Complessità computazionale

La classe NP

Piero A. Bonatti

Università di Napoli Federico II

Laurea Magistrale in Informatica

Tema della lezione – I

- La classe di problemi **NP** è importante per diversi motivi
- Pratici: contiene molti problemi di concreto interesse applicativo
  - di ottimizzazione
  - di ragionamento automatico
- Teorici:
  - è una delle classi più piccole per cui non sono noti algoritmi polinomiali
  - le ragioni di questo (ricerca cieca in spazi ampi) ricompaiono in molte altre classi

Tema della lezione – I

- La classe di problemi **NP** è importante per diversi motivi
- Pratici: contiene molti problemi di concreto interesse applicativo
  - di ottimizzazione
  - di ragionamento automatico
- Teorici:
  - è una delle classi più piccole per cui non sono noti algoritmi polinomiali
  - le ragioni di questo (ricerca cieca in spazi ampi) ricompaiono in molte altre classi

Tema della lezione – I

- La classe di problemi **NP** è importante per diversi motivi
- Pratici: contiene molti problemi di concreto interesse applicativo
  - di ottimizzazione
  - di ragionamento automatico
- Teorici:
  - è una delle classi più piccole per cui non sono noti algoritmi polinomiali
  - le ragioni di questo (ricerca cieca in spazi ampi) ricompaiono in molte altre classi

Tema della lezione – II

- Pertanto dedicheremo alcuni CFU allo studio di **NP**
- In questo capitolo del corso
  - Analizzeremo la struttura comune dei problemi in **NP**
  - Forniremo alcuni esempi di tali problemi

Struttura comune dei problemi in **NP**

Struttura *generate and test*

- Nonostante la varietà di algoritmi esprimibili con MdT nondeterministiche che terminano in tempo polinomiale i problemi in **NP** sono risolubili con strategie algoritmiche relativamente uniformi
  1. Generare nondeterministicamente un "oggetto" che rappresenta una potenziale soluzione
  2. Controllare deterministicamente se davvero rappresenta una soluzione
- Questo tipo di procedimento viene modellato in astratto con le cosiddette relazioni polinomialmente bilanciate

Struttura *generate and test*

- Nonostante la varietà di algoritmi esprimibili con MdT nondeterministiche che terminano in tempo polinomiale i problemi in **NP** sono risolubili con strategie algoritmiche relativamente uniformi
  1. Generare nondeterministicamente un "oggetto" che rappresenta una potenziale soluzione
  2. Controllare deterministicamente se davvero rappresenta una soluzione
- Questo tipo di procedimento viene modellato in astratto con le cosiddette relazioni polinomialmente bilanciate

Struttura *generate and test*

- Nonostante la varietà di algoritmi esprimibili con MdT nondeterministiche che terminano in tempo polinomiale i problemi in **NP** sono risolubili con strategie algoritmiche relativamente uniformi
  1. Generare nondeterministicamente un "oggetto" che rappresenta una potenziale soluzione
  2. Controllare deterministicamente se davvero rappresenta una soluzione
- Questo tipo di procedimento viene modellato in astratto con le cosiddette relazioni polinomialmente bilanciate

Polynomially decidable relations

Definizione: Polynomially decidable relations

$R \subseteq \Sigma^* \times \Sigma^*$ è *polynomially decidable* sse il corrispondente linguaggio

$$\{x; y \mid (x, y) \in R\}$$

è deciso da una MdT deterministica in tempo polinomiale

Significato intuitivo:

1. x è l'istanza del problema e y rappresenta una soluzione candidata
   - ad es. un assegnamento di verità T, una permutazione π dei nodi di un grafo, ecc.
2. la MdT può controllare in tempo polinomiale se y "certifica" che la risposta a x è "yes"
   - ad es. se T soddisfa x, o se π ha costo $\leq K$

Polynomially decidable relations

Definizione: Polynomially decidable relations

$R \subseteq \Sigma^* \times \Sigma^*$ è *polynomially decidable* sse il corrispondente linguaggio

$$\{x; y \mid (x, y) \in R\}$$

è deciso da una MdT deterministica in tempo polinomiale

Significato intuitivo:

1. x è l'istanza del problema e y rappresenta una soluzione candidata
   - ad es. un assegnamento di verità T, una permutazione π dei nodi di un grafo, ecc.
2. la MdT può controllare in tempo polinomiale se y "certifica" che la risposta a x è "yes"
   - ad es. se T soddisfa x, o se π ha costo $\leq K$

Polynomially decidable relations

Definizione: Polynomially decidable relations

$R \subseteq \Sigma^* \times \Sigma^*$ è *polynomially decidable* sse il corrispondente linguaggio

$$\{x; y \mid (x, y) \in R\}$$

è deciso da una MdT deterministica in tempo polinomiale

Significato intuitivo:

1. x è l'istanza del problema e y rappresenta una soluzione candidata
   - ad es. un assegnamento di verità T, una permutazione π dei nodi di un grafo, ecc.
2. la MdT può controllare in tempo polinomiale se y "certifica" che la risposta a x è "yes"
   - ad es. se T soddisfa x, o se π ha costo $\leq K$

Polynomially balanced relations

Definizione: Polynomially balanced relations

Una relazione binaria $R \subseteq \Sigma^* \times \Sigma^*$ è *polynomially balanced* sse esiste $k \in \mathbb{N}$ t.c. per ogni $(x, y) \in R$,

$$|y| \leq |x|^k$$

Significato intuitivo:

- La "soluzione candidata" o "certificato" y ha dimensioni moderate
  - polinomiale nell'input x

Polynomially balanced relations

Definizione: Polynomially balanced relations

Una relazione binaria $R \subseteq \Sigma^* \times \Sigma^*$ è *polynomially balanced* sse esiste $k \in \mathbb{N}$ t.c. per ogni $(x, y) \in R$,

$$|y| \leq |x|^k$$

Significato intuitivo:

- La "soluzione candidata" o "certificato" y ha dimensioni moderate
  - polinomiale nell'input x

Caratterizzazione di **NP**

Teorema [Prop. 9.1 del Papadimitriou]

$L \in$ **NP** sse esiste una relazione R

1. polynomially decidable
2. polynomially balanced

tale che $L = \{x \mid (x, y) \in R \text{ per } \underline{\text{qualche}}\ y\}$

- **Significato intuitivo**: Si può sempre risolvere un problema in **NP** in tempo polinomiale con un metodo *generate and test*
  1. generare (nondeterministicamente) y (rappresenta la ricerca nello spazio di soluzioni)
     - la costruzione di y è polinomiale perchè $|y| \leq |x|^k$
  2. controllare (deterministicamente) se y certifica che $x \in L$
     - il test è polinomiale perchè R è polynomially decidable

Caratterizzazione di **NP**

Teorema [Prop. 9.1 del Papadimitriou]

$L \in$ **NP** sse esiste una relazione R

1. polynomially decidable
2. polynomially balanced

tale che $L = \{x \mid (x, y) \in R \text{ per } \underline{\text{qualche}}\ y\}$

- **Significato intuitivo**: Si può sempre risolvere un problema in **NP** in tempo polinomiale con un metodo *generate and test*
  1. generare (nondeterministicamente) y (rappresenta la ricerca nello spazio di soluzioni)
     - la costruzione di y è polinomiale perchè $|y| \leq |x|^k$
  2. controllare (deterministicamente) se y certifica che $x \in L$
     - il test è polinomiale perchè R è polynomially decidable

Caratterizzazione di **NP**

Teorema [Prop. 9.1 del Papadimitriou]

$L \in$ **NP** sse esiste una relazione R

1. polynomially decidable
2. polynomially balanced

tale che $L = \{x \mid (x, y) \in R \text{ per } \underline{\text{qualche}}\ y\}$

- **Significato intuitivo**: Si può sempre risolvere un problema in **NP** in tempo polinomiale con un metodo *generate and test*
  1. generare (nondeterministicamente) y (rappresenta la ricerca nello spazio di soluzioni)
     - la costruzione di y è polinomiale perchè $|y| \leq |x|^k$
  2. controllare (deterministicamente) se y certifica che $x \in L$
     - il test è polinomiale perchè R è polynomially decidable

Dimostrazione [Prop. 9.1 del Papadimitriou] – I

Parte I ("se")

- Supponiamo che esista una tale R. Dobbiamo mostrare che $L \in$ **NP**
  - cioè che esiste una MdT M che decide L in tempo polinomiale
- Definiamo $M = (M_1; M_2)$ dove:
- M_1 copia x sul nastro di output, poi aggiunge ';' e (nondeterministicamente) y
  - con un ciclo con $|x|^k$ iterazioni
  - ad ogni passo M_1 sceglie nondeterministicamente se scrivere sul nastro di output un simbolo di Σ oppure nessun simbolo
- M_2 è una MdT deterministica che in tempo polinomiale $p(|x|)$ decide R
  - esiste perchè R è *polynomially decidable*

Dimostrazione [Prop. 9.1 del Papadimitriou] – I

Parte I ("se")

- Supponiamo che esista una tale R. Dobbiamo mostrare che $L \in \mathbf{NP}$
  - cioè che esiste una MdT M che decide L in tempo polinomiale
- Definiamo $M = (M_1; M_2)$ dove:
- M_1 copia x sul nastro di output, poi aggiunge ';' e (nondeterministicamente) y
  - con un ciclo con $|x|^k$ iterazioni
  - ad ogni passo M_1 sceglie nondeterministicamente se scrivere sul nastro di output un simbolo di Σ oppure nessun simbolo
- M_2 è una MdT deterministica che in tempo polinomiale $p(|x|)$ decide R
  - esiste perchè R è *polynomially decidable*

Dimostrazione [Prop. 9.1 del Papadimitriou] – I

Parte I ("se")

- Supponiamo che esista una tale R. Dobbiamo mostrare che $L \in \mathbf{NP}$
  - cioè che esiste una MdT M che decide L in tempo polinomiale
- Definiamo $M = (M_1; M_2)$ dove:
- M_1 copia x sul nastro di output, poi aggiunge ';' e (nondeterministicamente) y
  - con un ciclo con $|x|^k$ iterazioni
  - ad ogni passo M_1 sceglie nondeterministicamente se scrivere sul nastro di output un simbolo di Σ oppure nessun simbolo
- M_2 è una MdT deterministica che in tempo polinomiale $p(|x|)$ decide R
  - esiste perchè R è *polynomially decidable*

Dimostrazione [Prop. 9.1 del Papadimitriou] – II

Parte I ("se")

- Tempo totale: $|x| + 1 + |x|^k + p(|x|)$ (polinomiale, di grado $\geq k$)
- Correttezza:

$$M(x) ='' \mathit{yes}'' \text{ sse } M_2(x; y) ='' \mathit{yes}'' \text{ sse } (x, y) \in R \text{ sse } x \in L.$$

- Quindi M è la MdT cercata

1/2 QED

Dimostrazione [Prop. 9.1 del Papadimitriou] – II

Parte I ("se")

- Tempo totale: $|x| + 1 + |x|^k + p(|x|)$ (polinomiale, di grado $\geq k$)
- Correttezza:

$$M(x) ='' yes'' \text{ sse } M_2(x; y) ='' yes'' \text{ sse } (x, y) \in R \text{ sse } x \in L.$$

- Quindi M è la MdT cercata

1/2 QED

Dimostrazione [Prop. 9.1 del Papadimitriou] – III

Parte II ("solo se")

- Supponiamo adesso che esista una MdT nondeterministica N che decide L in tempo $|x|^k$.
  - Dobbiamo definire R polynomially decidable/bounded
- Scegliamo come y le codifiche delle computazioni di $N(x)$

$$y = (s, w_0, u_0)(q_1, w_1, u_1) \cdots (q_t, w_t, u_t), \text{ con } t \leq |x|^k$$

- Il limite di tempo impone che la lunghezza dei $w_i u_i$ sia $\leq |x|^k$
  - quindi $|y| \leq (|x|^k + 5) \cdot |x|^k = O(|x|^{2k})$

  $\Rightarrow$ R is polynomially bounded
- Resta da dimostrare che R è *polynomially decidable*

Dimostrazione [Prop. 9.1 del Papadimitriou] – III

Parte II ("solo se")

- Supponiamo adesso che esista una MdT nondeterministica N che decide L in tempo $|x|^k$.
  - Dobbiamo definire R polynomially decidable/bounded
- Scegliamo come y le codifiche delle computazioni di $N(x)$

$$y = (s, w_0, u_0)(q_1, w_1, u_1) \cdots (q_t, w_t, u_t), \text{ con } t \leq |x|^k$$

- Il limite di tempo impone che la lunghezza dei $w_i u_i$ sia $\leq |x|^k$
  - quindi $|y| \leq (|x|^k + 5) \cdot |x|^k = O(|x|^{2k})$

  $\Rightarrow$ R is polynomially bounded
- Resta da dimostrare che R è *polynomially decidable*

Dimostrazione [Prop. 9.1 del Papadimitriou] – III

Parte II ("solo se")

- Supponiamo adesso che esista una MdT nondeterministica N che decide L in tempo $|x|^k$.
  - Dobbiamo definire R polynomially decidable/bounded
- Scegliamo come y le codifiche delle computazioni di $N(x)$

$$y = (s, w_0, u_0)(q_1, w_1, u_1) \cdots (q_t, w_t, u_t), \text{ con } t \leq |x|^k$$

- Il limite di tempo impone che la lunghezza dei $w_i u_i$ sia $\leq |x|^k$
  - quindi $|y| \leq (|x|^k + 5) \cdot |x|^k = O(|x|^{2k})$

$\Rightarrow$ R is polynomially bounded

- Resta da dimostrare che R è *polynomially decidable*

Dimostrazione [Prop. 9.1 del Papadimitriou] – III

Parte II ("solo se")

- Supponiamo adesso che esista una MdT nondeterministica N che decide L in tempo $|x|^k$.
  - Dobbiamo definire R polynomially decidable/bounded
- Scegliamo come y le codifiche delle computazioni di $N(x)$

$$y = (s, w_0, u_0)(q_1, w_1, u_1) \cdots (q_t, w_t, u_t), \text{ con } t \leq |x|^k$$

- Il limite di tempo impone che la lunghezza dei $w_i u_i$ sia $\leq |x|^k$
  - quindi $|y| \leq (|x|^k + 5) \cdot |x|^k = O(|x|^{2k})$

  $\Rightarrow$ R is polynomially bounded
- Resta da dimostrare che R è *polynomially decidable*

Dimostrazione [Prop. 9.1 del Papadimitriou] – IV

Parte II ("solo se")

Per verificare se y codifica una computazione di N

1. Controllare correttezza config. iniziale poi scandire y verso destra. Per ogni (q_i, w_i, u_i):
   1. scrivere su di un nastro ausiliario l'effetto delle possibili azioni di N, una dopo l'altra

$$(q'_1, w'_1, u'_1) \cdots (q'_d, w'_d, u'_d) \qquad \text{(tempo } O(|y|)\text{)}$$

   2. cercare successiva config. $(q_{i+1}, w_{i+1}, u_{i+1})$ nell'input
   3. verificare se è uguale a una delle configurazioni sul nastro ausiliario (tempo $O(|y|)$)
   4. Se nessuna corrisponde terminare con "no"
2. Al termine del nastro, terminare con "yes"

Dimostrazione [Prop. 9.1 del Papadimitriou] – IV

Parte II ("solo se")

Per verificare se y codifica una computazione di N

1. Controllare correttezza config. iniziale poi scandire y verso destra. Per ogni (q_i, w_i, u_i):
   1. scrivere su di un nastro ausiliario l'effetto delle possibili azioni di N, una dopo l'altra

      $$(q'_1, w'_1, u'_1) \cdots (q'_d, w'_d, u'_d) \qquad \text{(tempo } O(|y|))$$

   2. cercare successiva config. $(q_{i+1}, w_{i+1}, u_{i+1})$ nell'input
   3. verificare se è uguale a una delle configurazioni sul nastro ausiliario (tempo $O(|y|)$)
   4. Se nessuna corrisponde terminare con "no"
2. Al termine del nastro, terminare con "yes"

Dimostrazione [Prop. 9.1 del Papadimitriou] – IV

Parte II ("solo se")

