \mathbf{P} \wedge \Pi_i\mathbf{P} = \mathbf{P}$
cioè abbiamo che:

$$B(x, w_1, w_2, \ldots, w_i) = B'(x) \text{ (elimino tutte le quantificazioni)}$$

Proposizione 8.0.1. Se $\mathbf{NP} = \mathbf{CO\text{-}NP} \quad \Rightarrow \quad \Sigma_1\mathbf{P} = \Pi_1\mathbf{P}$.
Quindi $\Sigma_i\mathbf{P} = \Pi_i\mathbf{P} = \Sigma_1\mathbf{P} = \Pi_1\mathbf{P} \quad \forall i \geqslant 1$.
Tutte le classi sopra collassano sulla classe 1.

Dimostrazione. Assumiamo che $\mathbf{NP} \equiv \mathbf{CO\text{-}NP}$:
$\mathcal{A}(x) = \exists w_1 \quad B(x, w_1) \Leftrightarrow \mathcal{A}(x) = \forall w_1 \quad B'(x, w_1)$
Sia $\mathcal{A}'(x) \in \Sigma_1\mathbf{P} \qquad \mathcal{A}'(x) = \exists w_1 \forall w_2 \quad C(x, w_1, w_2) = \mathcal{D}_{w_2}(x)$.
$\mathcal{D}_{w_2}(x) \in \mathbf{CO\text{-}NP} \equiv \mathbf{NP}$ quindi $\mathcal{D}_{w_2}(x) = \exists w_1' \quad C'(x, w_1', w_2)$ perciò diventa:

$$\begin{aligned} \mathbb{A}'(x) &= \exists w_2 \exists w_1'' \quad C'(x, w_1'', w_2) \\ &= \exists w_{12} \quad C'(x, w_{12}) \quad \in \mathbf{NP} \end{aligned}$$

Quindi deduciamo che se $\mathbf{NP} \equiv \mathbf{CO\text{-}NP} \quad \Rightarrow \quad \Sigma_2\mathbf{P} = \Sigma_1\mathbf{P}$ □

Definizione 8.0.3 (Gerarchia Polinomiale)**.** Definiamo gerarchia polinomiale la classe **PH** delle proprietà $\mathbb{A}$ che possono essere espresse da una formula con quantificatori contenente un numero costante di quantificatori alternati:

$$\mathbf{PH} = \bigcup_k \Sigma_k\mathbf{P} = \bigcup_k \Pi_k\mathbf{P}$$

Teorema 8.0.1 (Collasso della gerarchia polinomiale)**.** *Se*

$$\mathbf{P} = \mathbf{NP} \quad \Rightarrow \quad \mathbf{NP} = \mathbf{CO\text{-}NP} = \mathbf{P} \quad \Rightarrow \quad \Sigma_i\mathbf{P} = \Pi_i\mathbf{P} = \mathbf{P} \quad \forall i$$

la gerarchia polinomiale collassa in **P***.*
Se $\mathbf{NP} = \mathbf{CO\text{-}NP} \quad \Rightarrow \quad \mathbf{PH} = \mathbf{NP} = \mathbf{CO\text{-}NP}$.

Teorema 8.0.2. *Se* $\Pi_i\mathbf{P} = \Sigma_i\mathbf{P} \quad \Rightarrow \quad \mathbf{PH} = \Pi_i\mathbf{P} = \Sigma_i\mathbf{P}$

8.1 Funzione time-costruibile

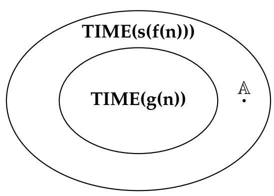
Proposizione 8.1.1. Nel modello computazionale in oggetto è possibile simulare t passi di un algoritmo (programma) mentre controlliamo che $\leqslant t$ passi sono fatti in $s(t)$ passi.

Esempio 8.1.1. Se il modello computazionale è la Macchina di Turing, allora $s(t) = O(t \log t)$.

Esempio 8.1.2. Se il modello computazionale è la RAM, allora $s(t) = O(t)$

Definizione 8.1.1. Diciamo che $f(n)$ è **Time-costruibile** se esiste un programma (algoritmo) che calcola $f(n)$ in $O(f(n))$.

