

Automi e Linguaggi Formali

Parte 18 – NP Completezza

Davide Bresolin

Ultimo aggiornamento: 26 maggio 2022



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Sommario

- 1** Tesi di Church computazionale
- 2** Riduzioni polinomiali
- 3** Il Re dei problemi NP
- 4** Problemi su grafi
- 5** Conclusioni

Tesi di Church computazionale



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Tutti i formalismi di calcolo **ragionevoli** sono equivalenti a meno di fattori **polinomiali** nei tempi di calcolo.

Tutti i formalismi di calcolo **ragionevoli** sono equivalenti a meno di fattori **polinomiali** nei tempi di calcolo.

Esempi:

- Macchine di Turing Deterministiche
- Linguaggi di programmazione concreti: Java, C++, Python,
....

Tutti i formalismi di calcolo **ragionevoli** sono equivalenti a meno di fattori **polinomiali** nei tempi di calcolo.

Esempi:

- Macchine di Turing Deterministiche
- Linguaggi di programmazione concreti: Java, C++, Python,

....

Eccezioni:

- Computer quantistici
- DNA Computing, Bio Computing

Verificatori

Definition

Un **verificatore** per un linguaggio A è un algoritmo V tale che

$$A = \{w \mid V \text{ accetta } \langle w, c \rangle \text{ per qualche stringa } c\}$$

- il verificatore usa **ulteriori informazioni** per stabilire se w appartiene al linguaggio
- questa informazione è il **certificato** c
- un **verificatore polinomiale** opera in tempo $O(|w|)^k$ per qualche costante k

Due problemi in P . . .

Raggiungibilità in un grafo

$PATH = \{\langle G, s, t \rangle \mid G \text{ grafo che contiene un cammino da } s \text{ a } t\}$

Numeri relativamente primi

$RELPRIME = \{\langle x, y \rangle \mid 1 \text{ è il massimo comun divisore di } x \text{ e } y\}$

... e due problemi in NP

Problema del circuito Hamiltoniano

$HAMILTON = \{\langle G \rangle \mid G \text{ è un grafo con un circuito Hamiltoniano}\}$

Numeri composti

$COMPOSITES = \{\langle x \rangle \mid x = pq, \text{ per gli interi } p, q > 1\}$

Un verificatore per *COMPOSITES*



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

La seguente TM è un **verificatore** per *COMPOSITES*. Il certificato è uno dei due divisori di x

V = “su input $\langle x, p \rangle$, con x, p numeri interi:

- 1** se $p \leq 1$ o $p = x$, **rifiuta**
- 2** se p è un divisore di x , **accetta**, altrimenti **rifiuta**.”

Problemi P ed NP

- **P** è la classe dei linguaggi in cui l'appartenenza di una stringa $x \in \Sigma^*$ al linguaggio può essere **decisa** da una macchina di Turing deterministica in tempo $O(|x|^k)$
- **NP** è la classe dei linguaggi in cui l'appartenenza di una stringa $x \in \Sigma^*$ al linguaggio può essere **verificata** da un verificatore in tempo $O(|x|^k)$.
- **Equivalente:** è la classe dei linguaggi in cui l'appartenenza di una stringa $x \in \Sigma^*$ al linguaggio può essere decisa da una macchina di Turing **nondeterministica** in tempo $O(|x|^k)$.
- **coNP** è la classe dei linguaggi tali che il loro complementare è in **NP**

Sommario



1 Tesi di Church computazionale

2 Riduzioni polinomiali

3 Il Re dei problemi NP

4 Problemi su grafi

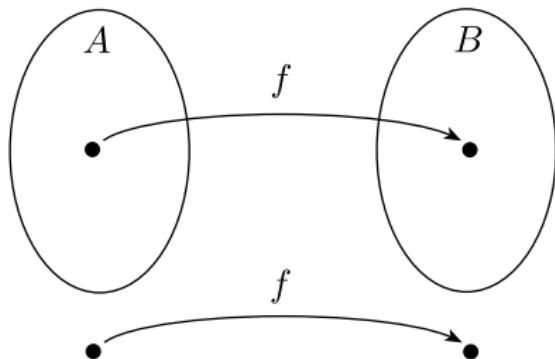
5 Conclusioni

Riducibilità in tempo polinomiale

Definition

Un linguaggio A è **riducibile in tempo polinomiale** al linguaggio B ($A \leq_P B$), se esiste una **funzione calcolabile in tempo polinomiale** $f : \Sigma^* \mapsto \Sigma^*$ tale che

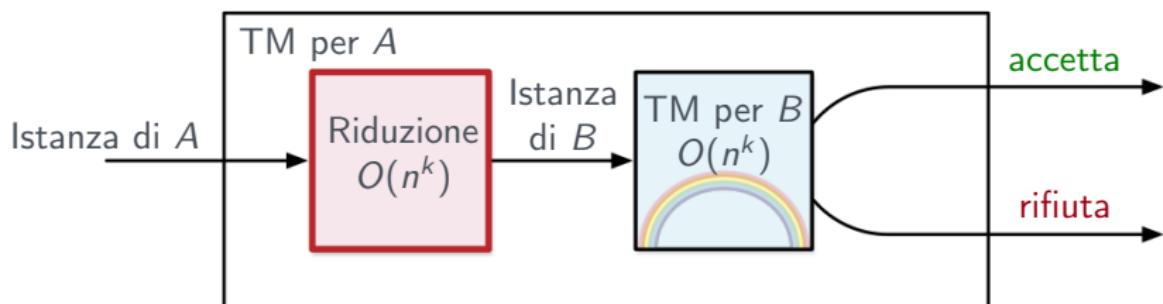
per ogni $w \in \Sigma^* : w \in A$ se e solo se $f(w) \in B$



f è la **riduzione**
polinomiale da A a B

Riduzioni polinomiali

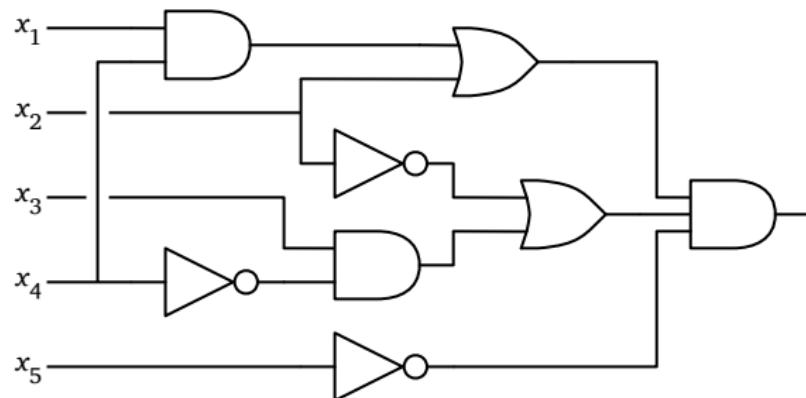
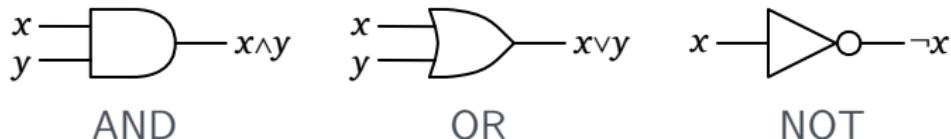
Se $A \leq_P B$, e $B \in P$, allora $A \in P$:



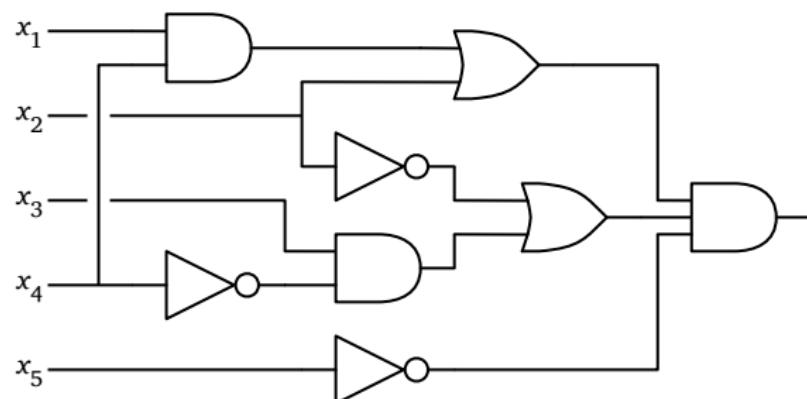
Sommario

- 1** Tesi di Church computazionale
- 2** Riduzioni polinomiali
- 3** Il Re dei problemi NP
- 4** Problemi su grafi
- 5** Conclusioni

Un problema NP



Un problema NP



CircuitSAT: dato un circuito Booleano, esistono dei **valori di input** che permettono di ottenere **output = 1**?