Per verificare se y codifica una computazione di N

1. Controllare correttezza config. iniziale poi scandire y verso destra. Per ogni (q_i, w_i, u_i):
   1. scrivere su di un nastro ausiliario l'effetto delle possibili azioni di N, una dopo l'altra

      $$(q'_1, w'_1, u'_1) \cdots (q'_d, w'_d, u'_d) \qquad \text{(tempo } O(|y|))$$

   2. cercare successiva config. $(q_{i+1}, w_{i+1}, u_{i+1})$ nell'input
   3. verificare se è uguale a una delle configurazioni sul nastro ausiliario (tempo $O(|y|)$)
   4. Se nessuna corrisponde terminare con "no"
2. Al termine del nastro, terminare con "yes"

Dimostrazione [Prop. 9.1 del Papadimitriou] – IV

Parte II ("solo se")

Per verificare se y codifica una computazione di N

1. Controllare correttezza config. iniziale poi scandire y verso destra. Per ogni (q_i, w_i, u_i):
   1. scrivere su di un nastro ausiliario l'effetto delle possibili azioni di N, una dopo l'altra

      $$(q'_1, w'_1, u'_1) \cdots (q'_d, w'_d, u'_d) \qquad \text{(tempo } O(|y|))$$

   2. cercare successiva config. $(q_{i+1}, w_{i+1}, u_{i+1})$ nell'input
   3. verificare se è uguale a una delle configurazioni sul nastro ausiliario (tempo $O(|y|)$)
   4. Se nessuna corrisponde terminare con "no"
2. Al termine del nastro, terminare con "yes"

Dimostrazione [Prop. 9.1 del Papadimitriou] – V

Parte II ("solo se")

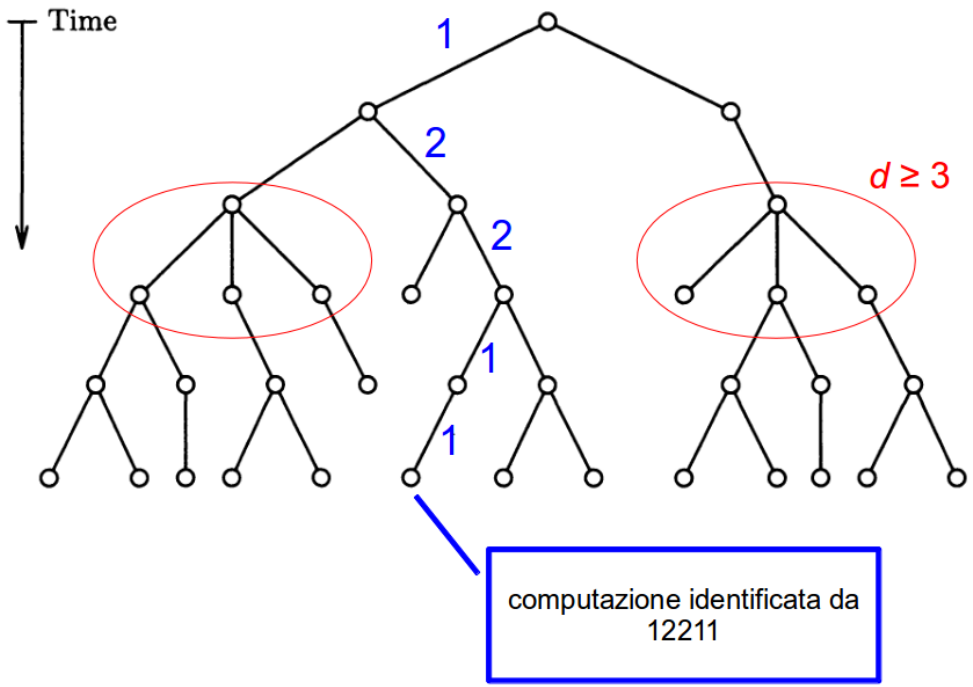
Complessità della decisione di R:

- Controllare la configurazione iniziale costa $O(|x|^2)$
- Poi il numero di iterazioni è $O(|x; y|)$ e ciascuna costa $O(|y|)$
- Costo totale: $O(n^2)$

QED

Tecnica di dimostrazione alternativa

- Usare come y le scelte nondeterministiche (interi tra 1 e d)
- Simulare N su nastro ausiliario eseguendo le azioni specificate da y
- Esempio: $y = 1, 2, 2, 1, 1$



Terminologia

Altri nomi per y:

- *succinct certificate*
- *polynomial witness* (testimone)

(del fatto che $x \in L$)

Ricapitolando: struttura dei problemi in **NP**

- Le istanze la cui risposta è "si" possiedono un certificato succinto (polinomiale)
  - il problema è trovarlo...
  - lo spazio di ricerca è $|\Sigma|^{|x|^k}$
- Il problema ammette sempre una soluzione *generate and test*
  - si "indovina" (guess) un certificato
  - si controlla (deterministicamente) se testimonia che la risposta è "si"

Ricapitolando: struttura dei problemi in **NP**

- Le istanze la cui risposta è "si" possiedono un certificato succinto (polinomiale)
  - il problema è trovarlo...
  - lo spazio di ricerca è $|\Sigma|^{|x|^k}$
- Il problema ammette sempre una soluzione *generate and test*
  - si "indovina" (guess) un certificato
  - si controlla (deterministicamente) se testimonia che la risposta è "si"

Una lista di problemi **NP**-completi

MAX2SAT

- Il problema MAX2SAT ci interessa da un punto di vista didattico
- La dimostrazione di **NP**-hardness fa uso di una tecnica molto usata:
  - definizione e utilizzo di *gadgets*
  - sottoproblemi elementari che vengono composti nella prova per codificare problemi più complessi

MAX2SAT

Definizione

MAX2SAT

Dati un insieme di clausole, ciascuna con 2 letterali (come in 2-SAT) e un intero K, dire se esiste un truth assignment che soddisfa almeno K delle clausole

- Nota: è il problema di decisione corrispondente a trovare un truth assignment che massimizza il numero di clausole soddisfatte
- La versione non ristretta a clausole binarie viene utilizzata per "riparare" le basi di conoscenza inconsistenti
- Questa versione serve a mostrare che la massimizzazione è una fonte di complessità; può rendere "difficile" anche un problema di ragionamento facile

MAX2SAT è **NP**-hard

Dimostrazione – Il *gadget*

- Considerare le 10 clausole $(x)(y)(z)(w)$ $(\neg x \vee \neg y)(\neg y \vee \neg z)(\neg z \vee \neg x)$ $(x \vee \neg w)(y \vee \neg w)(z \vee \neg w)$
- Sono insoddisfacibili (ad es. prime 2 righe).
  Quante ne posso soddisfare? Sfruttare simmetria di x, y, z
  - Se tutte vere, perdo la 2^a riga; con $w = \mathit{true}$ soddisfo tutto il resto (7 clausole soddisfatte)
  - Se 2 sono vere: perdo 1 clausola dalla 1^a riga e una dalla 2^a. Con $w = \mathit{true}$ soddisfo anche una della 1^a riga mentre con $w = \mathit{false}$ una della 3^a. (ancora 7)
  - Se 1 vera: soddisfo una della 1^a riga, tutta la 2^a e una della 3^a. Se $w = \mathit{true}$ aggiungo una dalla 1^a; se $w = \mathit{false}$ 2 della 3^a (ancora 7)
  - Se tutte false: soddisfo al massimo 6 sole clausole

MAX2SAT è **NP**-hard

Dimostrazione – Il *gadget*

- Considerare le 10 clausole $(x)(y)(z)(w)$ $(\neg x \vee \neg y)(\neg y \vee \neg z)(\neg z \vee \neg x)$ $(x \vee \neg w)(y \vee \neg w)(z \vee \neg w)$
- Sono insoddisfacibili (ad es. prime 2 righe).
  Quante ne posso soddisfare? Sfruttare simmetria di x, y, z
  - Se tutte vere, perdo la 2^a riga; con $w = \mathit{true}$ soddisfo tutto il resto (7 clausole soddisfatte)
  - Se 2 sono vere: perdo 1 clausola dalla 1^a riga e una dalla 2^a. Con $w = \mathit{true}$ soddisfo anche una della 1^a riga mentre con $w = \mathit{false}$ una della 3^a. (ancora 7)
  - Se 1 vera: soddisfo una della 1^a riga, tutta la 2^a e una della 3^a. Se $w = \mathit{true}$ aggiungo una dalla 1^a; se $w = \mathit{false}$ 2 della 3^a (ancora 7)
  - Se tutte false: soddisfo al massimo 6 sole clausole

MAX2SAT è **NP**-hard

Dimostrazione – Il *gadget*

- Considerare le 10 clausole $(x)(y)(z)(w)$ $(\neg x \vee \neg y)(\neg y \vee \neg z)(\neg z \vee \neg x)$ $(x \vee \neg w)(y \vee \neg w)(z \vee \neg w)$
- Sono insoddisfacibili (ad es. prime 2 righe).
  Quante ne posso soddisfare? Sfruttare simmetria di x, y, z
  - Se tutte vere, perdo la 2^a riga; con $w = \mathit{true}$ soddisfo tutto il resto (7 clausole soddisfatte)
  - Se 2 sono vere: perdo 1 clausola dalla 1^a riga e una dalla 2^a. Con $w = \mathit{true}$ soddisfo anche una della 1^a riga mentre con $w = \mathit{false}$ una della 3^a. (ancora 7)
  - Se 1 vera: soddisfo una della 1^a riga, tutta la 2^a e una della 3^a. Se $w = \mathit{true}$ aggiungo una dalla 1^a; se $w = \mathit{false}$ 2 della 3^a (ancora 7)
  - Se tutte false: soddisfo al massimo 6 sole clausole

MAX2SAT è **NP**-hard

Dimostrazione – Il *gadget*

- Considerare le 10 clausole $(x)(y)(z)(w)$ $(\neg x \vee \neg y)(\neg y \vee \neg z)(\neg z \vee \neg x)$ $(x \vee \neg w)(y \vee \neg w)(z \vee \neg w)$
- Sono insoddisfacibili (ad es. prime 2 righe).
  Quante ne posso soddisfare? Sfruttare simmetria di x, y, z
  - Se tutte vere, perdo la 2^a riga; con $w = \mathit{true}$ soddisfo tutto il resto (7 clausole soddisfatte)
  - Se 2 sono vere: perdo 1 clausola dalla 1^a riga e una dalla 2^a. Con $w = \mathit{true}$ soddisfo anche una della 1^a riga mentre con $w = \mathit{false}$ una della 3^a. (ancora 7)
  - Se 1 vera: soddisfo una della 1^a riga, tutta la 2^a e una della 3^a. Se $w = \mathit{true}$ aggiungo una dalla 1^a; se $w = \mathit{false}$ 2 della 3^a (ancora 7)
  - Se tutte false: soddisfo al massimo 6 sole clausole

MAX2SAT è **NP**-hard

Dimostrazione – Il *gadget*

- Considerare le 10 clausole $(x)(y)(z)(w)$ $(\neg x \vee \neg y)(\neg y \vee \neg z)(\neg z \vee \neg x)$ $(x \vee \neg w)(y \vee \neg w)(z \vee \neg w)$
- Sono insoddisfacibili (ad es. prime 2 righe).
  Quante ne posso soddisfare? Sfruttare simmetria di x, y, z
  - Se tutte vere, perdo la 2^a riga; con $w = \mathit{true}$ soddisfo tutto il resto (7 clausole soddisfatte)
  - Se 2 sono vere: perdo 1 clausola dalla 1^a riga e una dalla 2^a. Con $w = \mathit{true}$ soddisfo anche una della 1^a riga mentre con $w = \mathit{false}$ una della 3^a. (ancora 7)
  - Se 1 vera: soddisfo una della 1^a riga, tutta la 2^a e una della 3^a. Se $w = \mathit{true}$ aggiungo una dalla 1^a; se $w = \mathit{false}$ 2 della 3^a (ancora 7)
  - Se tutte false: soddisfo al massimo 6 sole clausole

MAX2SAT è **NP**-hard

Dimostrazione – Il *gadget*

- Considerare le 10 clausole $(x)(y)(z)(w)$ $(\neg x \vee \neg y)(\neg y \vee \neg z)(\neg z \vee \neg x)$ $(x \vee \neg w)(y \vee \neg w)(z \vee \neg w)$
- Sono insoddisfacibili (ad es. prime 2 righe). Quante ne posso soddisfare? Sfruttare simmetria di x, y, z
  - Se tutte vere, perdo la 2^a riga; con $w = \mathit{true}$ soddisfo tutto il resto (7 clausole soddisfatte)
  - Se 2 sono vere: perdo 1 clausola dalla 1^a riga e una dalla 2^a. Con $w = \mathit{true}$ soddisfo anche una della 1^a riga mentre con $w = \mathit{false}$ una della 3^a. (ancora 7)
  - Se 1 vera: soddisfo una della 1^a riga, tutta la 2^a e una della 3^a. Se $w = \mathit{true}$ aggiungo una dalla 1^a; se $w = \mathit{false}$ 2 della 3^a (ancora 7)
  - Se tutte false: soddisfo al massimo 6 sole clausole

MAX2SAT è **NP**-hard

Dimostrazione – II *gadget* – II

- Ricapitolando le proprietà del gadget:
- Se un truth assignment soddisfa $(x \vee y \vee z)$ allora si possono soddisfare fino a 7 clausole (scegliendo opportunamente w)
- Altrimenti, se non soddisfa $(x \vee y \vee z)$, allora si arriva al massimo a 6 clausole

MAX2SAT è **NP**-hard

Dimostrazione – La *riduzione*

- Data una istanza ϕ di 3-SAT, la riduzione $R(\phi)$ a MAX2SAT è definita così:
  - per ciascuna clausola $(\alpha \vee \beta \vee \gamma)$ in ϕ,
  - includere in $R(\phi)$ una copia del gadget con x, y, z sostituiti da α, β, γ e w sostituita da una nuova proposizione w_i
- Totale clausole in $R(\phi) = 10 \cdot m$, dove m sono le clausole in ϕ
- Porre $K = 7 \cdot m$
  - è chiaramente il massimo raggiungibile
  - si può raggiungere solo se ϕ è soddisfacibile
  - ogni assegnamento in cui una clausola è falsa ne soddisfa al massimo 6 nel gadget corrispondente

  $\Rightarrow$ la riduzione è corretta

MAX2SAT è **NP**-hard

Dimostrazione – La *riduzione*

- Data una istanza ϕ di 3-SAT, la riduzione $R(\phi)$ a MAX2SAT è definita così:
  - per ciascuna clausola $(\alpha \vee \beta \vee \gamma)$ in ϕ,
  - includere in $R(\phi)$ una copia del gadget con x, y, z sostituiti da α, β, γ e w sostituita da una nuova proposizione w_i
- Totale clausole in $R(\phi) = 10 \cdot m$, dove m sono le clausole in ϕ
- Porre $K = 7 \cdot m$
  - è chiaramente il massimo raggiungibile
  - si può raggiungere solo se ϕ è soddisfacibile
  - ogni assegnamento in cui una clausola è falsa ne soddisfa al massimo 6 nel gadget corrispondente

  $\Rightarrow$ la riduzione è corretta

MAX2SAT è **NP**-hard

Dimostrazione – La *riduzione*

- Data una istanza ϕ di 3-SAT, la riduzione $R(\phi)$ a MAX2SAT è definita così:
  - per ciascuna clausola $(\alpha \vee \beta \vee \gamma)$ in ϕ,
  - includere in $R(\phi)$ una copia del gadget con x, y, z sostituiti da α, β, γ e w sostituita da una nuova proposizione w_i
- Totale clausole in $R(\phi) = 10 \cdot m$, dove m sono le clausole in ϕ
- Porre $K = 7 \cdot m$
  - è chiaramente il massimo raggiungibile
  - si può raggiungere solo se ϕ è soddisfacibile
  - ogni assegnamento in cui una clausola è falsa ne soddisfa al massimo 6 nel gadget corrispondente

  $\Rightarrow$ la riduzione è corretta

MAX2SAT è **NP**-hard

Dimostrazione – La *riduzione* – II

- Complessità: $R(\phi)$ si può calcolare in spazio logaritmico
- Ad esempio per ogni $(\alpha \vee \beta \vee \gamma)$ in ϕ si può:
  1. scrivere $\alpha, \beta, \gamma, w_i$ su 4 nastri ausiliari (spazio $O(\log |\phi|)$; perchè?)
  2. scrivere sull'output la copia del gadget, prendendo x, y, z, w dai 4 nastri