Teorema 8.1.1. *Data l'assunzione precedente, per ogni funzione* $f(n)$ *time-costruibile e per ogni* $g(n) = o(f(n))$ *la classe* ***TIME(g(n))*** $\subset$ ***TIME(s(f(n)))***



8.2 Problema Catch 22

- Input: Π (programma)
- Output: se $\Pi(\Pi)$ termina in meno di $f(|\Pi|)$ passi allora ritorna $\overline{\Pi(\Pi)}$ altrimenti ritorna 0.

Supponiamo che esista un algoritmo Π_{22} tale che risolve il problema Catch 22 in $g(n)$ passi, dove $g(n) < f(n)$. Questo è equivalente a dire che Catch 22 $\in$ **TIME(g(n))**.
Se $\Pi_{22}(\Pi_{22}) =$ Catch 22(Π_{22}) siccome ci mette meno di $f(\Pi_{22})$ passi, allora e uguale a $\overline{\Pi_{22}(\Pi_{22})}$. Questo è assurdo perché non può essere che $\Pi_{22}(\Pi_{22}) = \overline{\Pi_{22}(\Pi_{22})}$, quindi *non* esiste l'algoritmo Π_{22} che impiega $g(n) < f(n)$ passi.

Supponiamo che il programma Π risolve Catch 22 se e solo se $\forall x \in \mathcal{I}$(Catch 22) $\Pi(x) =$ Catch 22(x). Se Π termina in $\leqslant f(n)$ passi per ogni x, allora $\exists x$ t.c. $\Pi(x) \neq$ Catch 22(x).

Proposizione 8.2.1. Per ogni algoritmo esistono infiniti programmi Π che implementano l'algoritmo (fanno la stessa cosa) di lunghezza arbitrariamente grandi.

Proposizione 8.2.2. Per ogni $n \geqslant |\Pi_{22}|$ fissato esiste un altro Π'_{22} tale che $|\Pi'_{22}| = n$. Quindi $\Pi'_{22}(\Pi'_{22}) = \Pi_{22}(\Pi_{22})$.

9 Teorema di Ladner

Ci chiediamo se esiste un problema **NP** che non appartiene nè alla classe **P** nè alla classe **NPC**.

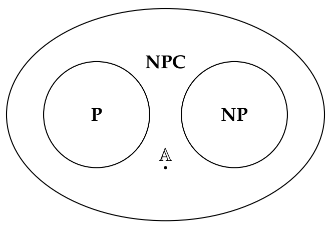


Figura 3: Esiste il problema $\mathbb{A}$?

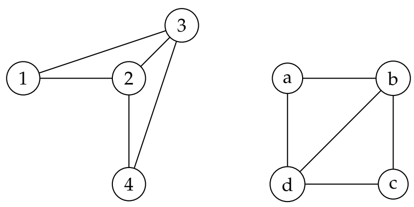
Teorema 9.0.1 (Teorema di Ladner)**.** *Se* **P** $\neq$ **NP** *allora esiste un problema* $\mathbb{A} \in$ **NP** $\setminus$ (**P** $\cup$ **NPC**).

Dimostrazione. Vediamo un problema esempio che soddisfa il *teorema di Ladner*:
Graph Isomorphism

- Input: G_1, G_2 grafi
- Output: $yes \Leftrightarrow G_1$ è *isomorfo* a G_2.

Definizione 9.0.1 (Isomorfismo)**.** $\exists f: \quad V(G_1) \mapsto V(G_2)$
t.c. $(v, u) \in E(G_1) \Leftrightarrow (f(v), f(u)) \in E(G_2)$

Esempio 9.0.1 (Grafi isomorfi)**.** Ecco un esempio di due grafi isomorfi:



$$f(1) = a \quad f(2) = b$$
$$f(3) = c \quad f(4) = d$$

$$\mathbb{A}(x) = \begin{cases} \mathrm{SAT}(x) & \text{se } f(|x|) \text{ è pari} \\ 0 & \text{se } f(|x|) \text{ è dispari} \end{cases}$$

Vogliamo far vedere che:

1. $\mathbb{A} \in$ **NP**
2. $\mathbb{A} \notin$ **P**, cioè $\forall \Pi$ polinomiale $\exists x$ t.c. $\Pi(x) \neq \mathbb{A}(x)$.
3. $\mathbb{A} \notin$ **NPC**, cioè $\forall \Pi$ polinomiale $\exists x$ t.c. $\mathrm{SAT}(x) \neq \mathbb{A}(\Pi(x))$.
   Se SAT è **NPC** sappiamo che $\mathrm{SAT} \leqslant_K \mathbb{A}$

□

9.1 Problema Clique

- grafo $G = (V, E), K$
- $yes \Leftrightarrow G$ contiene una clique di taglia K

Clique è un insieme di vertici tutti connessi a due a due da un arco.