CircuitSAT è un problema NP

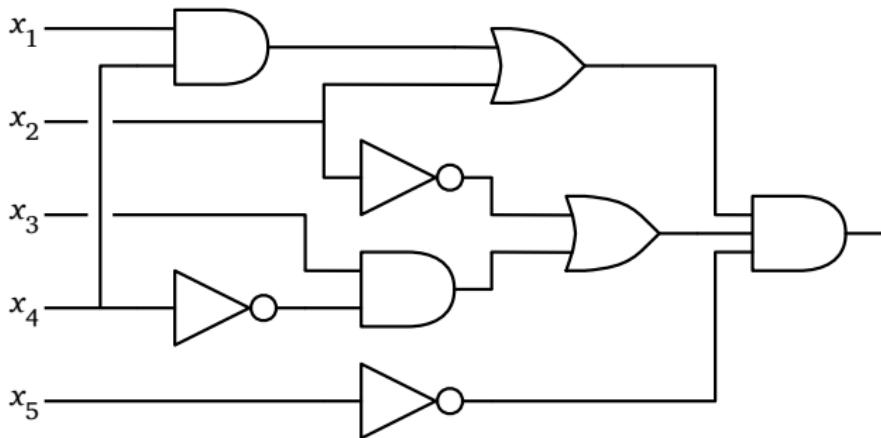
- 1 Trovare un **certificato** di appartenenza:
⇒ i **valori** degli input x_1, x_2, \dots

CircuitSAT è un problema NP

- 1 Trovare un **certificato** di appartenenza:
⇒ i **valori** degli input x_1, x_2, \dots
- 2 Trovare un **algoritmo polinomiale** per verificare il certificato.

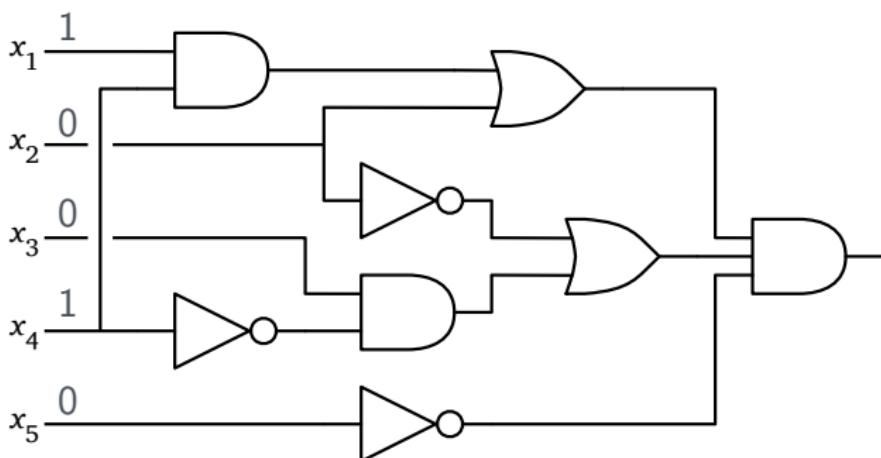
Verifica del certificato

Certificato: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0$



Verifica del certificato

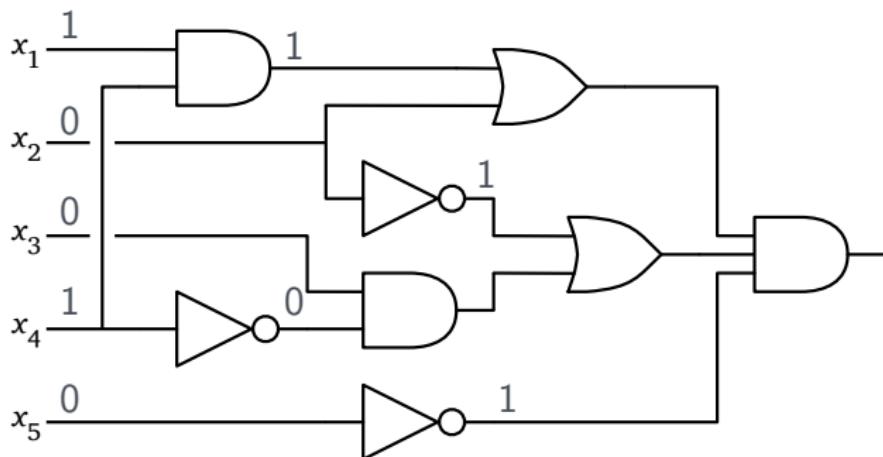
Certificato: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0$



- 1 Assegna gli input come stabilito nel certificato

Verifica del certificato

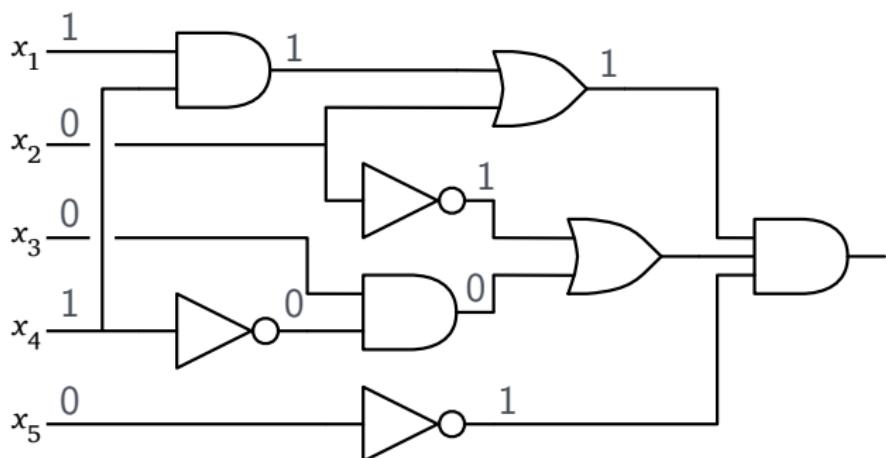
Certificato: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0$



- 2 Calcola gli output delle porte logiche di primo livello

Verifica del certificato

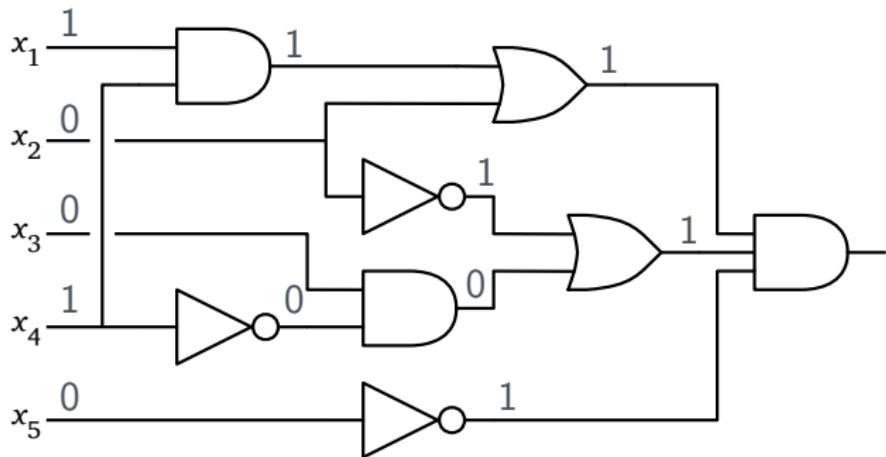
Certificato: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0$



- 3 Calcola gli output delle porte logiche di secondo livello

Verifica del certificato

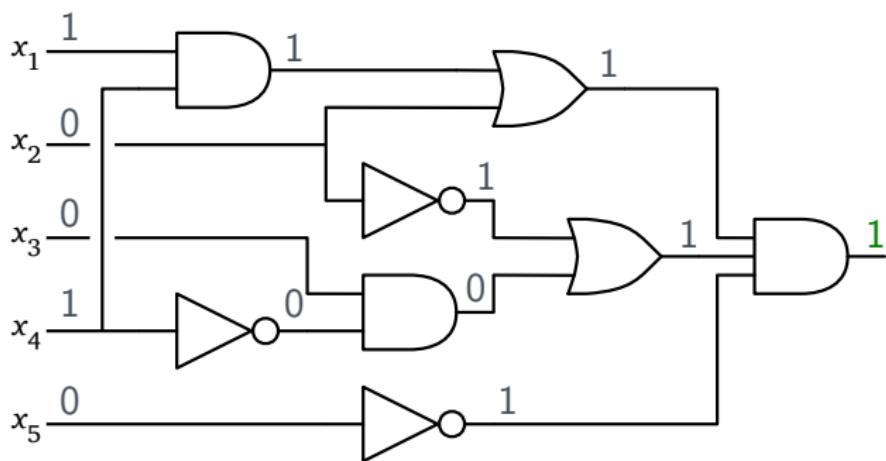
Certificato: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0$