QED

MAX2SAT è **NP**-completo

- Sappiamo già che è **NP**-hard
- Basta mostrare che appartiene a **NP** – ovvero esiste una MdT nondeterministica M che lo risolve
- Cenno alla dimostrazione:
  - M genera nondeterministicamente un truth assignment T su un nastro ausiliario
    - $p_1 = true, p_2 = false, \ldots$
  - M scandisce l'input contando (su di un altro nastro) quante clausole sono soddisfatte da T
  - entrambi i passi richiedono tempo polinomiale (perchè?)
  - infine si confronta il contatore con K
    - oppure si inizializza il contatore a K e si decrementa fino a zero
  - **Nota:** il truth assignment funge da certificato/witness QED

MAX2SAT è **NP**-completo

- Sappiamo già che è **NP**-hard
- Basta mostrare che appartiene a **NP** – ovvero esiste una MdT nondeterministica M che lo risolve
- Cenno alla dimostrazione:
  - M genera nondeterministicamente un truth assignment T su un nastro ausiliario
    - $p_1 = true, p_2 = false, \ldots$
  - M scandisce l'input contando (su di un altro nastro) quante clausole sono soddisfatte da T
  - entrambi i passi richiedono tempo polinomiale (perchè?)
  - infine si confronta il contatore con K
    - oppure si inizializza il contatore a K e si decrementa fino a zero
  - **Nota:** il truth assignment funge da certificato/witness QED

MAX2SAT è **NP**-completo

- Sappiamo già che è **NP**-hard
- Basta mostrare che appartiene a **NP** – ovvero esiste una MdT nondeterministica M che lo risolve
- Cenno alla dimostrazione:
  - M genera nondeterministicamente un truth assignment T su un nastro ausiliario
    - $p_1 = true, p_2 = false, \ldots$
  - M scandisce l'input contando (su di un altro nastro) quante clausole sono soddisfatte da T
  - entrambi i passi richiedono tempo polinomiale (perchè?)
  - infine si confronta il contatore con K
    - oppure si inizializza il contatore a K e si decrementa fino a zero
  - **Nota:** il truth assignment funge da certificato/witness QED

MAX2SAT è **NP**-completo

- Sappiamo già che è **NP**-hard
- Basta mostrare che appartiene a **NP** – ovvero esiste una MdT nondeterministica M che lo risolve
- Cenno alla dimostrazione:
  - M genera nondeterministicamente un truth assignment T su un nastro ausiliario
    - $p_1 = \mathit{true}, p_2 = \mathit{false}, \ldots$
  - M scandisce l'input contando (su di un altro nastro) quante clausole sono soddisfatte da T
  - entrambi i passi richiedono tempo polinomiale (perchè?)
  - infine si confronta il contatore con K
    - oppure si inizializza il contatore a K e si decrementa fino a zero
  - **Nota:** il truth assignment funge da certificato/witness QED

Problemi su grafi non orientati

Premessa: grafi non orientati

- Nei grafi non orientati gli archi possono essere percorsi in entrambe le direzioni
  - equivalentemente, per ogni arco (i, j) ne esiste un (j, i)
  - inoltre non esistono "self loops", cioè archi (i, i)
- Notazione: un arco non orientato viene indicato con $[i, j]$

Independent Set

Definizione (insieme independente)

Dati un grafo non orientato $G = (V, E)$ e un insieme $I \subseteq V$
diciamo che I è indipendente se e solo se
per ogni $i, j \in I$, non c'è alcun arco tra i e j

Independent Set problem

Dati un grafo non orientato $G = (V, E)$ e un $K \in \mathbb{N}$
esiste un insieme indipendente $I \subseteq V$ tale che $|I| = K$?

Independent Set è **NP**-completo

Il gadget e la riduzione da 3-SAT

- Il gadget utilizzato è il triangolo

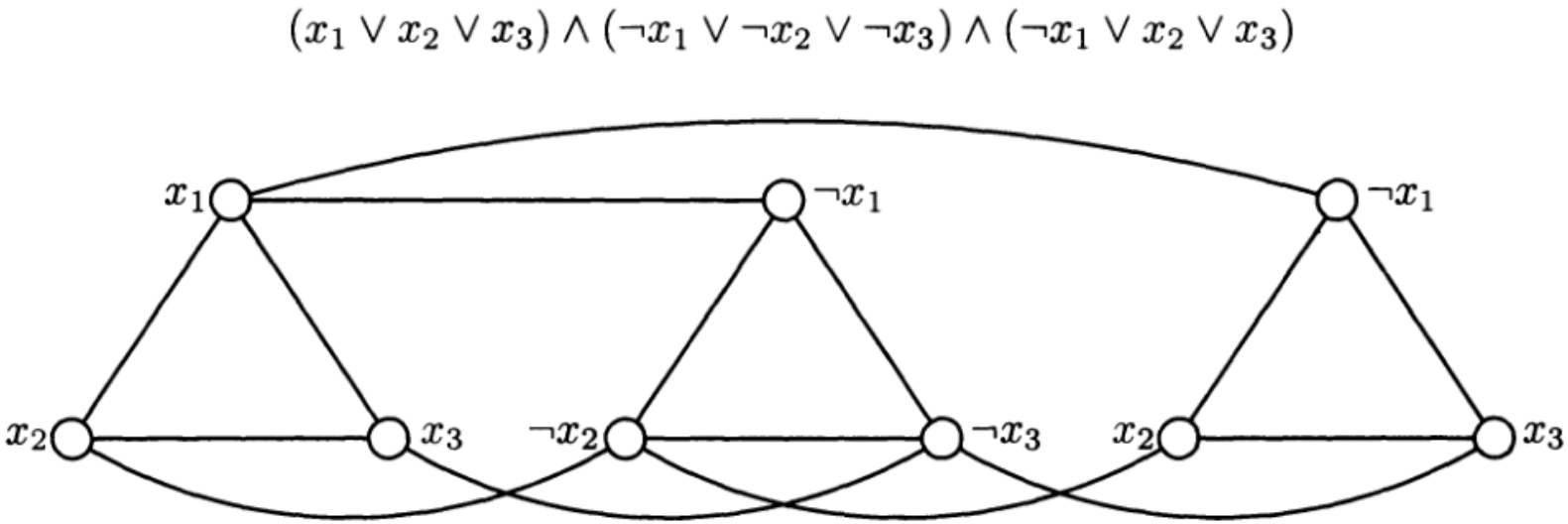


Figure 9-2. Reduction to INDEPENDENT SET.

Independent Set è **NP**-completo

Il gadget e la riduzione da 3-SAT

- uno per ogni 3-clausola di ϕ, istanza di 3-SAT
  supponiamo che vi siano m clausole

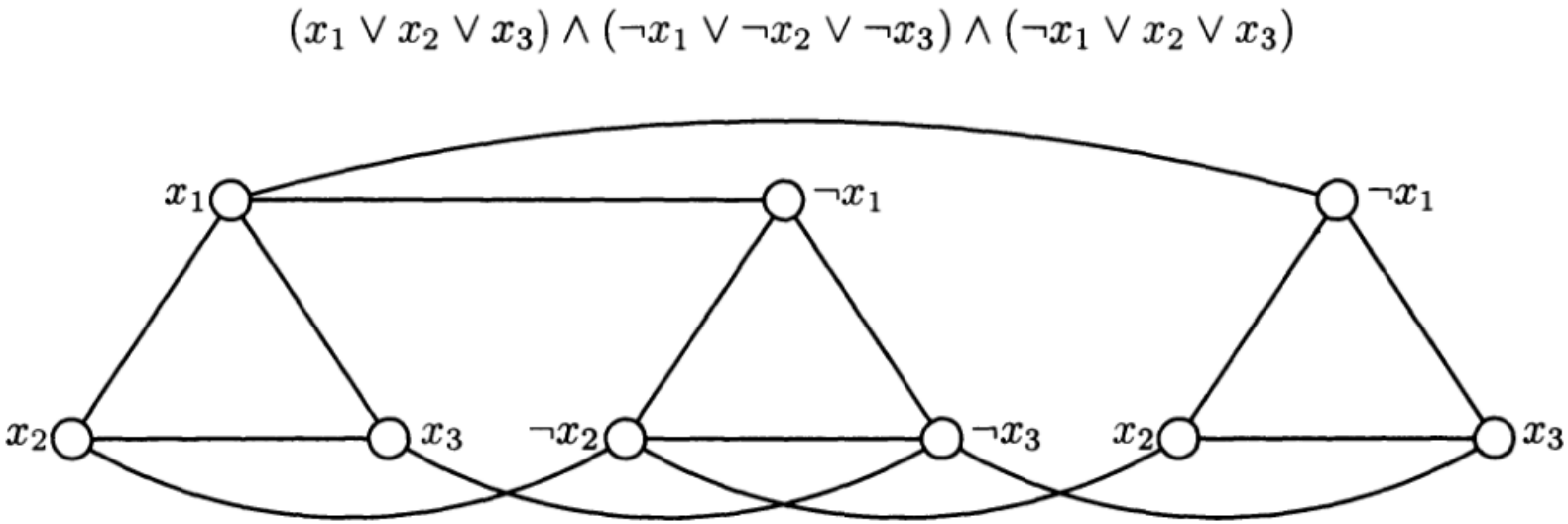


Figure 9-2. Reduction to INDEPENDENT SET.

Independent Set è **NP**-completo

Il gadget e la riduzione da 3-SAT

- letterali complementari uniti da archi $\Rightarrow$ mai insieme in I
  I non conterrà mai x_i e $\neg x_i$

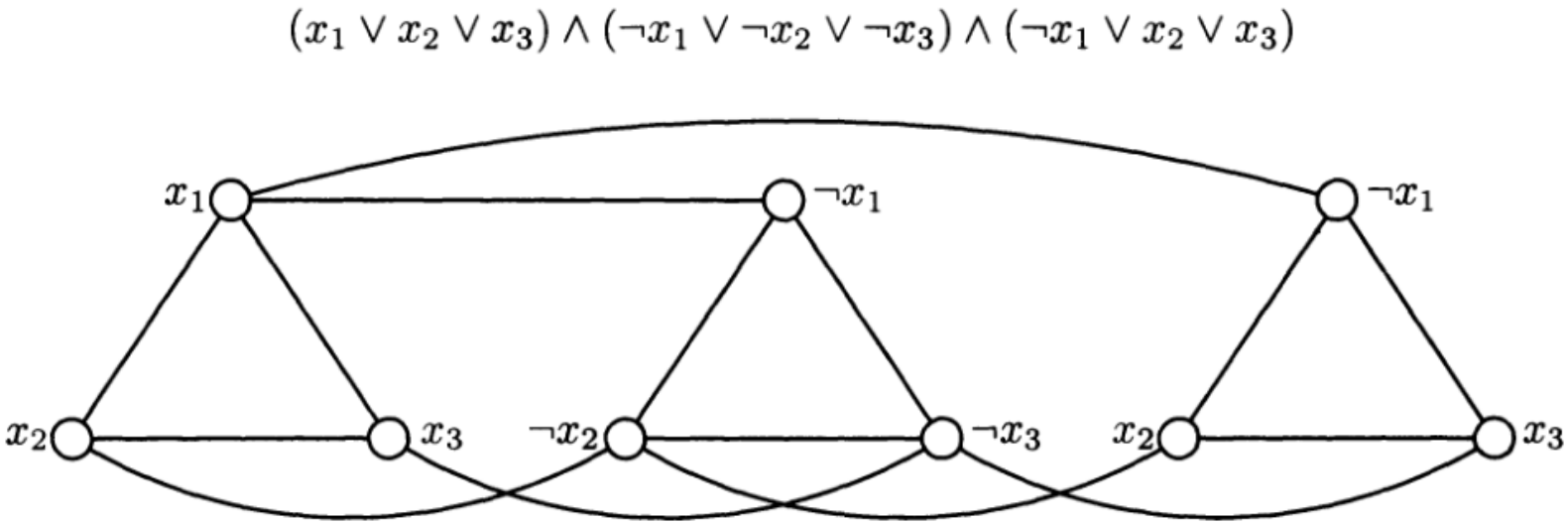


Figure 9-2. Reduction to INDEPENDENT SET.

Independent Set è **NP**-completo

Il gadget e la riduzione da 3-SAT

- Per ogni triangolo (clausola) si può prendere al massimo 1 nodo $\Rightarrow$ al massimo m nodi $\Rightarrow$ poniamo $K = m$

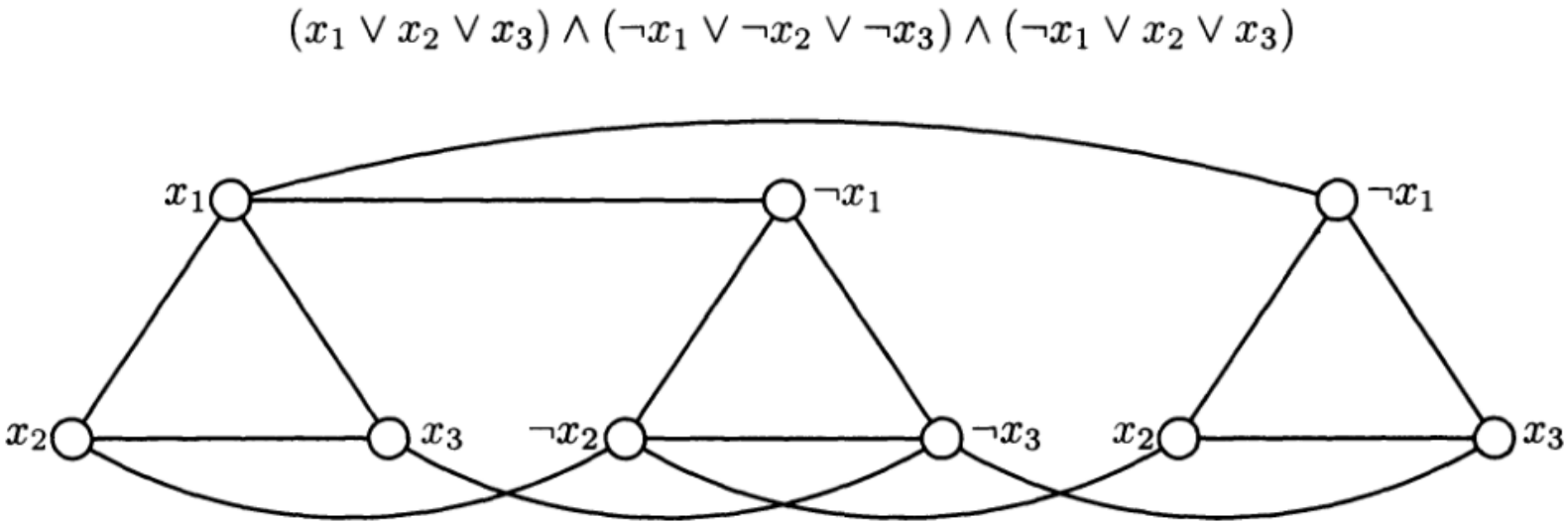


Figure 9-2. Reduction to INDEPENDENT SET.

Independent Set è **NP**-completo

Il gadget e la riduzione da 3-SAT

- Correttezza 1: Se T soddisfa ϕ, allora otteniamo I con m nodi prendendo da ogni triangolo 1 nodo soddisfatto da T

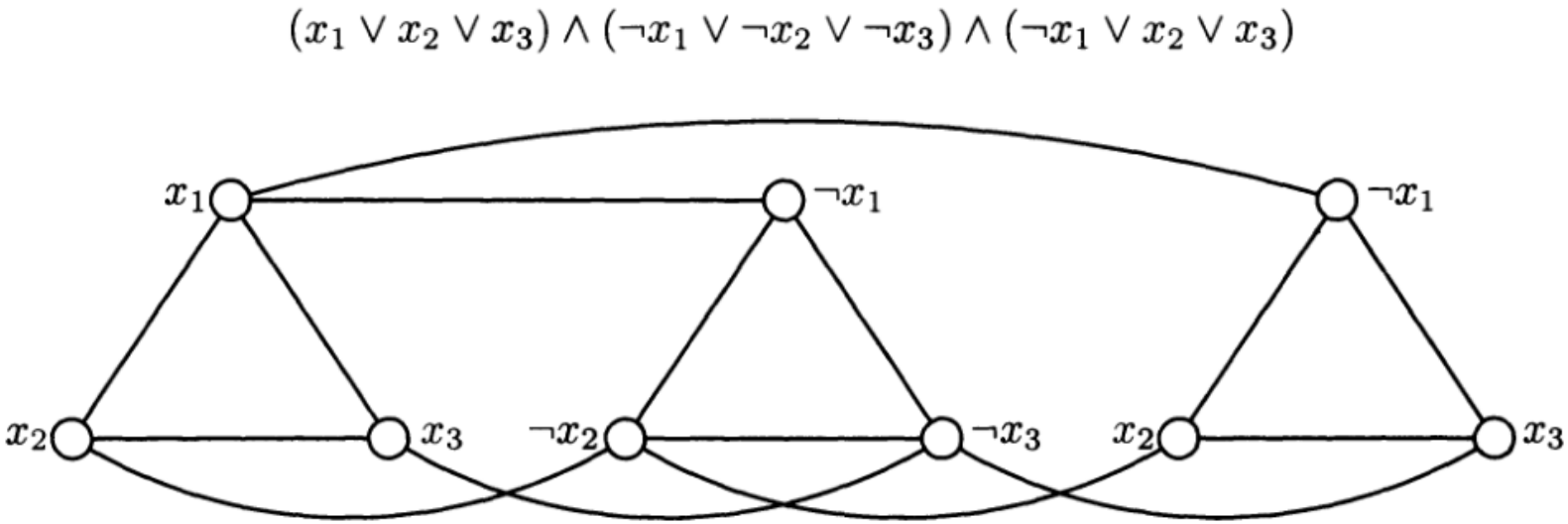


Figure 9-2. Reduction to INDEPENDENT SET.