Clique $\in$ **NPC** Facciamo vedere che il problema Clique appartiene alla classe **NPC** e che quindi appartiene alla classe **NP** e che esiste la riduzione 3-SAT $\leqslant_K$ Clique che trasforma in tempo polinomiale una formula ϕ CNF in un grafo per il problema Clique.

Clique $\in$ **NP** Creiamo un verificatore per il problema Clique:

- Conta i vertici del grafo C. $[O(n)]$
- Per ogni $(u, v) \in C$ verifica che $(u, v) \in E$. $\left[O(|K|^2 \times |E|)\right]$

Questo verificatore è polinomiale.
Il certificato per il verificatore è una clique C di taglia K in G, tale clique ha taglia polinomiale perché K può essere al massimo n. Perciò Clique $\in$ **NP**.

3-SAT $\leqslant_K$ **Clique** Vediamo la seguente riduzione che mappa la formula

$$\phi = (x_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3)$$

in un grafo che soddisfa il problema Clique.

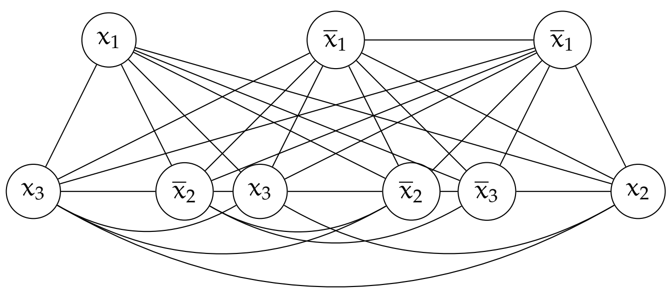


Figura 4: Grafo in cui c'è un arco per ogni letterale diverso dal proprio negato

Il grafo mostra le seguenti caratteristiche:

- Numero di vertici: $|V| = 3m$ con $\phi = C^{(1)} \wedge \cdots \wedge C^{(m)}$.

- Numero di archi: $|E| \leqslant 9m^2$

Quindi il grafo, e di conseguenza la riduzione, è costruibile in tempo polinomiale.

Dimostriamo ora che se ϕ è soddisfacibile allora esiste un assegnamento $a_1, a_2, \dots, a_n$ per $x_1, \dots, x_n$ tale che in ogni clausola un letterale è posto a T.
Siano $v_{i1}^{(1)} v_{i2}^{(2)} \cdots v_{in}^{(n)}$ i vertici corrispondenti ai letterali posti a T dell'assegnamento (uno per clausola). Tali vertici rappresentano nel grafo una clique.

Dimostriamo ora che se G ha una clique di taglia m allora ϕ è soddisfacibile. Supponiamo che G abbia una clique C è taglia m.

1. Gli m vertici di C sono uno per tripla. Le triple corrispondono alle clausole.
2. Due vertici in C non corrispondono a letterali opposti di ϕ.

Dall'ultimo punto in questione costruiamo un assegnamento che soddisfa ϕ. Se prendiamo i vertici di C e li assegniamo a T, gli altri vengono assegnati di conseguenza:

$$\begin{aligned} \overline{x}_2 &= T \quad x_2 = F \\ \overline{x}_3 &= T \quad x_3 = F \\ x_1 &= F \end{aligned}$$

Perciò abbiamo che

$$\phi(F, F, F) = (\underset{F}{x_1} \vee \underset{T}{\overline{x}_2} \vee \underset{F}{x_3}) \wedge (\underset{T}{\overline{x}_1} \vee \underset{F}{x_2} \vee \underset{F}{x_3}) \wedge (\underset{T}{\overline{x}_1} \vee \underset{F}{x_2} \vee \underset{F}{x_3}) = T$$

9.2 Problema Independent Set