- #### 4 Calcola gli output delle porte logiche di terzo livello

Verifica del certificato

Certificato: $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0$

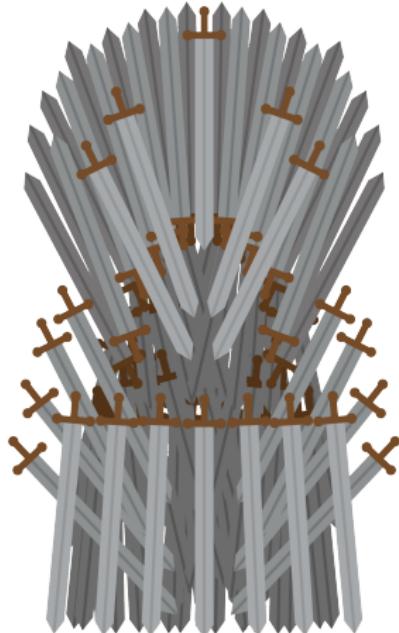


- 5 Calcola l'output dell'intero circuito: se = 1, SI, altrimenti NO

Il Re dei problemi NP



UNIVERSITÀ
DEGLI STUDI
DI PADOVA



CircuitSAT è Re tra i problemi in NP:

- Ogni problema che sta in NP può essere **trasformato** in una **istanza** di **CircuitSAT** in **tempo polinomiale**
- **CircuitSAT** può essere usato per **risolvere** tutti i problemi che stanno in NP
- chi scopre un algoritmo polinomiale per **CircuitSAT** sa risolvere **tutti i problemi NP** in **tempo polinomiale!**

Teorema di Cook-Levin

Theorem (Cook e Levin, 1973)

*L'esistenza di una **macchina di Turing polinomiale** per risolvere **CircuitSAT** implica che $P = NP$.*

Teorema di Cook-Levin

Theorem (Cook e Levin, 1973)

*L'esistenza di una **macchina di Turing polinomiale** per risolvere **CircuitSAT** implica che $P = NP$.*

P contro NP è uno dei **problemi del millennio** del Clay Institute:



ABOUT PROGRAMS MILLENNIUM PROBLEMS PEOPLE PUBLICATIONS EVENTS EUCLID

P vs NP Problem



Suppose that you are organizing housing accommodations for a group of four hundred university students. Space is limited and can only accommodate two hundred students in the dormitory. To complicate matters the Dean has provided you with a list of pairs of incompatible students, and requested that no pair from this list appear in your final choice. This is an example of what computer scientists call an NP-problem, since it is easy to check if a given choice of one hundred students proposed by a coworker is satisfactory (i.e., no pair taken from your coworker's list also appears on the list from the Dean's office), however the task of generating such a list from scratch seems to be so hard as to be completely impractical. Indeed, the total number of ways of choosing one hundred students from the four hundred applicants is greater than the number of atoms in the known universe! Thus no future civilization could ever hope to build a supercomputer capable of solving the problem! See the French site for further details.

Rules
Rules for the Millennium Prizes

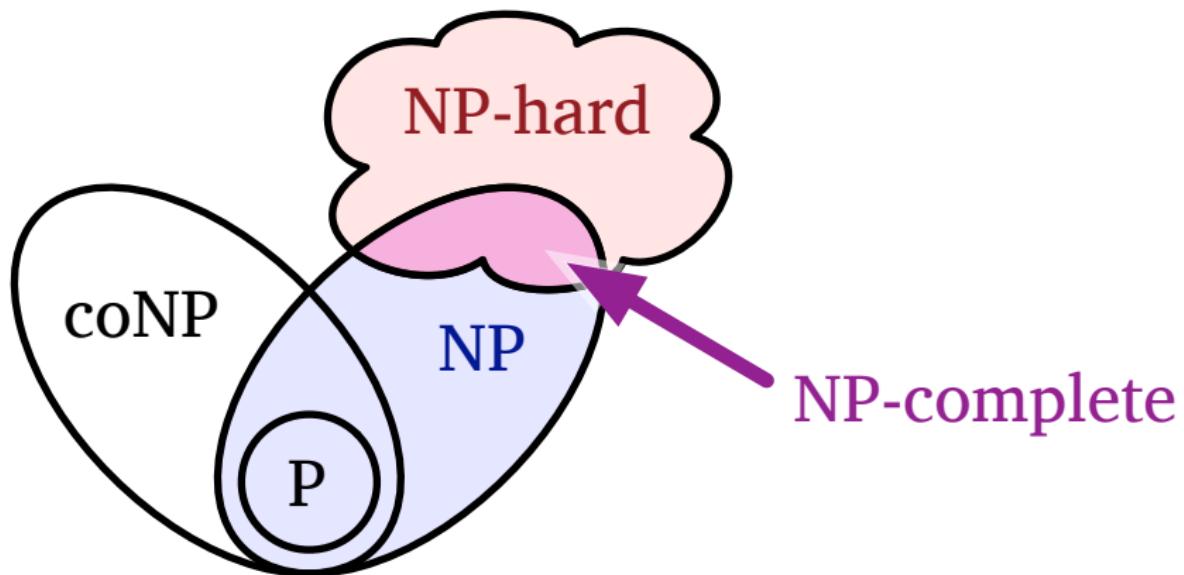
Related Documents:

- Official Problem Description
- Minesweeper

1.000.000 US\$ di taglia
per chi lo risolve!

Problemi NP-hard

- Un problema è **NP-hard** se l'esistenza di un algoritmo polinomiale per risolverlo implica l'esistenza di un algoritmo polinomiale **per ogni problema in NP**.
- Se siamo in grado di risolvere un problema **NP-hard** in modo efficiente, allora possiamo risolvere in modo efficiente **ogni problema** di cui possiamo verificare facilmente una soluzione, usando la soluzione del problema **NP-hard** come sottoprocedura.
- Un problema è **NP-completo** se è sia **NP-hard** che appartenente alla classe **NP** (o “**NP-easy**”).
 - Esempio: **CircuitSAT!**



Perché studiare la NP-completezza?

- Progettare ed implementare algoritmi richiede la conoscenza dei principi di base della teoria della complessità
- Stabilire che un problema è NP-completo costituisce una prova piuttosto forte della sua intrattabilità
- Conviene cercare di risolverlo con un approccio diverso . . .
 - identificare un caso particolare trattabile
 - cercare una soluzione approssimata
- . . . invece di cercare un algoritmo efficiente per il caso generale che probabilmente non esiste nemmeno

Dimostrare che un problema è NP-completo

Ogni dimostrazione di **NP-completezza** di compone di due parti:

- 1 dimostrare che il problema appartiene alla classe **NP**;
- 2 dimostrare che il problema è **NP-hard**.

Dimostrare che un problema è NP-completo

Ogni dimostrazione di **NP-completezza** di compone di due parti:

- 1 dimostrare che il problema appartiene alla classe **NP**;
 - 2 dimostrare che il problema è **NP-hard**.
- Dimostrare che un problema è in **NP** vuol dire dimostrare l'esistenza di un **verificatore polinomiale**.

Dimostrare che un problema è NP-completo

Ogni dimostrazione di **NP-completezza** di compone di due parti:

- 1 dimostrare che il problema appartiene alla classe **NP**;
 - 2 dimostrare che il problema è **NP-hard**.
- Dimostrare che un problema è in **NP** vuol dire dimostrare l'esistenza di un **verificatore polinomiale**.
 - Le tecniche che si usano per dimostrare che un problema è **NP-hard** sono fondamentalmente diverse.

Riduzioni tra problemi diversi

- Per dimostrare che un certo problema è **NP-hard** si procede tipicamente con una **dimostrazione per riduzione polinomiale**

Riduzioni tra problemi diversi

- Per dimostrare che un certo problema è **NP-hard** si procede tipicamente con una **dimostrazione per riduzione polinomiale**

Per dimostrare che **B** è **NP-hard** dobbiamo ridurre un problema **NP-hard** a **B**.

Riduzioni tra problemi diversi

- Per dimostrare che un certo problema è **NP-hard** si procede tipicamente con una **dimostrazione per riduzione polinomiale**

Per dimostrare che **B** è **NP-hard** dobbiamo ridurre un problema **NP-hard** a **B**.

- Abbiamo bisogno di un problema **NP-hard** da cui partire: **CircuitSAT**

Schema di riduzione polinomiale

Per dimostrare che un problema B è **NP-hard**:

- 1 Scegli un problema A che **sai essere NP-hard**.
- 2 Descrivi una **riduzione polinomiale** da A a B :
 - data un'istanza di A , **trasformala** in un'istanza di B ,
 - con una funzione che opera in **tempo polinomiale**.
- 3 Dimostra che la riduzione è **corretta**:
 - Dimostra che la funzione trasforma **istanze “buone”** di A in **istanze “buone”** di B .
 - Dimostra che la funzione trasforma **istanze “cattive”** di A in **istanze “cattive”** di B .
Equivalente: se la tua funzione produce un'istanza “buona” di B , allora era partita da un'istanza “buona” di A .
- 4 Mostra che la funzione impiega **tempo polinomiale**.