Independent Set è **NP**-completo

Il gadget e la riduzione da 3-SAT

- Correttezza 2: Da I con m nodi definire T soddisfando i nodi in I

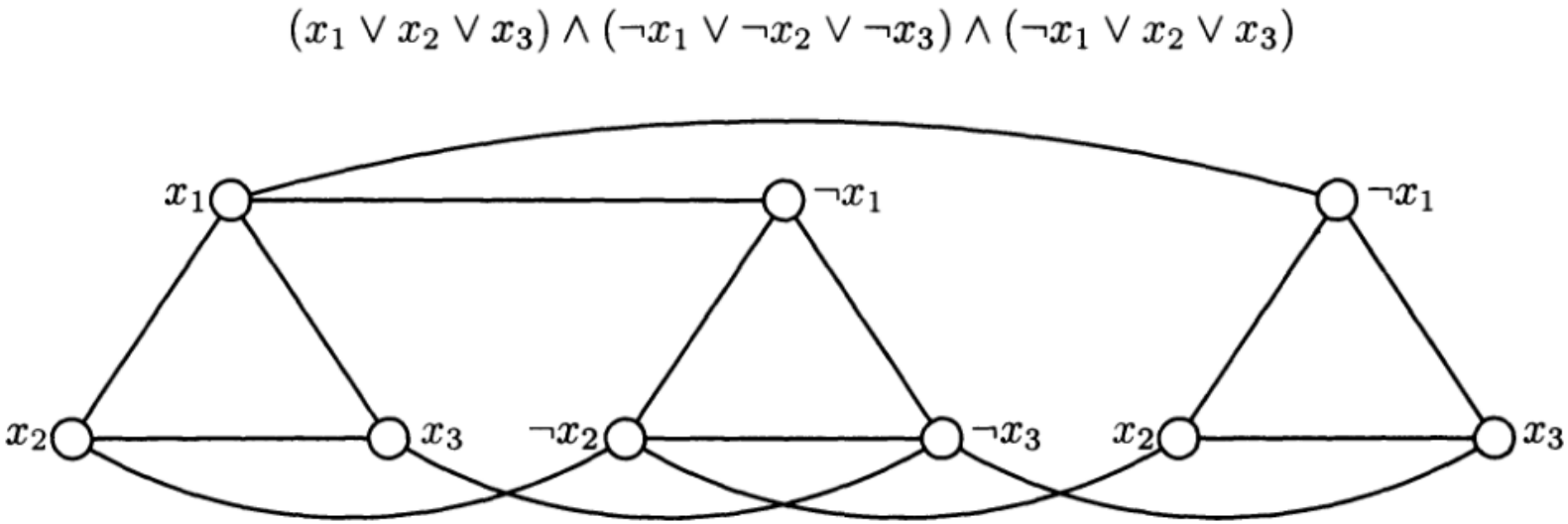


Figure 9-2. Reduction to INDEPENDENT SET.

Independent Set è **NP**-completo

Il gadget e la riduzione da 3-SAT

- Complessità: scrivere su 2 nastri ausiliari gli indici dei nodi da collegare e con quelli costruire gli archi ($O(\log m)$)

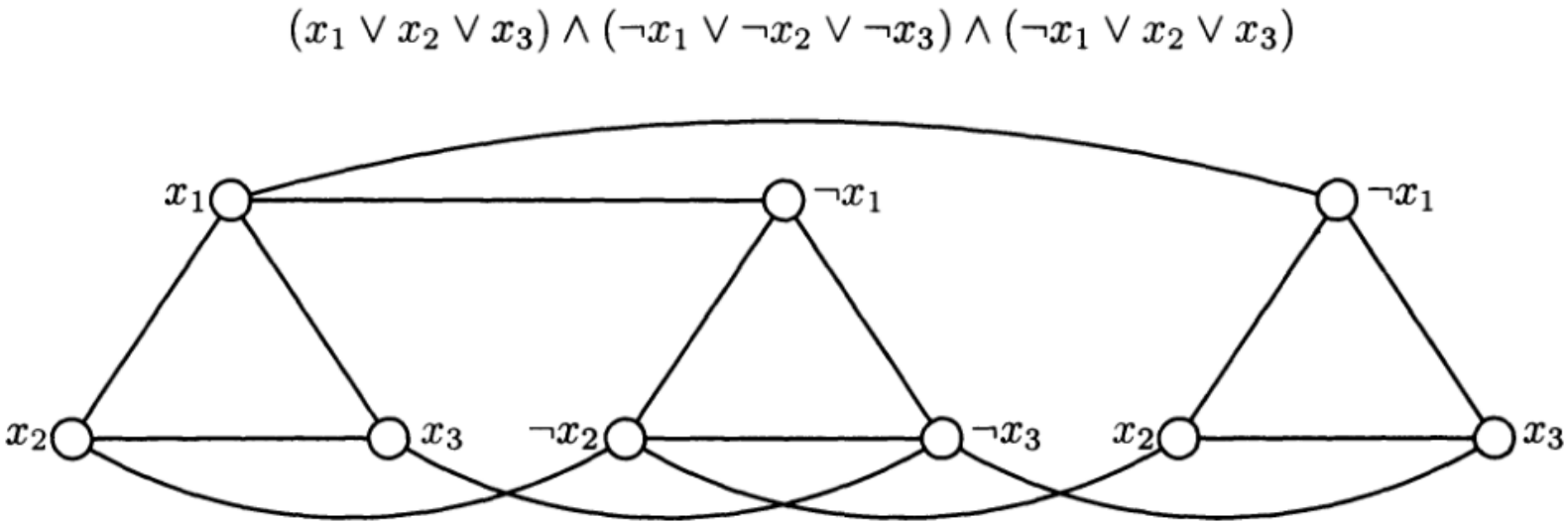


Figure 9-2. Reduction to INDEPENDENT SET.

Independent Set è **NP**-completo

Il gadget e la riduzione da 3-SAT

- Quindi questa è una vera riduzione e Independent Set è **NP**-hard

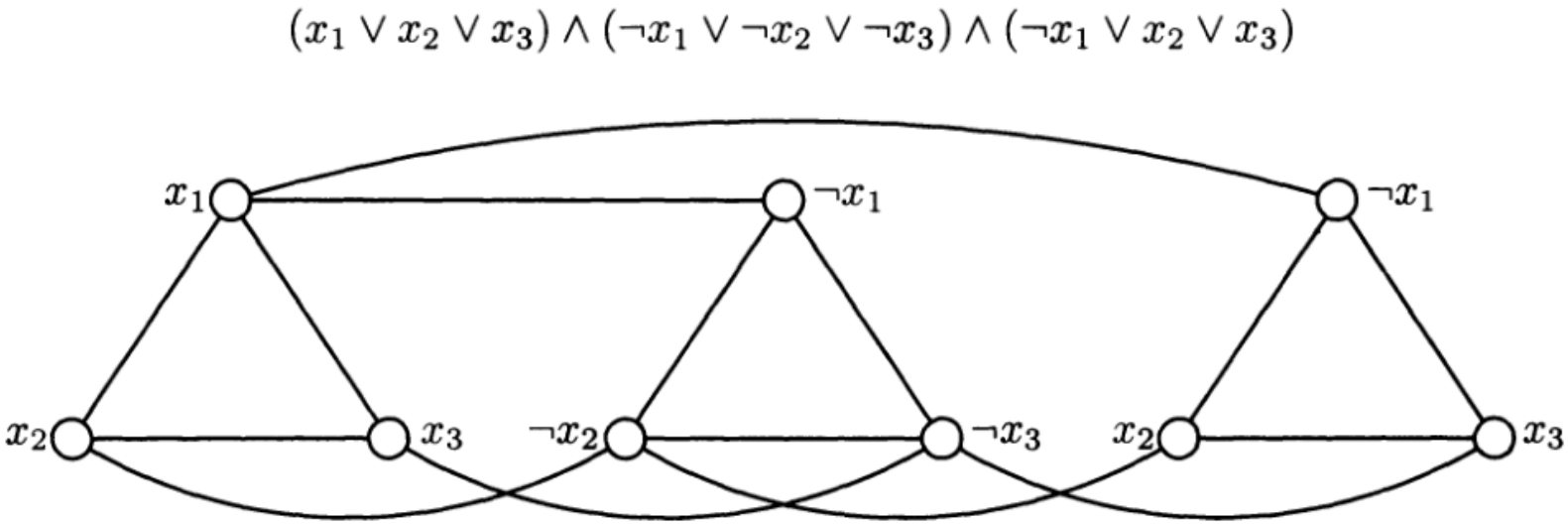


Figure 9-2. Reduction to INDEPENDENT SET.

Independent Set è **NP**-completo

Appartenenza a **NP**

1. Generare nondeterministicamente una soluzione candidata $I \subseteq V$ (costo $O(|V|)$) (funge da certificato)
   - per ogni nodo decidere se sta in I o no
2. Scandire I verificando che non contenga nodi collegati (costo $O(|V|^2)$)
3. Verificare se $|I| \geq K$
4. Costo totale: $O(|V|^2)$

QED

Independent Set è **NP**-completo

Appartenenza a **NP**

1. Generare nondeterministicamente una soluzione candidata $I \subseteq V$ (costo $O(|V|)$) (funge da certificato)
   - per ogni nodo decidere se sta in I o no
2. Scandire I verificando che non contenga nodi collegati (costo $O(|V|^2)$)
3. Verificare se $|I| \geq K$
4. Costo totale: $O(|V|^2)$

QED

Independent Set è **NP**-completo

Appartenenza a **NP**

1. Generare nondeterministicamente una soluzione candidata $I \subseteq V$ (costo $O(|V|)$) (funge da certificato)
   - per ogni nodo decidere se sta in I o no
2. Scandire I verificando che non contenga nodi collegati (costo $O(|V|^2)$)
3. Verificare se $|I| \geq K$
4. Costo totale: $O(|V|^2)$

QED

Independent Set è **NP**-completo

Appartenenza a **NP**

1. Generare nondeterministicamente una soluzione candidata $I \subseteq V$ (costo $O(|V|)$) (funge da certificato)
   - per ogni nodo decidere se sta in I o no
2. Scandire I verificando che non contenga nodi collegati (costo $O(|V|^2)$)
3. Verificare se $|I| \geq K$
4. Costo totale: $O(|V|^2)$

QED

Clique

Definizione

Dati $G = (V, E)$ e $K \in \mathbb{N}$, esiste un *clique* con $\geq K$ nodi?
Cioè un $C \subseteq V$ i cui nodi sono collegati direttamente da archi (fortemente connesso)

- Nel complemento di G (i cui archi sono quelli che non esistono in G) i clique diventano independent sets (e viceversa)
  - nodi collegati in G sse non collegati nel complemento
- Ne segue immediatamente che Clique è **NP**-completo
  - riduzioni banali, invertendo 0 e 1 nelle matrici di adiacenza

Clique

Definizione

Dati $G = (V, E)$ e $K \in \mathbb{N}$, esiste un *clique* con $\geq K$ nodi?
Cioè un $C \subseteq V$ i cui nodi sono collegati direttamente da archi (fortemente connesso)

- Nel complemento di G (i cui archi sono quelli che non esistono in G) i clique diventano independent sets (e viceversa)
  - nodi collegati in G sse non collegati nel complemento
- Ne segue immediatamente che Clique è **NP**-completo
  - riduzioni banali, invertendo 0 e 1 nelle matrici di adiacenza

Clique

Definizione

Dati $G = (V, E)$ e $K \in \mathbb{N}$, esiste un *clique* con $\geq K$ nodi?
Cioè un $C \subseteq V$ i cui nodi sono collegati direttamente da archi (fortemente connesso)

- Nel complemento di G (i cui archi sono quelli che non esistono in G) i clique diventano independent sets (e viceversa)
  - nodi collegati in G sse non collegati nel complemento
- Ne segue immediatamente che Clique è **NP**-completo
  - riduzioni banali, invertendo 0 e 1 nelle matrici di adiacenza

Node Cover

Definizione

Dati $G = (V, E)$ e $K \in \mathbb{N}$,
esiste un $C \subseteq V$ con $|C| \leq K$
tale che ogni arco ha un estremo in C?

- Nota: I è independent set per G sse $V \setminus I$ è un node cover
  - "Ogni arco ha almeno un estremo fuori da I" $\Leftrightarrow$ "Ogni arco ha almeno un estremo in $V \setminus I$"
- Di nuovo, la **NP**-completezza di Node Cover segue immediatamente con riduzioni banali

Node Cover

Definizione

Dati $G = (V, E)$ e $K \in \mathbb{N}$,
esiste un $C \subseteq V$ con $|C| \leq K$
tale che ogni arco ha un estremo in C?

- Nota: I è independent set per G sse $V \setminus I$ è un node cover
  - "Ogni arco ha almeno un estremo fuori da I" $\Leftrightarrow$ "Ogni arco ha almeno un estremo in $V \setminus I$"
- Di nuovo, la **NP**-completezza di Node Cover segue immediatamente con riduzioni banali

Node Cover

Definizione

Dati $G = (V, E)$ e $K \in \mathbb{N}$,
esiste un $C \subseteq V$ con $|C| \leq K$
tale che ogni arco ha un estremo in C?

- Nota: I è independent set per G sse $V \setminus I$ è un node cover
  - "Ogni arco ha almeno un estremo fuori da I" $\Leftrightarrow$ "Ogni arco ha almeno un estremo in $V \setminus I$"
- Di nuovo, la **NP**-completezza di Node Cover segue immediatamente con riduzioni banali

Hamilton Path è **NP**-completo

Ricordiamo la definizione

Definizione

- le istanze sono grafi
- la risposta è "yes" sse esiste un *ciclo Hamiltoniano* (che tocca ogni nodo una e una sola volta)
- Formalmente: una permutazione π dei nodi $1, 2, \ldots, n$ tale che per ogni $i = 1, \ldots, n-1$, $(\pi(i), \pi(i+1))$ è un arco di G

- L'appartenenza a **NP** è semplice: generare nondeterministicamente una permutazione dei vertici e controllare che tra quelli adiacenti esista un arco
- La hardness richiede una riduzione complicata da 3-SAT (trovate cenni sul libro)

Hamilton Path è **NP**-completo

Ricordiamo la definizione

Definizione

- le istanze sono grafi
- la risposta è "yes" sse esiste un *ciclo Hamiltoniano* (che tocca ogni nodo una e una sola volta)
- Formalmente: una permutazione π dei nodi $1, 2, \ldots, n$ tale che per ogni $i = 1, \ldots, n-1$, $(\pi(i), \pi(i+1))$ è un arco di G

- L'appartenenza a **NP** è semplice: generare nondeterministicamente una permutazione dei vertici e controllare che tra quelli adiacenti esista un arco
- La hardness richiede una riduzione complicata da 3-SAT (trovate cenni sul libro)

Hamilton Path è **NP**-completo

Ricordiamo la definizione

Definizione

- le istanze sono grafi
- la risposta è "yes" sse esiste un *ciclo Hamiltoniano* (che tocca ogni nodo una e una sola volta)
- Formalmente: una permutazione π dei nodi $1, 2, \ldots, n$ tale che per ogni $i = 1, \ldots, n-1$, $(\pi(i), \pi(i+1))$ è un arco di G

- L'appartenenza a **NP** è semplice: generare nondeterministicamente una permutazione dei vertici e controllare che tra quelli adiacenti esista un arco
- La hardness richiede una riduzione complicata da 3-SAT (trovate cenni sul libro)

Prova che TSP (D) è **NP**-completo

Hardness: riduzione da Hamilton Path

Unico problema: Una istanza di TSP è come un grafo completo

- Per ogni vertice dell'istanza di Hamilton Path una città
- Per ogni arco originale $[i, j]$, porre $d_{ij} = d_{ji} = 1$
- Se $[i, j]$ non esiste, porre $d_{ij} = d_{ji} = 2$
- Scegliere come budget $B = n$ ($n =$ numero nodi)

Proposizione (correttezza della riduzione)

Esiste un Hamilton Path sse c'è un tour di costo B

- È facile dimostrare che questa è effettivamente una riduzione (spazio logaritmico)

3-COLORING è **NP**-completo

Definizione k-COLORING

Dato un grafo $G = (V, E)$, si possono colorare i suoi vertici con k colori in modo che vertici adiacenti abbiano colori diversi?