Il problema della soddisfacibilità booleana



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

- una formula Booleana come

$$(a \vee b \vee c \vee \overline{d}) \leftrightarrow ((b \wedge \overline{c}) \vee \overline{(\overline{a} \rightarrow d)} \vee (c \neq a \wedge b)),$$

- è **soddisfacibile** se è possibile assegnare dei valori booleani (Vero/Falso) alle variabili a, b, c, \dots , in modo che il valore di verità della formula sia Vero

SAT = { $\langle \varphi \rangle \mid \varphi$ è una formula booleana soddisfacibile}



SAT è NP-completo

- SAT è in NP:

SAT è NP-completo

■ **SAT** è in NP:

il **certificato** è l'assegnamento di verità alle variabili a, b, c, \dots

SAT è NP-completo

- **SAT** è in NP:
il certificato è l'assegnamento di verità alle variabili a, b, c, \dots
- **SAT** è NP-hard:

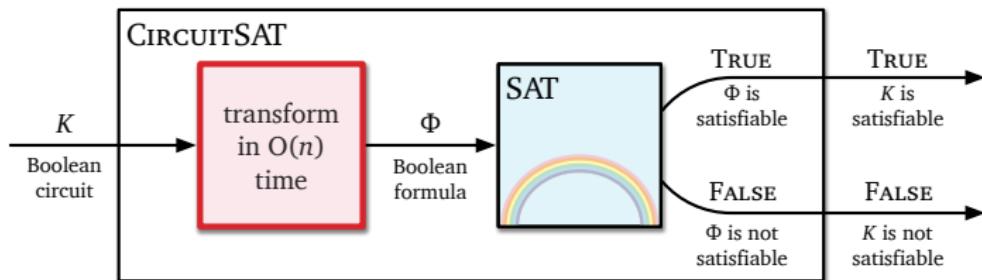
SAT è NP-completo

- **SAT è in NP:**

il **certificato** è l'assegnamento di verità alle variabili a, b, c, \dots

- **SAT è NP-hard:**

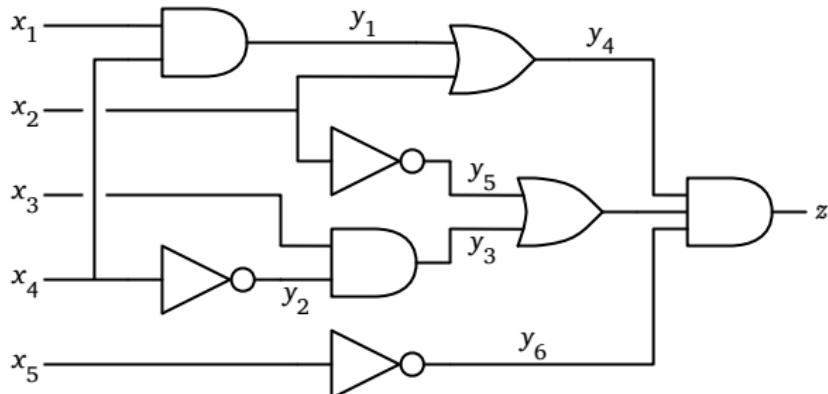
dimostrazione per **riduzione** di **CircuitSAT** a **SAT**



Riduzione di CircuitSAT a SAT

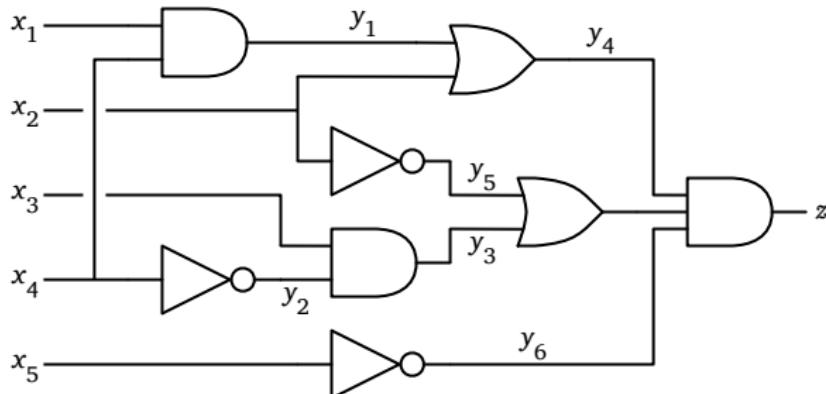


1 Dare un nome agli **output** delle porte logiche:



Riduzione di CircuitSAT a SAT

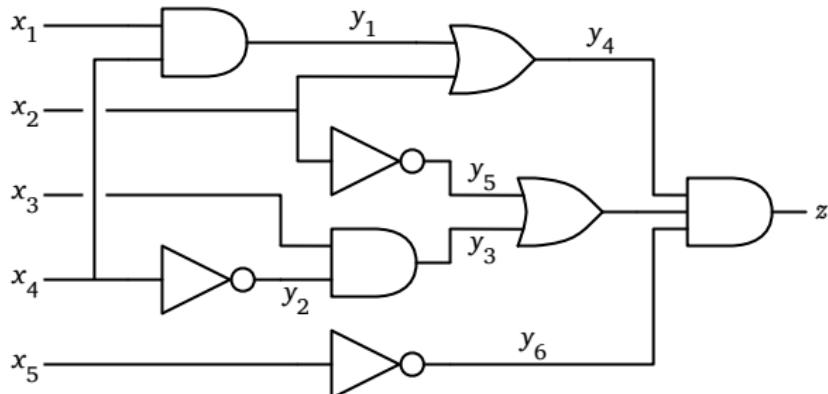
- 1 Dare un nome agli **output** delle porte logiche:



- 2 Scrivere le **espressioni booleane** per ogni porta logica e metterle in **and logico**, aggiungendo “**AND z**” alla fine:

Riduzione di CircuitSAT a SAT

- 1 Dare un nome agli **output** delle porte logiche:



- 2 Scrivere le **espressioni booleane** per ogni porta logica e metterle in **and** logico, aggiungendo “**AND z**” alla fine:

$$(y_1 = x_1 \wedge x_4) \wedge (y_2 = \overline{x_4}) \wedge (y_3 = x_3 \wedge y_2) \wedge (y_4 = y_1 \vee x_2) \wedge \\ (y_5 = \overline{x_2}) \wedge (y_6 = \overline{x_5}) \wedge (y_7 = y_3 \vee y_5) \wedge (z = y_4 \wedge y_7 \wedge y_6) \wedge z$$

Correttezza della riduzione

Ora dobbiamo mostrare che il circuito originale K è soddisfacibile
se e solo se la formula risultante Φ è soddisfacibile.



Correttezza della riduzione

Ora dobbiamo mostrare che il circuito originale K è soddisfacibile
se e solo se la formula risultante Φ è soddisfacibile.
Dimostriamo questa affermazione **in due passaggi**:

Correttezza della riduzione

Ora dobbiamo mostrare che il circuito originale K è soddisfacibile **se e solo se** la formula risultante Φ è soddisfacibile.

Dimostriamo questa affermazione **in due passaggi**:

- ⇒ Dato un insieme di input che rende vero il circuito K , possiamo ottenere i valori di verità per le variabili nella formula Φ calcolando l'output di ogni porta logica di K .

Correttezza della riduzione

Ora dobbiamo mostrare che il circuito originale K è soddisfacibile **se e solo se** la formula risultante Φ è soddisfacibile.
Dimostriamo questa affermazione **in due passaggi**:

- ⇒ Dato un insieme di input che rende vero il circuito K , possiamo ottenere i valori di verità per le variabili nella formula Φ calcolando l'output di ogni porta logica di K .
- ⇐ Dati i valori di verità delle variabili nella formula Φ , possiamo ottenere gli input del circuito semplicemente ignorando le variabili delle porte logiche interne y_i e la variabile di uscita z .

Correttezza della riduzione

Ora dobbiamo mostrare che il circuito originale K è soddisfacibile **se e solo se** la formula risultante Φ è soddisfacibile.

Dimostriamo questa affermazione **in due passaggi**:

- ⇒ Dato un insieme di input che rende vero il circuito K , possiamo ottenere i valori di verità per le variabili nella formula Φ calcolando l'output di ogni porta logica di K .
- ⇐ Dati i valori di verità delle variabili nella formula Φ , possiamo ottenere gli input del circuito semplicemente ignorando le variabili delle porte logiche interne y_i e la variabile di uscita z .

L'intera trasformazione da circuito a formula può essere eseguita **in tempo lineare**. Inoltre, la dimensione della formula risultante cresce di **un fattore costante** rispetto a qualsiasi ragionevole rappresentazione del circuito.

Una versione ristretta di soddisfacibilità

- Intendiamo dimostrare l'NP-completezza di un'**ampia gamma di problemi**
- Dovremmo procedere per **riduzione polinomiale** da **SAT** al problema in esame
- Esiste però un importante **problema “intermedio”**, detto **3SAT**, molto più **facile da ridurre** ai problemi tipici rispetto a **SAT**:
 - anche **3SAT** è un problema di **soddisfacibilità di formule booleane**
 - **3SAT** però richiede che le formule siano di una **forma ben precisa**, formate cioè da congiunzione logica di **clausole** ognuna delle quali è disgiunzione logica di **tre variabili** (anche negate)

Forme normali di formule booleane

- Un **letterale** è una variabile o una variabile negata, ad es. x , \bar{y}
- Una **clausola** è una disgiunzione logica (OR) di uno o più letterali, ad es. x , $x \vee \bar{y}$
- Una formula booleana si dice in **forma normale congiuntiva (CNF)**, se è la congiunzione logica (AND) di una o più clausole:
 - $(x \vee y) \wedge (\bar{x} \vee z)$, e $x \wedge y$ sono in CNF
 - mentre $(x \vee y \vee z) \wedge (\bar{y} \vee \bar{z}) \vee (x \vee y \wedge z)$ non è in CNF
- Una formula si dice in **forma normale 3-congiuntiva (3-CNF)** se è composta di clausole che hanno **esattamente** 3 letterali distinti

Il problema 3SAT

3SAT = { $\langle\varphi\rangle \mid \varphi$ è una formula booleana in 3-CNF soddisfacibile}

Il problema 3SAT

3SAT = { $\langle\varphi\rangle \mid \varphi$ è una formula booleana in 3-CNF soddisfacibile}

- **3SAT** è in NP:

Il problema 3SAT

3SAT = { $\langle \varphi \rangle \mid \varphi$ è una formula booleana in 3-CNF soddisfacibile}

- **3SAT** è in NP:
il **certificato** è l'assegnamento di verità alle variabili a, b, c, \dots

Il problema 3SAT

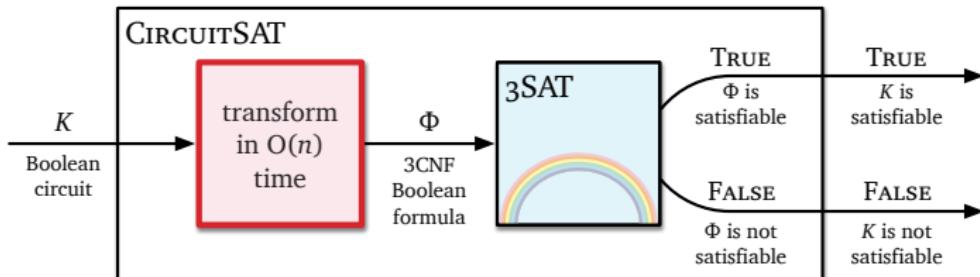
3SAT = { $\langle \varphi \rangle \mid \varphi$ è una formula booleana in 3-CNF soddisfacibile}

- **3SAT** è in NP:
il certificato è l'assegnamento di verità alle variabili a, b, c, \dots
- **3SAT** è NP-hard:

Il problema 3SAT

3SAT = { $\langle \varphi \rangle \mid \varphi$ è una formula booleana in 3-CNF soddisfacibile}

- **3SAT** è in NP:
il certificato è l'assegnamento di verità alle variabili a, b, c, \dots
- **3SAT** è NP-hard:
dimostrazione per **riduzione** di **CircuitSAT** a **3SAT**



Riduzione di CircuitSAT a 3SAT

- 1 **Fai in modo che ogni porta logica abbia al massimo due input**
- 2 **Trasforma il circuito in una formula booleana come fatto per SAT**
- 3 **Trasforma la formula in CNF usando le regole seguenti:**

$$a = b \wedge c \mapsto (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee b) \wedge (\bar{a} \vee c)$$

$$a = b \vee c \mapsto (\bar{a} \vee b \vee c) \wedge (a \vee \bar{b}) \wedge (a \vee \bar{c})$$

$$a = \bar{b} \mapsto (a \vee b) \wedge (\bar{a} \vee \bar{b})$$

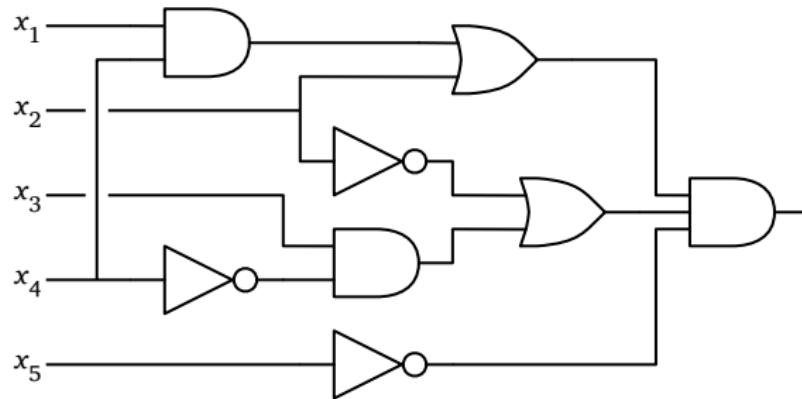
- 4 **Trasforma la formula in 3CNF aggiungendo variabili alle clausole con meno di tre letterali:**

$$a \vee b \mapsto (a \vee b \vee x) \wedge (a \vee b \vee \bar{x})$$

$$a \mapsto (a \vee x \vee y) \wedge (a \vee \bar{x} \vee y) \wedge (a \vee x \vee \bar{y}) \wedge (a \vee \bar{x} \vee \bar{y})$$

Esempio di Riduzione

- Dal circuito booleano ...



Esempio di Riduzione

■ ... alla formula in 3CNF

$$\begin{aligned} & (y_1 \vee \overline{x_1} \vee \overline{x_4}) \wedge (\overline{y_1} \vee x_1 \vee z_1) \wedge (\overline{y_1} \vee x_1 \vee \overline{z_1}) \wedge (\overline{y_1} \vee x_4 \vee z_2) \wedge (\overline{y_1} \vee x_4 \vee \overline{z_2}) \\ & \wedge (y_2 \vee x_4 \vee z_3) \wedge (y_2 \vee x_4 \vee \overline{z_3}) \wedge (\overline{y_2} \vee \overline{x_4} \vee z_4) \wedge (\overline{y_2} \vee \overline{x_4} \vee \overline{z_4}) \\ & \wedge (y_3 \vee \overline{x_3} \vee \overline{y_2}) \wedge (\overline{y_3} \vee x_3 \vee z_5) \wedge (\overline{y_3} \vee x_3 \vee \overline{z_5}) \wedge (\overline{y_3} \vee y_2 \vee z_6) \wedge (\overline{y_3} \vee y_2 \vee \overline{z_6}) \\ & \wedge (\overline{y_4} \vee y_1 \vee x_2) \wedge (y_4 \vee \overline{x_2} \vee z_7) \wedge (y_4 \vee \overline{x_2} \vee \overline{z_7}) \wedge (y_4 \vee \overline{y_1} \vee z_8) \wedge (y_4 \vee \overline{y_1} \vee \overline{z_8}) \\ & \wedge (y_5 \vee x_2 \vee z_9) \wedge (y_5 \vee x_2 \vee \overline{z_9}) \wedge (\overline{y_5} \vee \overline{x_2} \vee z_{10}) \wedge (\overline{y_5} \vee \overline{x_2} \vee \overline{z_{10}}) \\ & \wedge (y_6 \vee x_5 \vee z_{11}) \wedge (y_6 \vee x_5 \vee \overline{z_{11}}) \wedge (\overline{y_6} \vee \overline{x_5} \vee z_{12}) \wedge (\overline{y_6} \vee \overline{x_5} \vee \overline{z_{12}}) \\ & \wedge (\overline{y_7} \vee y_3 \vee y_5) \wedge (y_7 \vee \overline{y_3} \vee z_{13}) \wedge (y_7 \vee \overline{y_3} \vee \overline{z_{13}}) \wedge (y_7 \vee \overline{y_5} \vee z_{14}) \wedge (y_7 \vee \overline{y_5} \vee \overline{z_{14}}) \\ & \wedge (y_8 \vee \overline{y_4} \vee \overline{y_7}) \wedge (\overline{y_8} \vee y_4 \vee z_{15}) \wedge (\overline{y_8} \vee y_4 \vee \overline{z_{15}}) \wedge (\overline{y_8} \vee y_7 \vee z_{16}) \wedge (\overline{y_8} \vee y_7 \vee \overline{z_{16}}) \\ & \wedge (y_9 \vee \overline{y_8} \vee \overline{y_6}) \wedge (\overline{y_9} \vee y_8 \vee z_{17}) \wedge (\overline{y_9} \vee y_6 \vee z_{18}) \wedge (\overline{y_9} \vee y_6 \vee \overline{z_{18}}) \wedge (\overline{y_9} \vee y_8 \vee \overline{z_{17}}) \\ & \wedge (y_9 \vee z_{19} \vee z_{20}) \wedge (y_9 \vee \overline{z_{19}} \vee z_{20}) \wedge (y_9 \vee z_{19} \vee \overline{z_{20}}) \wedge (y_9 \vee \overline{z_{19}} \vee \overline{z_{20}}) \end{aligned}$$