- Esempio: carta geografica. Nodi = nazioni, archi = confini in comune.
- Per $k = 2$ è in **P** (vedere Problem 1.4.5 nel Papadimitriou)
- Per $k = 3$ è **NP**-completo

3-COLORING è **NP**-completo

Definizione k-COLORING

Dato un grafo $G = (V, E)$, si possono colorare i suoi vertici con k colori in modo che vertici adiacenti abbiano colori diversi?

- Esempio: carta geografica. Nodi = nazioni, archi = confini in comune.
- Per $k = 2$ è in **P** (vedere Problem 1.4.5 nel Papadimitriou)
- Per $k = 3$ è **NP**-completo

3-COLORING è **NP**-completo

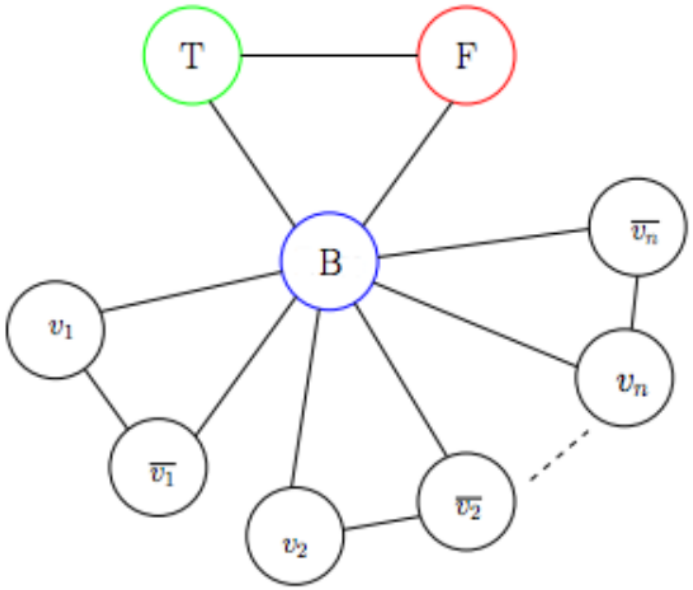
Dimostrazione - appartenenza

- Per dimostrare l'appartenenza a **NP**:
  1. Generare nondeterministicamente un colore per ogni nodo (memorizzare su nastro ausiliario)
  2. Controllare per ogni arco che i suoi estremi abbiano colori diversi
- Costo totale: $O(n^3)$

3-COLORING è **NP**-completo

Dimostrazione - hardness – I

- Riduzione da 3-SAT (diversamente da Papadimitriou)
- Useremo 2 gadgets. Il primo codifica i valori di verità e un truth assignment

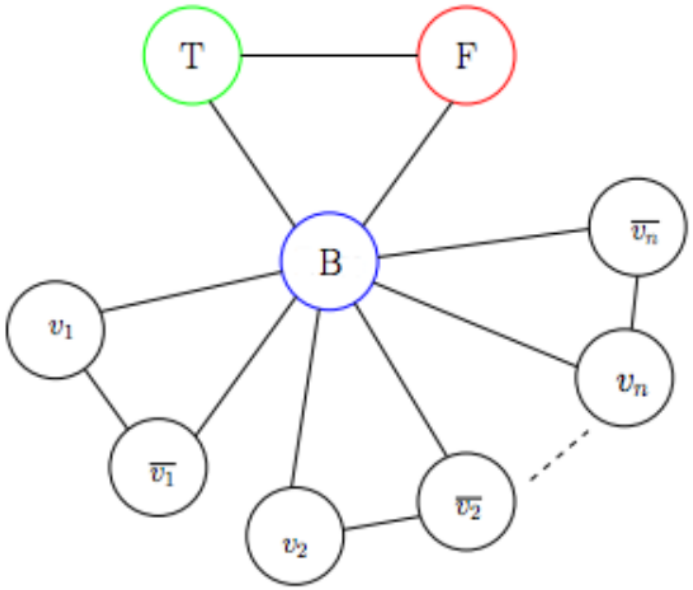


- Ogni proposizione v_i viene colorata come T o F
- Ogni letterale negativo $\bar{v}_i = \neg v_i$ viene colorato in modo opposto

3-COLORING è **NP**-completo

Dimostrazione - hardness – I

- Riduzione da 3-SAT (diversamente da Papadimitriou)
- Useremo 2 gadgets. Il primo codifica i valori di verità e un truth assignment

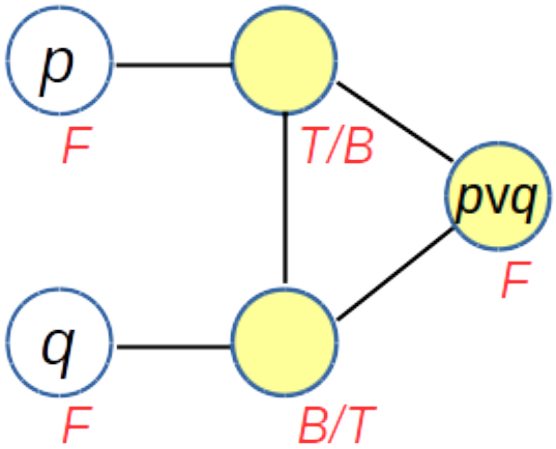


- Ogni proposizione v_i viene colorata come T o F
- Ogni letterale negativo $\bar{v}_i = \neg v_i$ viene colorato in modo opposto

3-COLORING è **NP**-completo

Dimostrazione - hardness – II

- Il 2^o gadget codifica l'OR

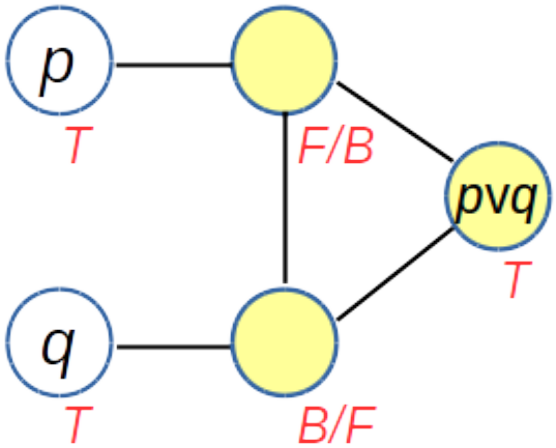


- Se p, q, F colorati nello stesso modo, $p \vee q$ è colorato come F
- Altrimenti $p \vee q$ *può* sempre essere colorato come T

3-COLORING è **NP**-completo

Dimostrazione - hardness – II

- Il 2^o gadget codifica l'OR

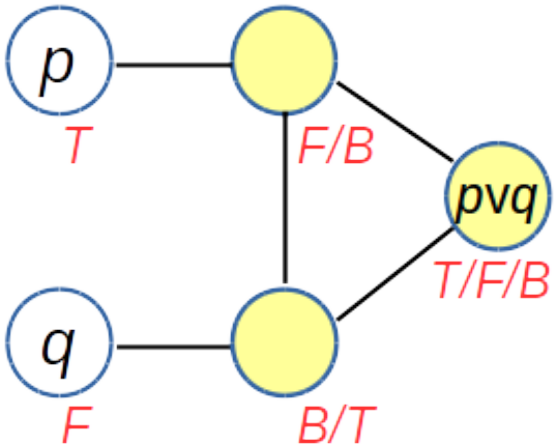


- Se p, q, F colorati nello stesso modo, $p \vee q$ è colorato come F
- Altrimenti $p \vee q$ *può* sempre essere colorato come T

3-COLORING è **NP**-completo

Dimostrazione - hardness – II

- Il 2^o gadget codifica l'OR

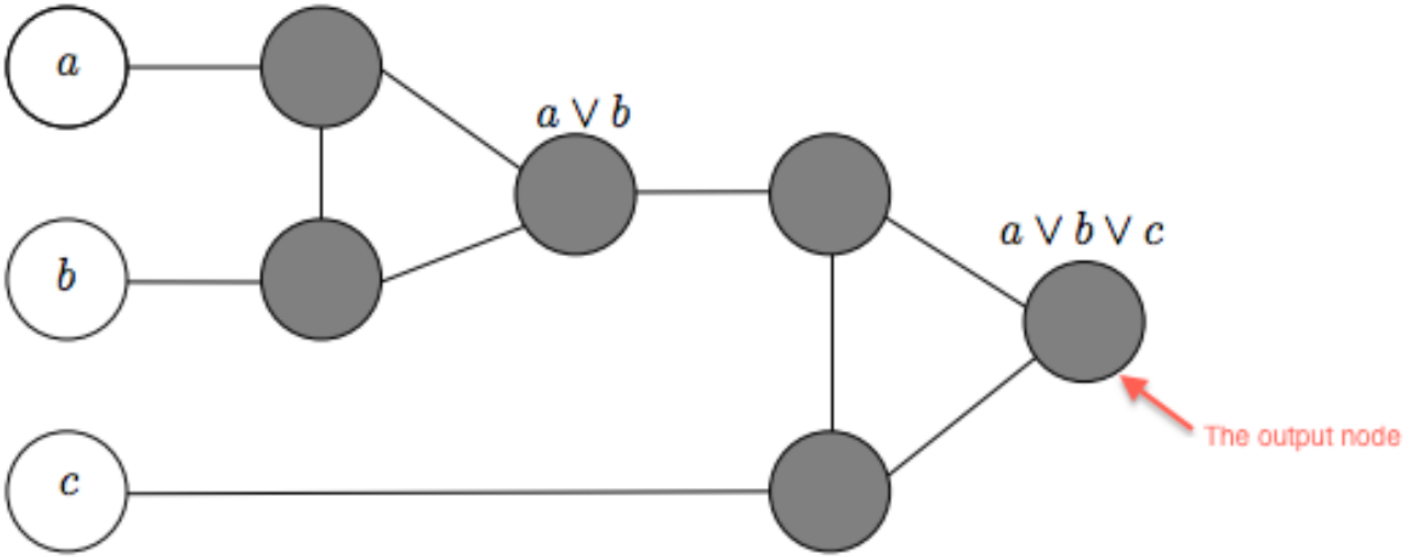


- Se p, q, F colorati nello stesso modo, $p \vee q$ è colorato come F
- Altrimenti $p \vee q$ *può* sempre essere colorato come T

3-COLORING è **NP**-completo

Dimostrazione - hardness – III

- Gli OR sono collegati in cascata per rappresentare le 3-clausole

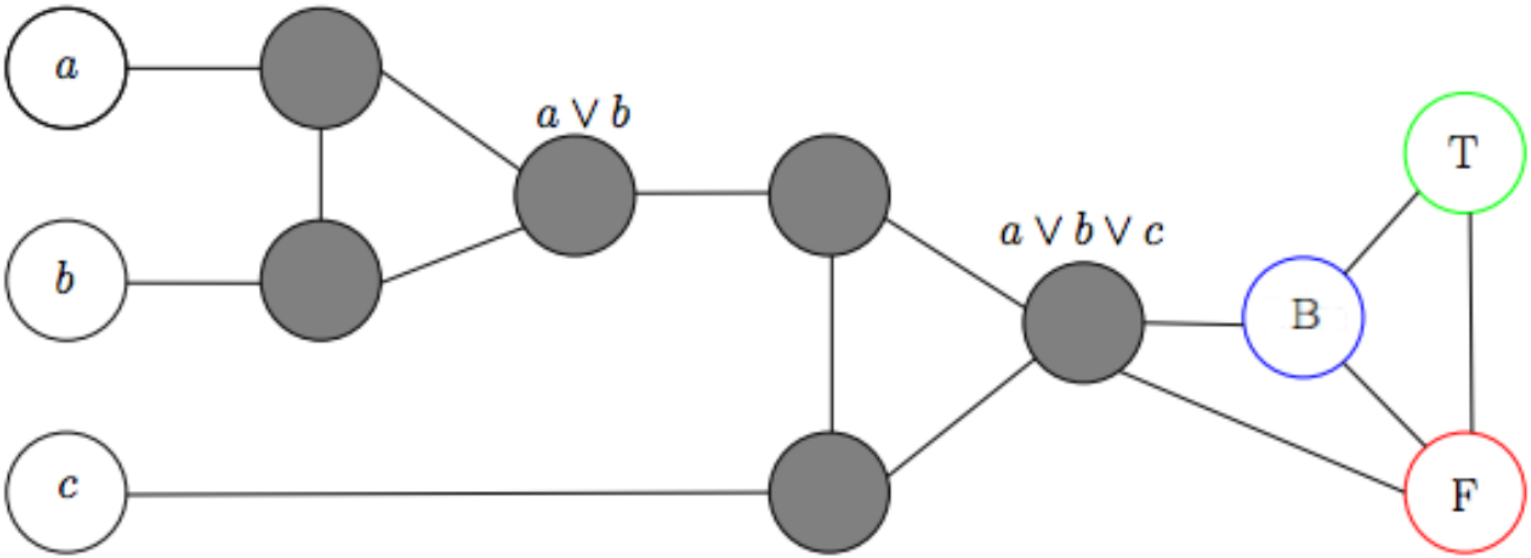


- Se a, b, c, F colorati nello stesso modo, $a \vee b \vee c$ è colorato come F
- Altrimenti $a \vee b \vee c$ *può* sempre essere colorato come T

3-COLORING è **NP**-completo

Dimostrazione - hardness – IV

- Per garantire che le clausole siano soddisfatte, colleghiamo gli output delle clausole così



- Se a, b, c, F colorati nello stesso modo, $a \vee b \vee c$ è colorato come F

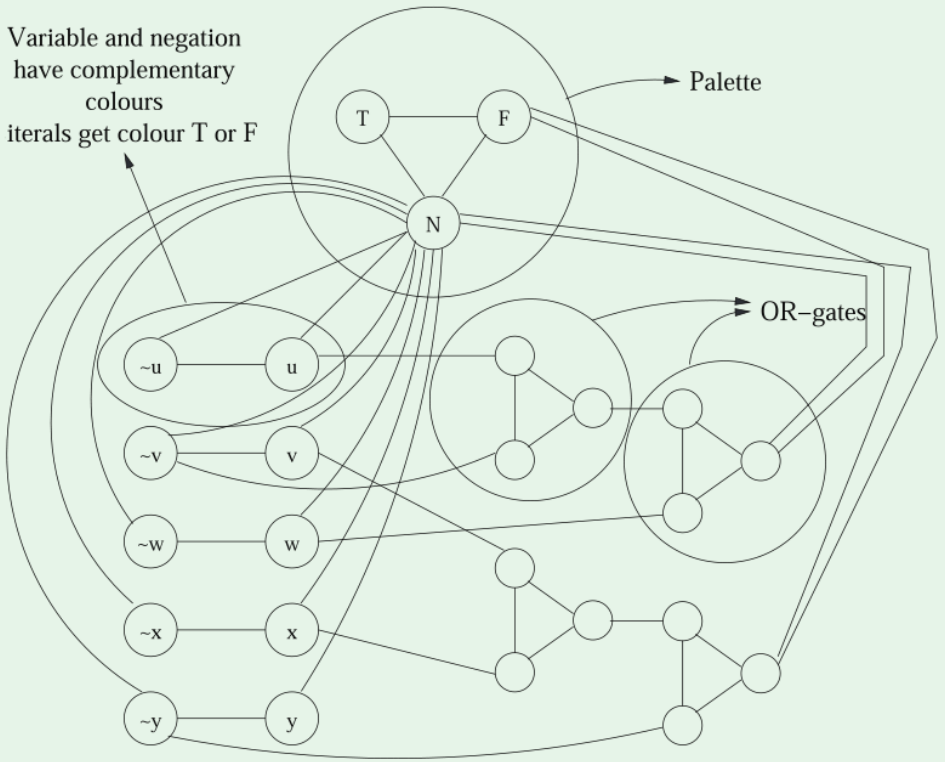
$\Rightarrow$ ora è impossibile! O a o b o c colorato come T