Algoritmo polinomiale per 2SAT

(soddisficiabilità di formule in 2CNF):

- Prendiamo una variabile x e assegnamo valore 1 (vero)
- In ogni clausola con \bar{x} , l'altro letterale deve essere vero
 - **Esempio:** in $(\bar{x} \vee \bar{y})$, y deve essere falso (0)
- Continuiamo assegnando le variabili il cui valore è “**forzato**”

si continua finché... .

- L'algoritmo assegna valori alle variabili finché non succede **una di tre cose**:
 - 1 una **contraddizione**: una variabile è forzata ad essere sia vera che falsa
 - 2 tutte le variabili con valore forzato sono state assegnate, ma ancora **ci sono clausole non soddisfatte**
 - 3 si **ottiene un assegnamento** che dà valore vero alla formula

si continua finché... .

- L'algoritmo assegna valori alle variabili finché non succede **una di tre cose**:
 - 1 una **contraddizione**: una variabile è forzata ad essere sia vera che falsa
 - 2 tutte le variabili con valore forzato sono state assegnate, ma ancora **ci sono clausole non soddisfatte**
 - 3 si **ottiene un assegnamento** che dà valore vero alla formula
- **Di conseguenza**:
 - 1 Ci può essere un assegnamento che dà valore vero solo valore falso per la variabile x . **Ricominciamo** assegnando x a 0 (falso).
 - 2 Ci sono ancora **variabili e clausole che non sono assegnate**. **Eliminiamo le clausole soddisfatte**, e ripartiamo.
 - 3 La **risposta è Si** (ho trovato un assegnamento che soddisfa la formula).

Esercizio

- Utilizzare l'algoritmo polinomiale per **2SAT** per verificare se le seguenti formule in 2CNF sono soddisfacibili:
 - $\Phi_1 = (\bar{x} \vee y) \wedge (\bar{y} \vee z) \wedge (\bar{z} \vee \bar{x})$
 - $\Phi_2 = (x \vee y) \wedge (y \vee \bar{x}) \wedge (x \vee \bar{y}) \wedge (\bar{x} \vee \bar{y})$

Sommario

- 1 Tesi di Church computazionale
- 2 Riduzioni polinomiali
- 3 Il Re dei problemi NP
- 4 Problemi su grafi
- 5 Conclusioni

Il problema del massimo insieme indipendente

- Sia $G = (V, E)$ un grafo non orientato.
- Un **insieme indipendente** in G è un sottoinsieme I dei vertici tali che per ogni coppia di vertici in I , non c'è **nessun arco che li collega**

Il problema del massimo insieme indipendente

- Sia $G = (V, E)$ un grafo non orientato.
- Un **insieme indipendente** in G è un sottoinsieme I dei vertici tali che per ogni coppia di vertici in I , non c'è **nessun arco che li collega**

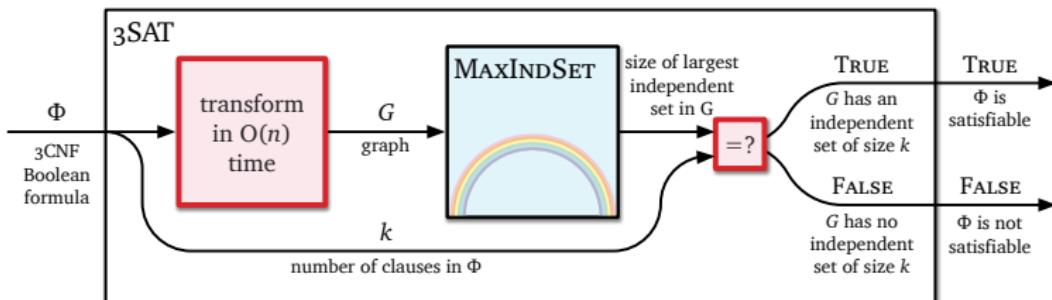
Problema del Massimo Insieme Indipendente (**MaxIndSet**)

Input: un grafo non orientato $G = (V, E)$

Output: la dimensione k dell'**insieme indipendente più grande** in G

MaxIndSet è NP-hard!

Dimostriamo che **MaxIndSet** è NP-hard con una **riduzione** da **3SAT**



Da 3SAT a MaxIndSet

Data una formula arbitraria in **3CNF**, come per esempio

$$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{c})$$

costruiamo un grafo $G = (V, E)$ tale che:

Da 3SAT a MaxIndSet

Data una formula arbitraria in **3CNF**, come per esempio

$$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{c})$$

costruiamo un grafo $G = (V, E)$ tale che:

- V contiene **3 vertici per clausola**, 1 per letterale

Da 3SAT a MaxIndSet

Data una formula arbitraria in **3CNF**, come per esempio

$$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{c})$$

costruiamo un grafo $G = (V, E)$ tale che:

- V contiene **3 vertici per clausola**, 1 per letterale
- Gli archi in E sono di **due tipi**

Da 3SAT a MaxIndSet

Data una formula arbitraria in **3CNF**, come per esempio

$$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{c})$$

costruiamo un grafo $G = (V, E)$ tale che:

- V contiene **3 vertici per clausola**, 1 per letterale
- Gli archi in E sono di **due tipi**
- **Archi di clausola** che collegano letterali nella stessa clausola

Da 3SAT a MaxIndSet

Data una formula arbitraria in **3CNF**, come per esempio

$$(a \vee b \vee c) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b} \vee \bar{c})$$

costruiamo un grafo $G = (V, E)$ tale che:

- V contiene **3 vertici per clausola**, 1 per letterale
- Gli archi in E sono di **due tipi**
- **Archi di clausola** che collegano letterali nella stessa clausola
- **Archi di consistenza** che collegano un letterale con la sua negazione

Correttezza della riduzione

- Se k è il **numero di clausole** nella formula, allora un insieme indipendente in G può contenere al più k elementi.
- Dimostriamo che G contiene un insieme indipendente di dimensione **esattamente k** se e solo se la formula Φ è soddisfacibile:

Correttezza della riduzione

- Se k è il **numero di clausole** nella formula, allora un insieme indipendente in G può contenere al più k elementi.
- Dimostriamo che G contiene un insieme indipendente di dimensione **esattamente k** se e solo se la formula Φ è **soddisfacibile**:
 - ⇒ Se la formula è soddisfacibile allora esiste un assegnamento delle variabili che la rende vera. Scegliamo un sottoinsieme S di k vertici di G , uno per ogni clausola, in modo che il letterale corrispondente sia vero. Si può far vedere che S è un insieme indipendente per G .

Correttezza della riduzione

- Se k è il **numero di clausole** nella formula, allora un insieme indipendente in G può contenere al più k elementi.
- Dimostriamo che G contiene un insieme indipendente di dimensione **esattamente k** se e solo se la formula Φ è **soddisfacibile**:
 - ⇒ Se la formula è soddisfacibile allora esiste un assegnamento delle variabili che la rende vera. Scegliamo un sottoinsieme S di k vertici di G , uno per ogni clausola, in modo che il letterale corrispondente sia vero. Si può far vedere che S è un insieme indipendente per G .
 - ⇐ Supponiamo che G contenga un insieme indipendente S di dimensione k . Ogni vertice di S deve stare in un triangolo diverso. Se assegnamo il valore Vero ai letterali presenti in S otteniamo un assegnamento che rende vera la formula.

Correttezza della riduzione

- Se k è il **numero di clausole** nella formula, allora un insieme indipendente in G può contenere al più k elementi.
- Dimostriamo che G contiene un insieme indipendente di dimensione **esattamente k** se e solo se la formula Φ è **soddisfacibile**:
 - ⇒ Se la formula è soddisfacibile allora esiste un assegnamento delle variabili che la rende vera. Scegliamo un sottoinsieme S di k vertici di G , uno per ogni clausola, in modo che il letterale corrispondente sia vero. Si può far vedere che S è un insieme indipendente per G .
 - ⇐ Supponiamo che G contenga un insieme indipendente S di dimensione k . Ogni vertice di S deve stare in un triangolo diverso. Se assegnamo il valore Vero ai letterali presenti in S otteniamo un assegnamento che rende vera la formula.
- La costruzione del grafo richiede un tempo polinomiale.

Un problema apparentemente simile

- Sia $G = (V, E)$ un grafo arbitrario.
- Un **accoppiamento** o **insieme di archi indipendenti** in G è un sottoinsieme M degli archi tali che non c'è **nessun vertice in comune** tra due archi.

Un problema apparentemente simile

- Sia $G = (V, E)$ un grafo arbitrario.
- Un **accoppiamento** o **insieme di archi indipendenti** in G è un sottoinsieme M degli archi tali che non c'è **nessun vertice in comune** tra due archi.

Problema del Massimo Accoppiamento (**MaxMatch**)

Input: un grafo arbitrario G

Output: la dimensione k dell'**accoppiamento più grande** in G

Un problema apparentemente simile

- Sia $G = (V, E)$ un grafo arbitrario.
- Un **accoppiamento** o **insieme di archi indipendenti** in G è un sottoinsieme M degli archi tali che non c'è **nessun vertice in comune** tra due archi.

Problema del Massimo Accoppiamento (**MaxMatch**)

Input: un grafo arbitrario G

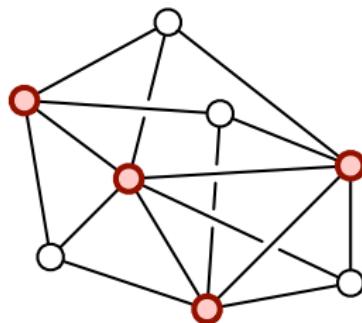
Output: la dimensione k dell'**accoppiamento più grande** in G

Algoritmo di Edmonds

Il problema dell'accoppiamento massimo è **risolvibile in tempo polinomiale!** Più precisamente, in tempo $O(|V|^2|E|)$

Il problema di copertura dei vertici

- Sia $G = (V, E)$ un grafo non orientato.
- Una **copertura tramite vertici** (Vertex Cover) in G è un sottoinsieme C dei vertici tali che **ogni arco ha almeno un'estremità in C**



MinVertexCover è NP-hard!

Problema del Minimum Vertex Cover (**MinVertexCover**)

Input: un grafo non orientato $G = (V, E)$

Output: la dimensione k della **più piccola copertura tramite vertici** di G

MinVertexCover è NP-hard!

Problema del Minimum Vertex Cover (**MinVertexCover**)

Input: un grafo non orientato $G = (V, E)$

Output: la dimensione k della più piccola copertura tramite vertici di G

Dimostriamo che **MinVertexCover** è NP-hard con una **riduzione** da **MaxIndSet**

MinVertexCover è NP-hard!

Problema del Minimum Vertex Cover (**MinVertexCover**)

Input: un grafo non orientato $G = (V, E)$

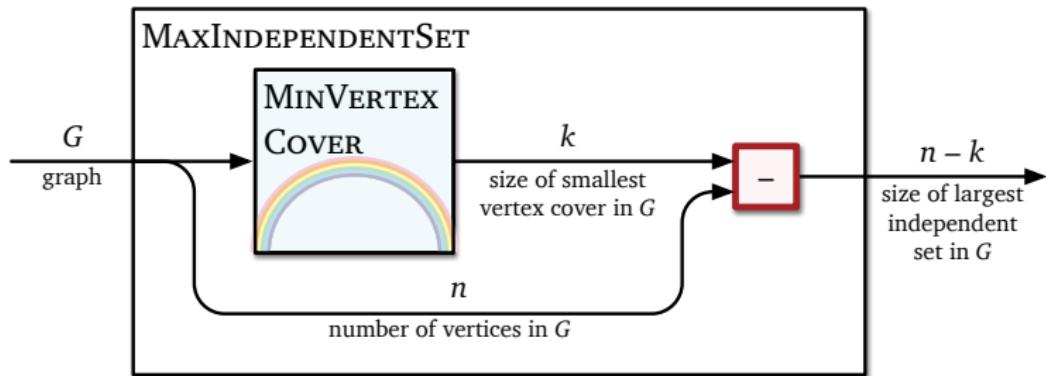
Output: la dimensione k della più piccola copertura tramite vertici di G

Dimostriamo che **MinVertexCover** è NP-hard con una **riduzione** da **MaxIndSet**

Fatto

I è un insieme indipendente di G , se e solo se $V \setminus I$ è una copertura tramite vertici di G .

Una riduzione facile





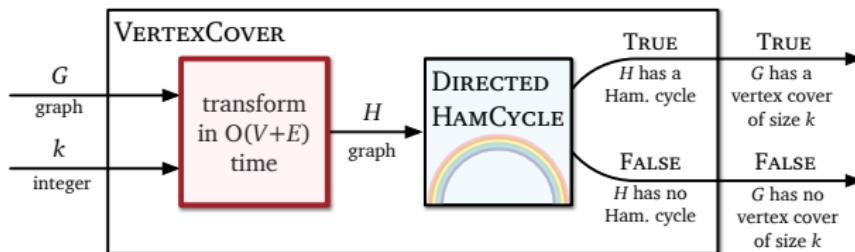
Torniamo al circuito Hamiltoniano

Dato un grafo G , un **Circuito Hamiltoniano** è un ciclo nel grafo che attraversa **tutti i vertici** una sola volta.

Torniamo al circuito Hamiltoniano

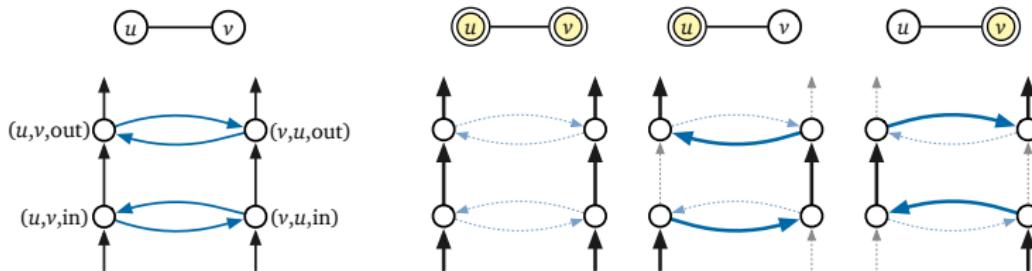
Dato un grafo G , un **Circuito Hamiltoniano** è un ciclo nel grafo che attraversa **tutti i vertici** una sola volta.

Dimostriamo che **DirectedHamCycle** è NP-hard con una **riduzione** da **VertexCover**



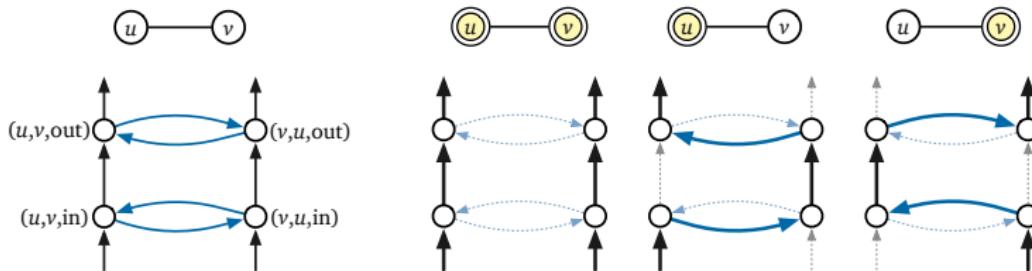
Edge gadget

1 Per ogni arco $uv \in G$, H contiene un **edge gadget**:



Edge gadget

- 1 Per ogni arco $uv \in G$, H contiene un **edge gadget**:

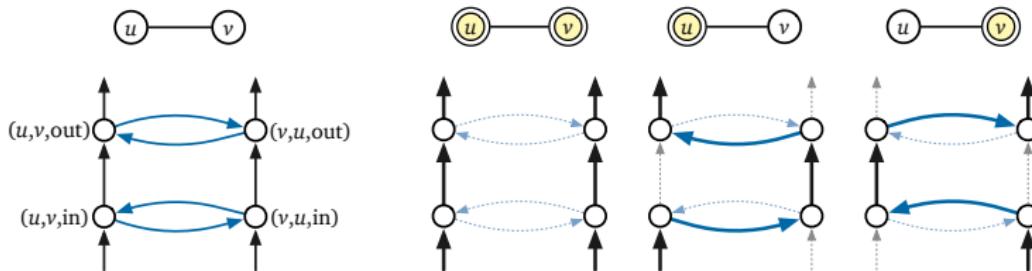


- 2 per ogni vertice $u \in G$ adiacente a v_1, \dots, v_d , una **vertex chain**

$$(u, v_i, out) \rightarrow (u, v_{i+1}, in) \text{ per } i = 1, \dots, d - 1$$

Edge gadget

- 1 Per ogni arco $uv \in G$, H contiene un **edge gadget**:

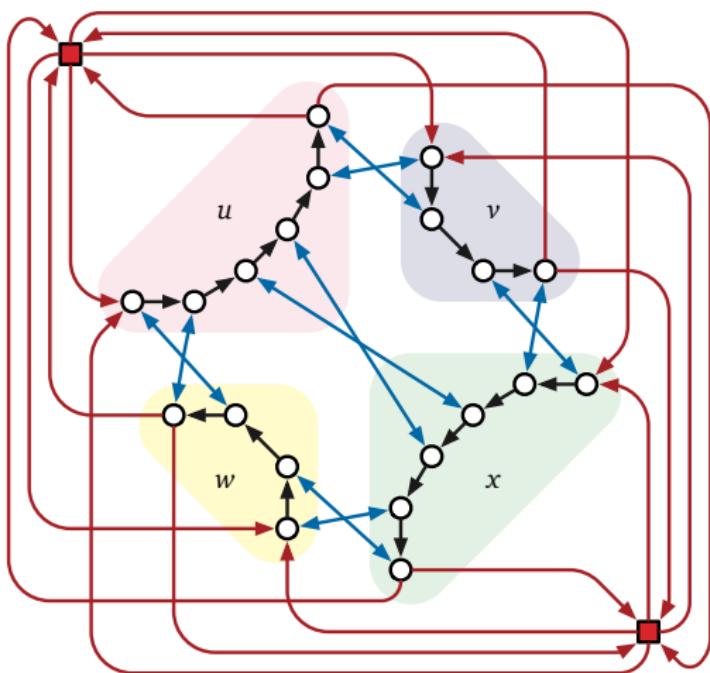
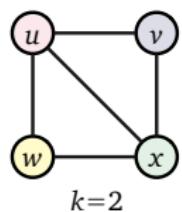


- 2 per ogni vertice $u \in G$ adiacente a v_1, \dots, v_d , una **vertex chain**

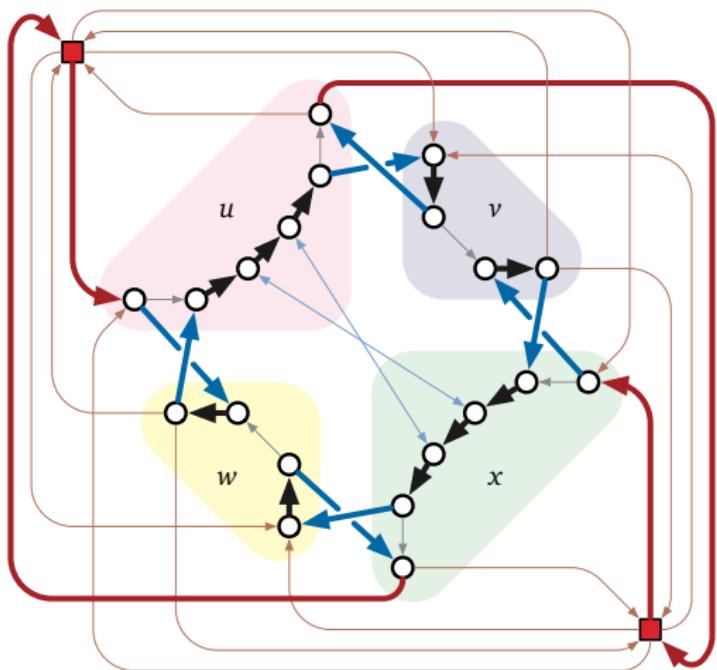
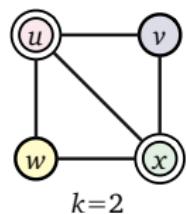
$$(u, v_i, out) \rightarrow (u, v_{i+1}, in) \text{ per } i = 1, \dots, d - 1$$

- 3 k cover vertex x_1, \dots, x_k , con un arco diretto verso il primo vertice di ogni vertex chain e uno da ogni ultimo vertice di ogni vertex chain

Esempio



Correttezza della riduzione



Sommario

- 1** Tesi di Church computazionale
- 2** Riduzioni polinomiali
- 3** Il Re dei problemi NP
- 4** Problemi su grafi
- 5** Conclusioni

Perché studiare la NP-completezza?

- Progettare ed implementare algoritmi richiede la conoscenza dei principi di base della teoria della complessità
- Stabilire che un problema è NP-completo costituisce una prova piuttosto forte della sua intrattabilità
- Conviene cercare di risolverlo con un approccio diverso . . .
 - identificare un caso particolare trattabile
 - cercare una soluzione approssimata
- . . . invece di cercare un algoritmo efficiente per il caso generale che probabilmente non esiste nemmeno

Altri esempi di dualità

Problemi NP-Completi

Problemi in P

Altri esempi di dualità

Problemi NP-Completi

■ circuito Hamiltoniano

Trovare un ciclo che visita **ogni vertice** esattamente una volta

Problemi in P

Altri esempi di dualità

Problemi NP-Completi

■ circuito Hamiltoniano

Trovare un ciclo che visita **ogni vertice** esattamente una volta

Problemi in P

■ Circuito Euleriano

Trovare un ciclo che visita **ogni arco** esattamente una volta

Altri esempi di dualità

Problemi NP-Completi

■ circuito Hamiltoniano

Trovare un ciclo che visita **ogni vertice** esattamente una volta

■ Copertura di vertici

Trovare il minimo sottoinsieme S di **vertici** tali che ogni arco ha almeno un'estremità in S

Problemi in P

■ Circuito Euleriano

Trovare un ciclo che visita **ogni arco** esattamente una volta

Altri esempi di dualità

Problemi NP-Completi

■ circuito Hamiltoniano

Trovare un ciclo che visita **ogni vertice** esattamente una volta

■ Copertura di vertici

Trovare il minimo sottoinsieme S di **vertici** tali che ogni arco ha almeno un'estremità in S

Problemi in P

■ Circuito Euleriano

Trovare un ciclo che visita **ogni arco** esattamente una volta

■ Copertura di archi

Trovare il minimo sottoinsieme M di **archi** tali che ogni vertice è adiacente ad un arco in M

Altri esempi di dualità

Problemi NP-Completi

■ circuito Hamiltoniano

Trovare un ciclo che visita **ogni vertice** esattamente una volta

■ Copertura di vertici

Trovare il minimo sottoinsieme S di **vertici** tali che ogni arco ha almeno un'estremità in S

■ 3-colorazione di un grafo

Trovare un modo per colorare i vertici di un grafo con **tre colori** tale che vertici adiacenti sono di colore diverso

Problemi in P

■ Circuito Euleriano

Trovare un ciclo che visita **ogni arco** esattamente una volta

■ Copertura di archi

Trovare il minimo sottoinsieme M di **archi** tali che ogni vertice è adiacente ad un arco in M

Altri esempi di dualità

Problemi NP-Completi

■ circuito Hamiltoniano

Trovare un ciclo che visita **ogni vertice** esattamente una volta

■ Copertura di vertici

Trovare il minimo sottoinsieme S di **vertici** tali che ogni arco ha almeno un'estremità in S

■ 3-colorazione di un grafo

Trovare un modo per colorare i vertici di un grafo con **tre colori** tale che vertici adiacenti sono di colore diverso

Problemi in P

■ Circuito Euleriano

Trovare un ciclo che visita **ogni arco** esattamente una volta

■ Copertura di archi

Trovare il minimo sottoinsieme M di **archi** tali che ogni vertice è adiacente ad un arco in M

■ 2-colorazione di un grafo

Trovare un modo per colorare i vertici di un grafo con **due colori** tale che vertici adiacenti sono di colore diverso

Come scegliere il problema giusto

- Se il problema richiede di **assegnare bit agli oggetti**: **SAT**
- Se il problema richiede di **assegnare etichette** agli oggetti prese un **piccolo insieme**, o di **partizionare** gli oggetti in un **numero costante di sottoinsiemi**: **3Color** o **kColor**
- Se il problema richiede di **organizzare** un insieme di oggetti **in un ordine particolare**: **Circuito Hamiltoniano**
- Se il problema richiede di trovare un **piccolo sottoinsieme** che soddisfi alcuni vincoli: **MinVertexCover**
- Se il problema richiede di trovare un **sottoinsieme grande** che soddisfi alcuni vincoli: **MaxIndependentSet**
- Se il **numero 3** appare in modo naturale nel problema, provare **3SAT** o **3Color** (No, questo non è uno scherzo.)
- Se tutto il resto fallisce, prova **3SAT** o anche **SAT**!