3-COLORING è **NP**-completo

Dimostrazione - hardness – V

- Esempio completo

$$\varphi = (u \vee \neg v \vee w) \wedge (v \vee x \vee \neg y)$$



3-COLORING è **NP**-completo

Dimostrazione - hardness – VI

- Correttezza della riduzione (cenni):
  - Dato un truth assignment si ottiene 3-coloring colorando come T tutti i nodi corrispondenti a proposizioni vere e OR veri
  - Viceversa, dato 3 coloring, ottenere truth assignment sulla base dei colori delle proposizioni
- Si può vedere che la riduzione richiede solo spazio logaritmico
  - (per generare il grafo bastano gli *indici* delle proposizioni)

QED

3-COLORING è **NP**-completo

Dimostrazione - hardness – VI

- Correttezza della riduzione (cenni):
  - Dato un truth assignment si ottiene 3-coloring colorando come T tutti i nodi corrispondenti a proposizioni vere e OR veri
  - Viceversa, dato 3 coloring, ottenere truth assignment sulla base dei colori delle proposizioni
- Si può vedere che la riduzione richiede solo spazio logaritmico
  - (per generare il grafo bastano gli *indici* delle proposizioni)

QED

Esercizio

- Quanto è difficile k-COLORING per $k > 3$?
- Tanti colori a disposizione, forse è più facile?
- Mostrare che rimane NP-completo
  - suggerimento per la riduzione: adattare quella per $k = 3$ aggiungendo $k - 3$ nodi
  - fare in modo che ogni colore dal quarto in poi possa essere usato per un solo nodo

Esercizio

- Quanto è difficile k-Coloring per $k > 3$?
- Tanti colori a disposizione, forse è più facile?
- Mostrare che rimane NP-completo
  - suggerimento per la riduzione: adattare quella per $k = 3$ aggiungendo $k - 3$ nodi
  - fare in modo che ogni colore dal quarto in poi possa essere usato per un solo nodo

Esercizio

- Quanto è difficile k-COLORING per $k > 3$?
- Tanti colori a disposizione, forse è più facile?
- Mostrare che rimane **NP**-completo
  - suggerimento per la riduzione: adattare quella per $k = 3$ aggiungendo $k - 3$ nodi
  - fare in modo che ogni colore dal quarto in poi possa essere usato per un solo nodo

Problemi su insiemi

Tripartite Matching

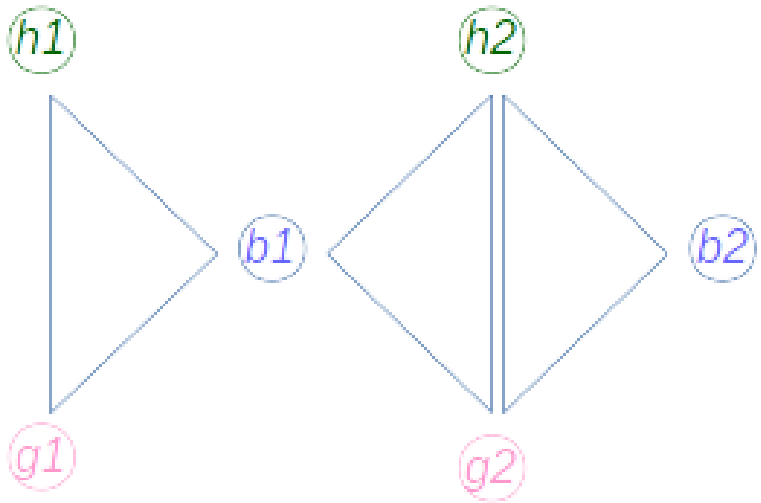
Definizione

Dati tre insiemi B, G, H (boys, girls, homes) con n elementi ciascuno
e una relazione ternaria $T \subseteq B \times G \times H$
trovare n triple in T che non hanno elementi in comune

Tripartite Matching

Esempio

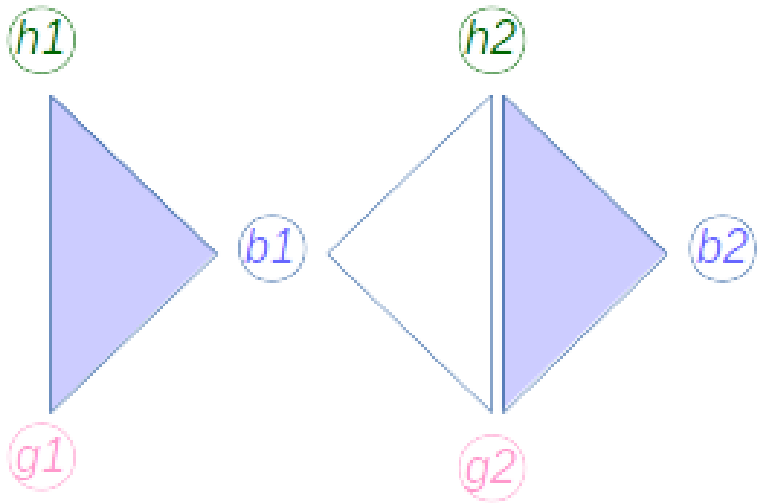
- Istanza: $B = \{b_1, b_2\}$, $G = \{g_1, g_2\}$, $H = \{h_1, h_2\}$
- Triple rappresentate con triangoli: $T = \{\langle b_1, g_1, h_1\rangle, \langle b_1, g_2, h_2\rangle, \langle b_2, g_2, h_2\rangle\}$
- $\langle b_1, g_1, h_1\rangle$ e $\langle b_2, g_2, h_2\rangle$ formano un matching: le triple sono 2 e non hanno elementi in comune
- Se scegliessi $\langle b_1, g_2, h_2\rangle$ non riuscirei a completare il matching



Tripartite Matching

Esempio

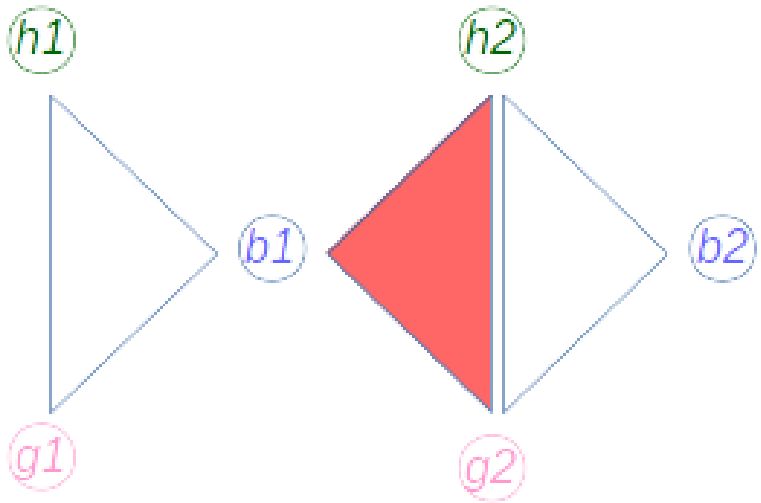
- Istanza: $B = \{b_1, b_2\}$, $G = \{g_1, g_2\}$, $H = \{h_1, h_2\}$
- Triple rappresentate con triangoli: $T = \{\langle b_1, g_1, h_1\rangle, \langle b_1, g_2, h_2\rangle, \langle b_2, g_2, h_2\rangle\}$
- $\langle b_1, g_1, h_1\rangle$ e $\langle b_2, g_2, h_2\rangle$ formano un matching: le triple sono 2 e non hanno elementi in comune
- Se scegliessi $\langle b_1, g_2, h_2\rangle$ non riuscirei a completare il matching



Tripartite Matching

Esempio

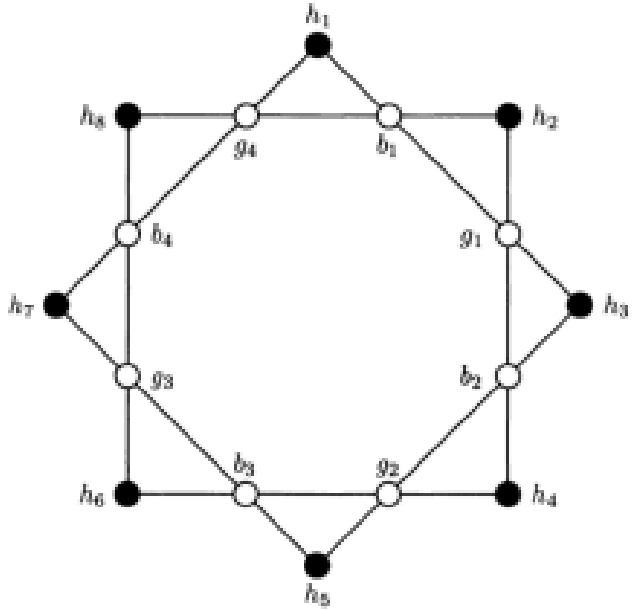
- Istanza: $B = \{b_1, b_2\}$, $G = \{g_1, g_2\}$, $H = \{h_1, h_2\}$
- Triple rappresentate con triangoli: $T = \{\langle b_1, g_1, h_1\rangle, \langle b_1, g_2, h_2\rangle, \langle b_2, g_2, h_2\rangle\}$
- $\langle b_1, g_1, h_1\rangle$ e $\langle b_2, g_2, h_2\rangle$ formano un matching: le triple sono 2 e non hanno elementi in comune
- Se scegliessi $\langle b_1, g_2, h_2\rangle$ non riuscirei a completare il matching



Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

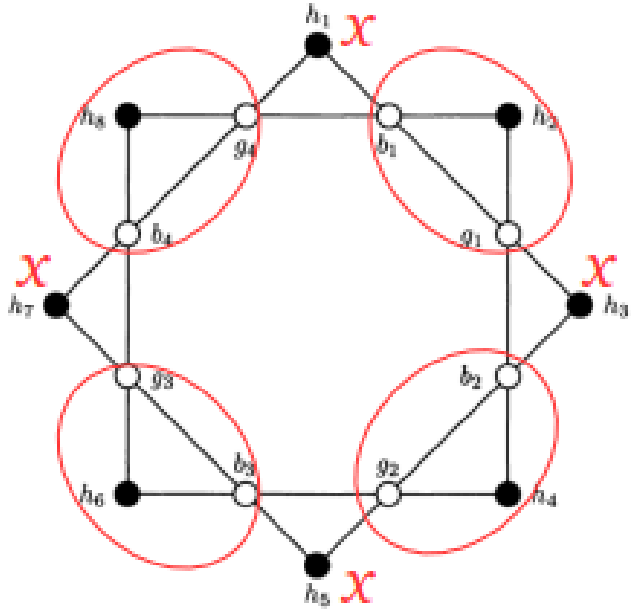
- Il gadget per generare le scelte $T(x) = true / T(x) = false$:
- Supponiamo per ora che ϕ contenga 4 occorrenze di x e 4 di $\neg x$



Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

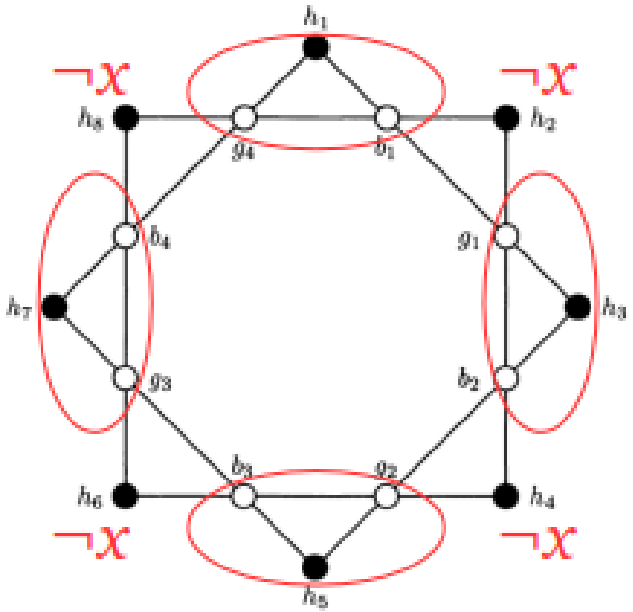
- Il gadget per generare le scelte $T(x) = true / T(x) = false$:
- Supponiamo per ora che ϕ contenga 4 occorrenze di x e 4 di $\neg x$



Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

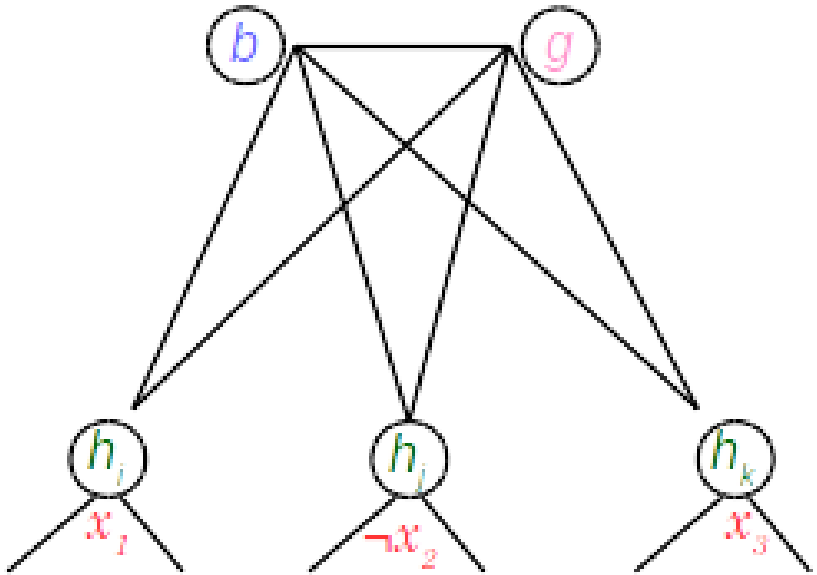
- Il gadget per generare le scelte $T(x) = true / T(x) = false$:
- Supponiamo per ora che ϕ contenga 4 occorrenze di x e 4 di $\neg x$



Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

- Per ogni 3-clausola, una ulteriore coppia b, g
- Le loro case: tre nodi h dei gadget corrispondenti ai 3 letterali
- Esempio: se la clausola è $x_1 \vee \neg x_2 \vee x_3$:



Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

- Problema: in generale ci sono *troppe case*
  - perchè il numero di occorrenze di x e $\neg x$ non è sempre uguale, alcune case sono "inutilizzate"
  - perchè per ogni clausola ci sono 3 case ma solo 2,5 ragazzi (e ragazze): uno per la clausola e 0,5 per ogni casa
- Soluzione:
  - Sia $\ell = |H| - |B|$ (equivalentemente $\ell = |H| - |G|$)
  - Introduciamo ℓ nuove coppie b, g, ciascuna collegata a tutte le case
  - In questo modo, qualunque soluzione che "soddisfa le clausole" può essere sicuramente completata per arrivare a un matching

Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

- Problema: in generale ci sono *troppe case*
  - perchè il numero di occorrenze di x e $\neg x$ non è sempre uguale, alcune case sono "inutilizzate"
  - perchè per ogni clausola ci sono 3 case ma solo 2,5 ragazzi (e ragazze): uno per la clausola e 0,5 per ogni casa
- Soluzione:
  - Sia $\ell = |H| - |B|$ (equivalentemente $\ell = |H| - |G|$)
  - Introduciamo ℓ nuove coppie b, g, ciascuna collegata a tutte le case
  - In questo modo, qualunque soluzione che "soddisfa le clausole" può essere sicuramente completata per arrivare a un matching

Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

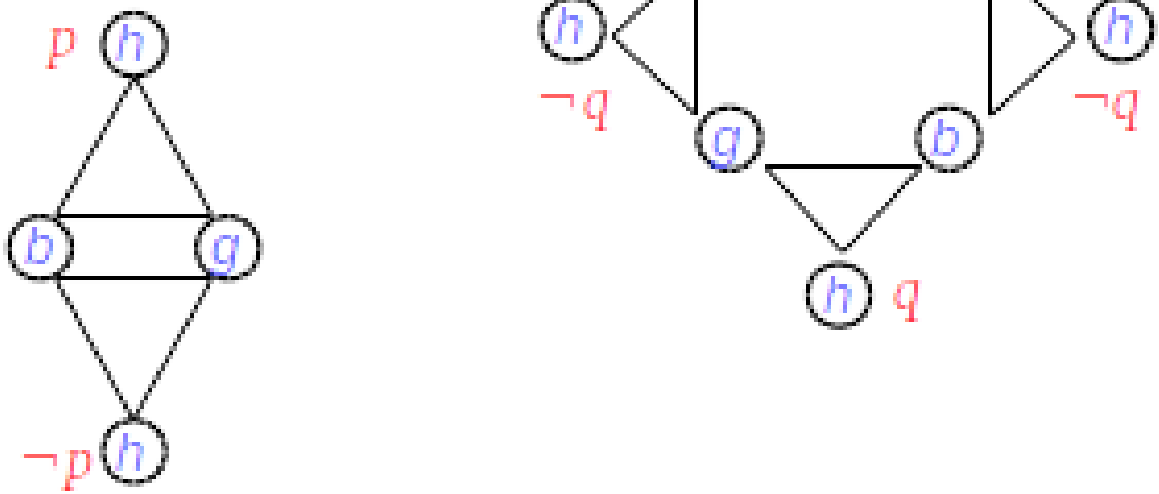
- Problema: in generale ci sono *troppe case*
  - perchè il numero di occorrenze di x e $\neg x$ non è sempre uguale, alcune case sono "inutilizzate"
  - perchè per ogni clausola ci sono 3 case ma solo 2,5 ragazzi (e ragazze): uno per la clausola e 0,5 per ogni casa
- Soluzione:
  - Sia $\ell = |H| - |B|$ (equivalentemente $\ell = |H| - |G|$)
  - Introduciamo ℓ nuove coppie b, g, ciascuna collegata a tutte le case
  - In questo modo, qualunque soluzione che "soddisfa le clausole" può essere sicuramente completata per arrivare a un matching

Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

- Esempio completo: riduzione di $\phi = (p \vee q \vee q) \wedge (\neg p \vee \neg q \vee \neg q)$

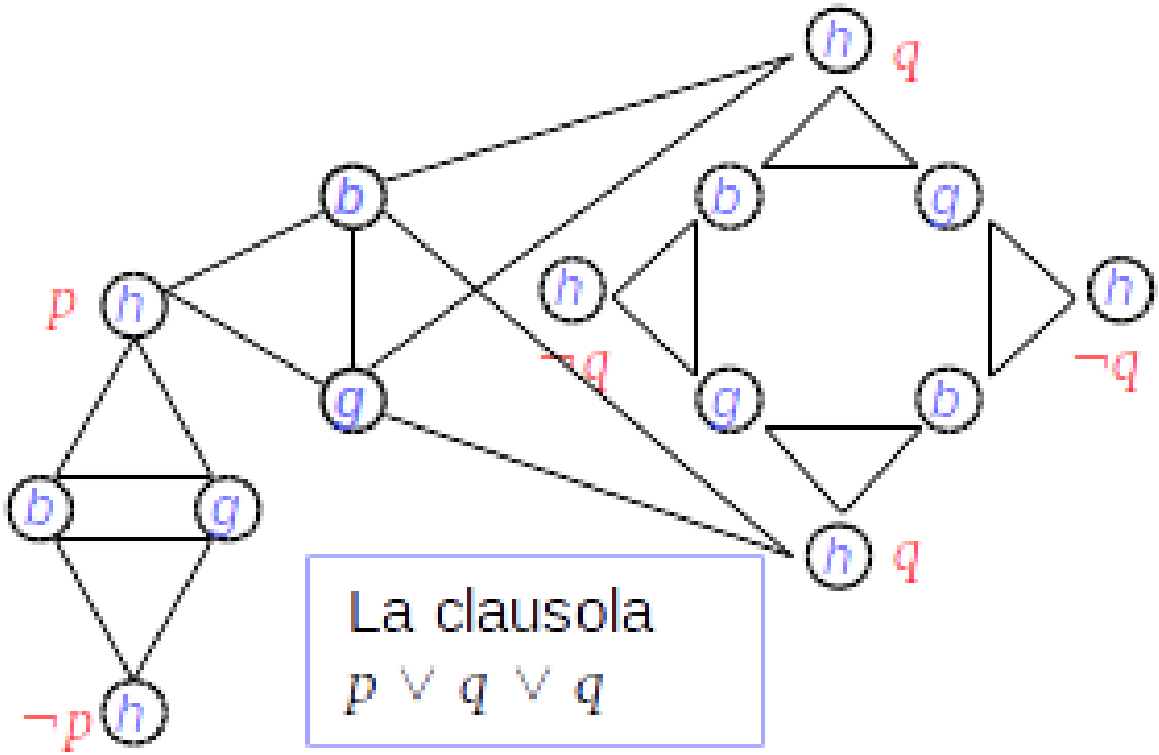
I gadgets per p e q



Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

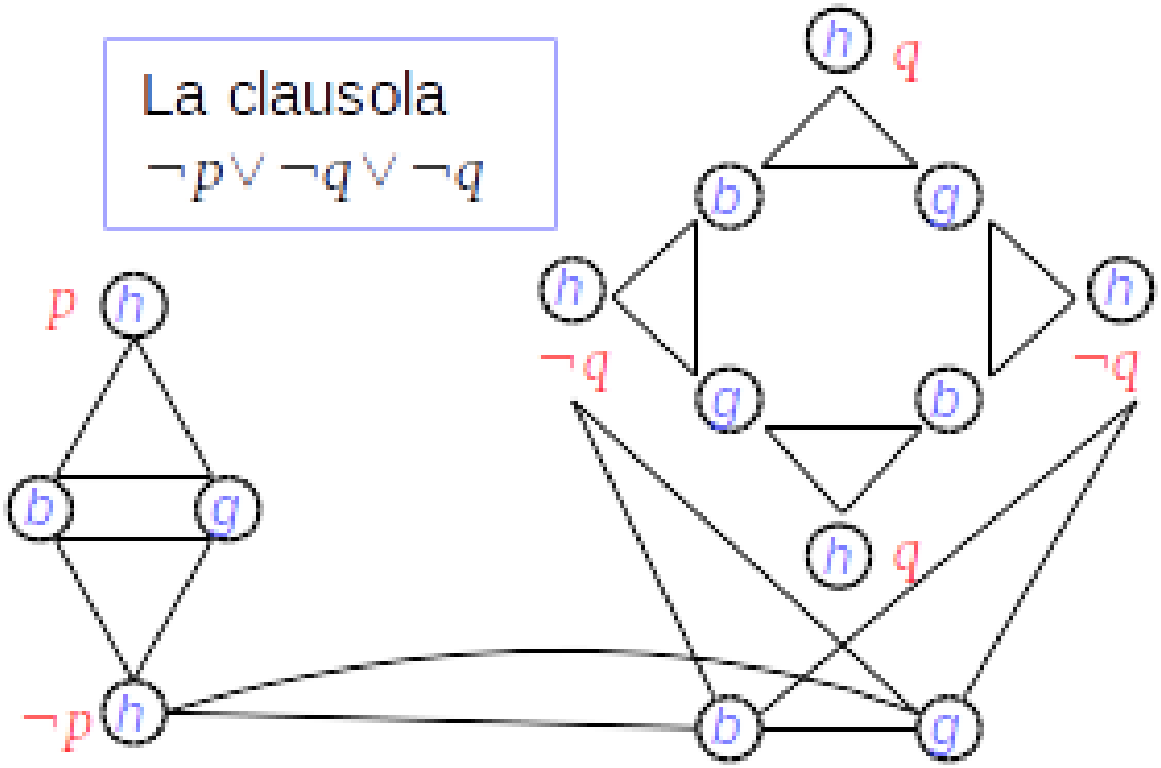
- Esempio completo: riduzione di $\phi = (p \vee q \vee q) \wedge (\neg p \vee \neg q \vee \neg q)$



Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

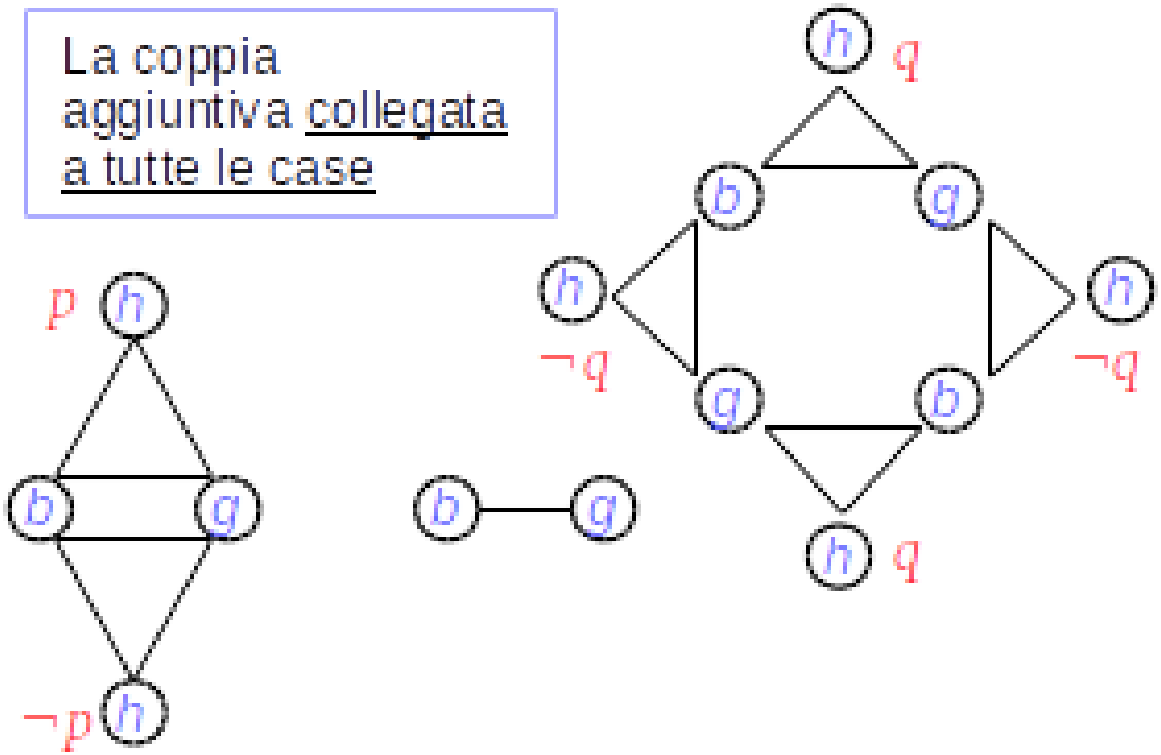
- Esempio completo: riduzione di $\phi = (p \vee q \vee q) \wedge (\neg p \vee \neg q \vee \neg q)$



Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

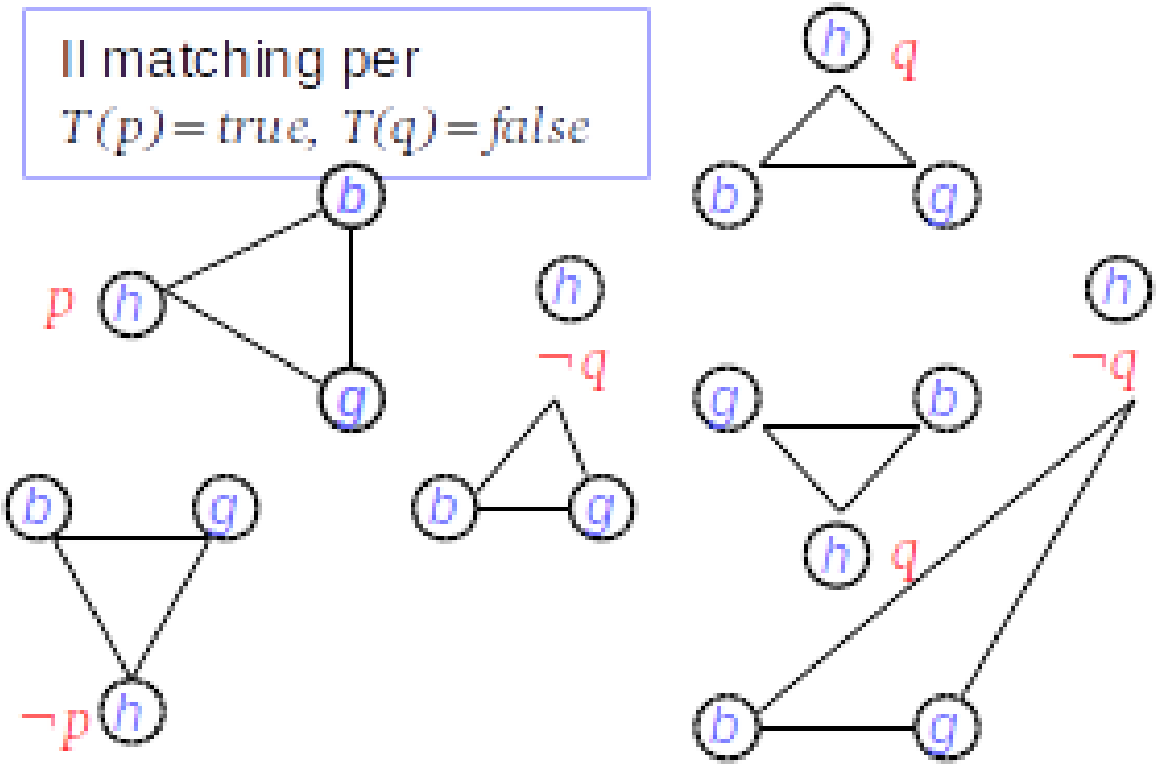
- Esempio completo: riduzione di $\phi = (p \vee q \vee q) \wedge (\neg p \vee \neg q \vee \neg q)$



Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

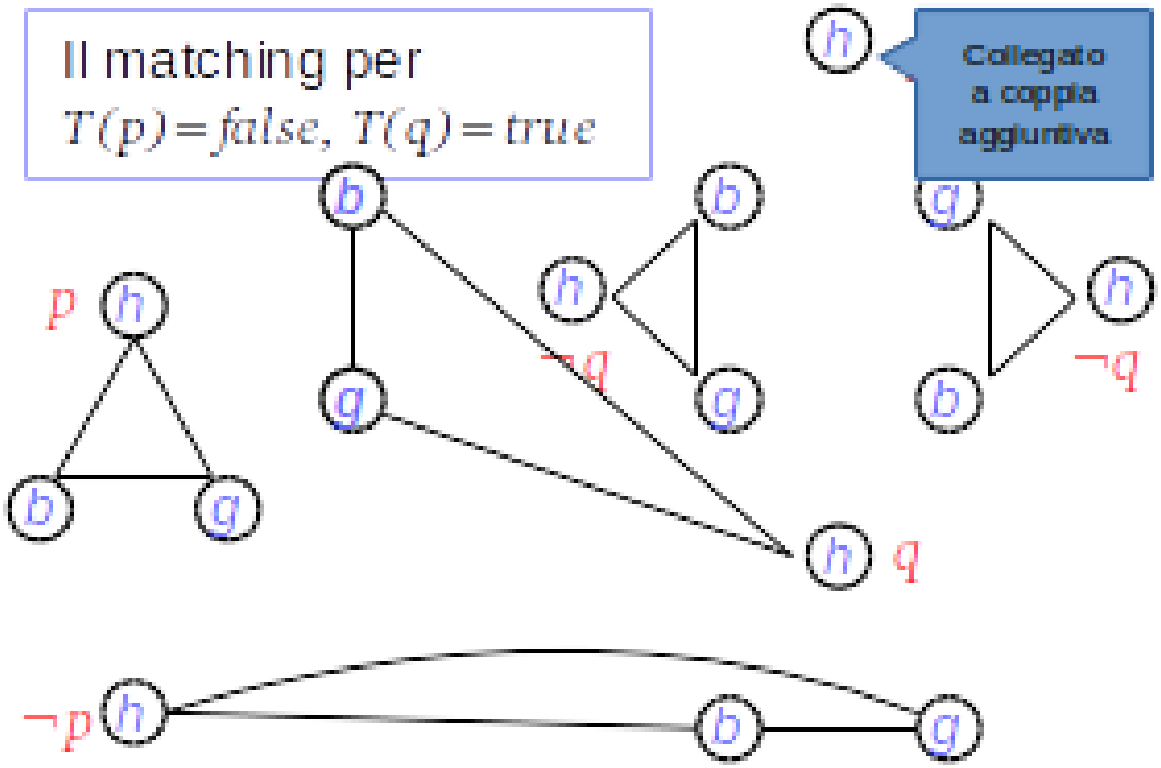
- Esempio completo: riduzione di $\phi = (p \vee q \vee q) \wedge (\neg p \vee \neg q \vee \neg q)$



Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

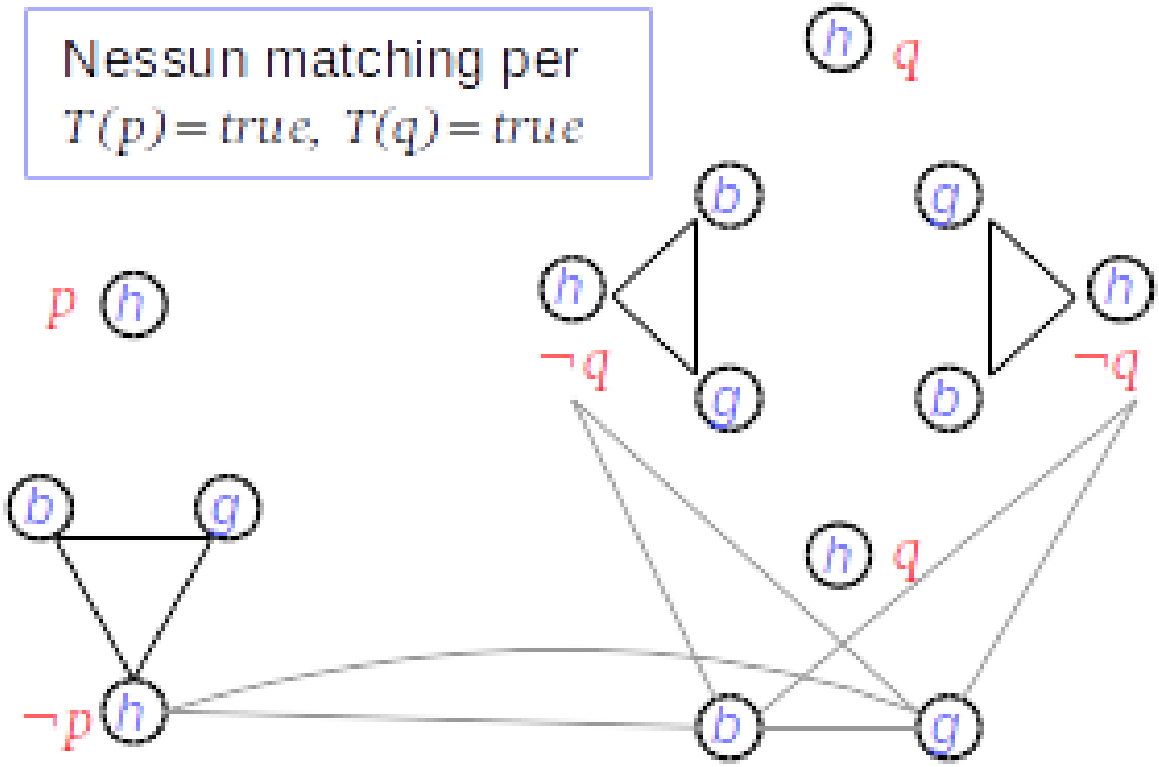
- Esempio completo: riduzione di $\phi = (p \vee q \vee q) \wedge (\neg p \vee \neg q \vee \neg q)$



Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

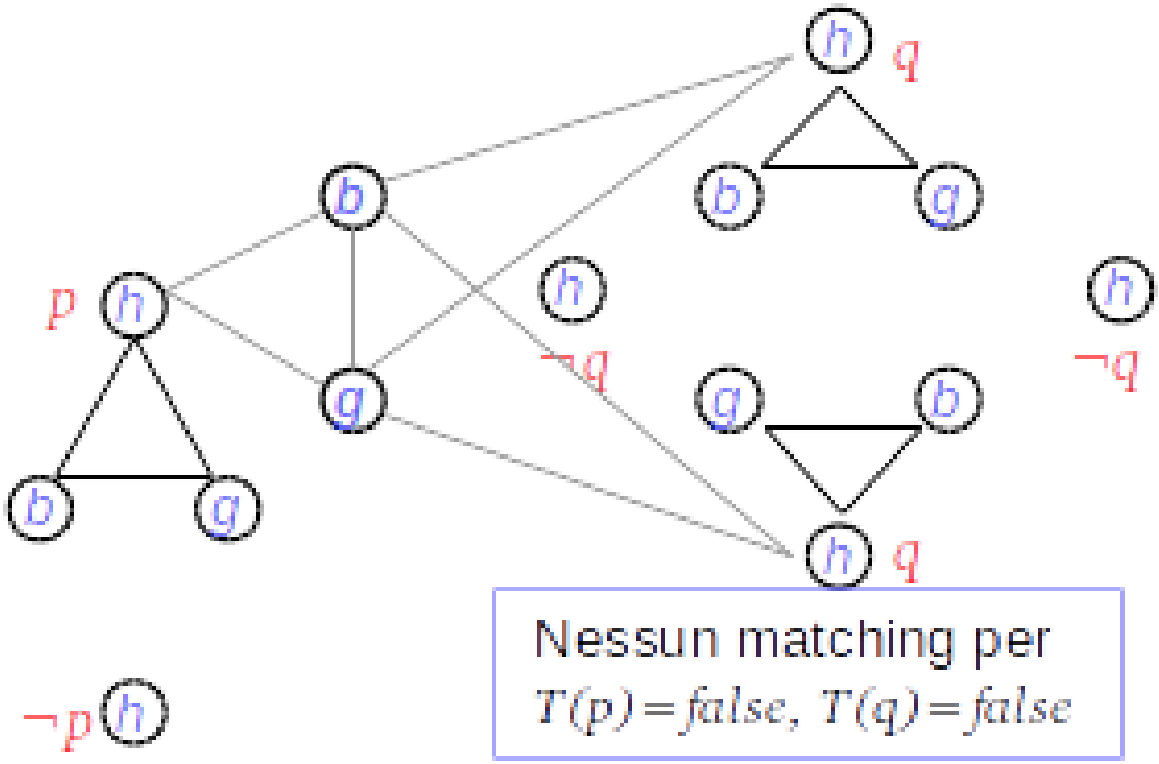
- Esempio completo: riduzione di $\phi = (p \vee q \vee q) \wedge (\neg p \vee \neg q \vee \neg q)$



Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

- Esempio completo: riduzione di $\phi = (p \vee q \vee q) \wedge (\neg p \vee \neg q \vee \neg q)$



Tripartite Matching è **NP**-completo

Riduzione 3-SAT→Tripartite Matching

- Si può dimostrare che la riduzione è corretta
- e che lo spazio per costruire i gadget si riduce agli indici di ragazzi, ragazze e case
  - che sono proporzionali al numero di letterali nelle clausole
  - quindi $O(\log n)$

1/2 QED (hardness)

Tripartite Matching è **NP**-completo

Appartenenza a **NP**

- Algoritmo nondeterministico:
  1. Generare nondeterministicamente il candidato a matching (sottinsieme delle triple T)
     - scandire l'input e per ogni tripla scegliere se tenerla (copiarla su nastro ausiliario) o scartarla ($O(n)$)
  2. Controllare se il candidato è un matching
     - scandire l'input e per ogni elemento (boy,girl,house) verificare che compaia una volta sola sul nastro ausiliario ($O(n^2)$)

QED

- Qui il witness è il matching.
  - La relazione *istanze* $\times$ *witness* è polynomially balanced perchè il witness è una sottostringa dell'input
  - Il punto 2 mostra che la relazione è polynomially decidable

Tripartite Matching è **NP**-completo

Appartenenza a **NP**

- Algoritmo nondeterministico:
  1. Generare nondeterministicamente il candidato a matching (sottinsieme delle triple T)
     - scandire l'input e per ogni tripla scegliere se tenerla (copiarla su nastro ausiliario) o scartarla ($O(n)$)
  2. Controllare se il candidato è un matching
     - scandire l'input e per ogni elemento (boy,girl,house) verificare che compaia una volta sola sul nastro ausiliario ($O(n^2)$)

  QED
- Qui il witness è il matching.
  - La relazione *istanze* $\times$ *witness* è polynomially balanced perchè il witness è una sottostringa dell'input
  - Il punto 2 mostra che la relazione è polynomially decidable

Problemi che generalizzano Tripartite Matching

Set Covering e simili

Definizione: Set Covering

Dati un insieme finito U, una famiglia $F = \{S_1, ..., S_n\}$ di sottinsiemi di U, e un intero B,
esistono B elementi di F la cui unione è U?
(versione decisionale di problema di minimizzazione)

Definizione: Exact Cover by 3-sets

Set Covering in cui $|U| = 3m$, $|S_i| = 3$ $(i = 1, .., n)$.
Domanda: esistono m elementi di F (disgiunti) la cui unione è U?

- Nota: Exact Cover by 3-sets è un caso particolare di Set Covering in cui $|U| = 3m$, $|S_i| = 3$ $(i = 1, .., n)$ e $B = m$
- Se Exact Cover by 3-sets $\in$ NP $\Rightarrow$ Set Covering $\in$ NP

Set Covering e simili

Definizione: Set Covering

Dati un insieme finito U, una famiglia $F = \{S_1, ..., S_n\}$ di sottinsiemi di U, e un intero B,
esistono B elementi di F la cui unione è U?
(versione decisionale di problema di minimizzazione)

Definizione: Exact Cover by 3-sets

Set Covering in cui $|U| = 3m$, $|S_i| = 3$ $(i = 1, .., n)$.
Domanda: esistono m elementi di F (disgiunti) la cui unione è U?

- Nota: Exact Cover by 3-sets è un caso particolare di Set Covering in cui $|U| = 3m$, $|S_i| = 3$ $(i = 1, .., n)$ e $B = m$
- Se Exact Cover by 3-sets $\in$ NP $\Rightarrow$ Set Covering $\in$ NP

Set Covering e simili

Definizione: Set Covering

Dati un insieme finito U, una famiglia $F = \{S_1, ..., S_n\}$ di sottinsiemi di U, e un intero B,
esistono B elementi di F la cui unione è U?
(versione decisionale di problema di minimizzazione)

Definizione: Exact Cover by 3-sets

Set Covering in cui $|U| = 3m$, $|S_i| = 3$ $(i = 1, .., n)$.
Domanda: esistono m elementi di F (disgiunti) la cui unione è U?

- Nota: Exact Cover by 3-sets è un caso particolare di Set Covering in cui $|U| = 3m$, $|S_i| = 3$ $(i = 1, .., n)$ e $B = m$
- Se Exact Cover by 3-sets $\in$ **NP** $\Rightarrow$ Set Covering $\in$ **NP**

Set Covering e simili

Definizione: Set Covering

Dati un insieme finito U, una famiglia $F = \{S_1, ..., S_n\}$ di sottinsiemi di U, e un intero B,
esistono B elementi di F la cui unione è U?
(versione decisionale di problema di minimizzazione)

Definizione: Exact Cover by 3-sets

Set Covering in cui $|U| = 3m$, $|S_i| = 3$ $(i = 1, .., n)$.
Domanda: esistono m elementi di F (disgiunti) la cui unione è U?

- Nota: Exact Cover by 3-sets è un caso particolare di Set Covering in cui $|U| = 3m$, $|S_i| = 3$ $(i = 1, .., n)$ e $B = m$
- Se Exact Cover by 3-sets $\in$ **NP** $\Rightarrow$ Set Covering $\in$ **NP**

Exact Cover by 3-sets è **NP**-completo

- Hardness: banale riduzione da Tripartite Matching
  - che è un caso particolare di Exact Cover by 3-sets
  - porre $U = B \cup G \cup H$
  - e $F = \{\{b, g, h\} \mid \langle b, g, h\rangle \in T\}$
- Membership
  1. scegli nondeterministicamente gli m insiemi
  2. verifica che l'unione copra U
  3. facile vedere che il tempo è polinomiale

Exact Cover by 3-sets è **NP**-completo

- Hardness: banale riduzione da Tripartite Matching
  - che è un caso particolare di Exact Cover by 3-sets
  - porre $U = B \cup G \cup H$
  - e $F = \{\{b, g, h\} \mid \langle b, g, h\rangle \in T\}$
- Membership
  1. scegli nondeterministicamente gli m insiemi
  2. verifica che l'unione copra U
  3. facile vedere che il tempo è polinomiale

Set Covering è **NP**-completo

- La hardness segue subito da quella di Exact Cover by 3-sets (vedi slides precedenti)
- Mostrare l'appartenenza a **NP** per esercizio

Set Packing è **NP**-completo

Definizione: Set Packing

Dati i soliti U, $F = \{S_1, .., S_n\}$, e un intero K,
dire se esistono K insiemi disgiunti in F

- È sempre una generalizzazione di Tripartite Matching:
  - porre $U = B \cup G \cup H$,
  - $F = \{\{b, g, h\} \mid \langle b, g, h\rangle \in T\}$
  - e $K = |B| = |G| = |H|$
  - se gli S_i sono disgiunti, tutti i ragazzi, le ragazze e le case sono "impegnati"
- Appartenenza a **NP** per esercizio

Set Packing è **NP**-completo

Definizione: Set Packing

Dati i soliti U, $F = \{S_1, .., S_n\}$, e un intero K,
dire se esistono K insiemi disgiunti in F

- È sempre una generalizzazione di Tripartite Matching:
  - porre $U = B \cup G \cup H$,
  - $F = \{\{b, g, h\} \mid \langle b, g, h\rangle \in T\}$
  - e $K = |B| = |G| = |H|$
  - se gli S_i sono disgiunti, tutti i ragazzi, le ragazze e le case sono "impegnati"
- Appartenenza a **NP** per esercizio

Integer programming è **NP**-completo

Definizione di Integer programming

Dato un sistema di disequazioni lineari a coefficienti interi in n variabili, dire se ha una soluzione intera.

- Hardness: riduzione banale da Set Covering
  - matrice binaria A di dim. $|U| \times |F|$:
    $A_{i,j} = 1$ sse l'i-esimo elemento di U appartiene a S_j
  - variabili binarie x_j $(j = 1, .., |F|)$:
    $x_j = 1$ sse S_j appartiene alla soluzione
  - istanza di Integer programming risultante:
    - $Ax \geq \mathbf{1}$ (vincolo di copertura)
    - $\sum_{j=1}^{n} x_j \leq B$ (vincolo di budget)
    - $0 \leq x_i \leq 1$
- Appartenenza a **NP**: non banale (e non la vediamo)

Integer programming è NP-completo

Definizione di Integer programming

Dato un sistema di disequazioni lineari a coefficienti interi in n variabili, dire se ha una soluzione intera.

- Hardness: riduzione banale da Set Covering
  - matrice binaria A di dim. $|U| \times |F|$:
    $A_{i,j} = 1$ sse l'i-esimo elemento di U appartiene a S_j
  - variabili binarie x_j $(j = 1, .., |F|)$:
    $x_j = 1$ sse S_j appartiene alla soluzione
  - istanza di Integer programming risultante:
    - $Ax \geq \mathbf{1}$ (vincolo di copertura)
    - $\sum_{j=1}^{n} x_j \leq B$ (vincolo di budget)
    - $0 \leq x_i \leq 1$
- Appartenenza a NP: non banale (e non la vediamo)

Integer programming è **NP**-completo

Definizione di Integer programming

Dato un sistema di disequazioni lineari a coefficienti interi in n variabili, dire se ha una soluzione intera.

- Hardness: riduzione banale da Set Covering
  - matrice binaria A di dim. $|U| \times |F|$:
    $A_{i,j} = 1$ sse l'i-esimo elemento di U appartiene a S_j
  - variabili binarie x_j $(j = 1, .., |F|)$:
    $x_j = 1$ sse S_j appartiene alla soluzione
  - istanza di Integer programming risultante:
    - $Ax \geq \mathbf{1}$ (vincolo di copertura)
    - $\sum_{j=1}^{n} x_j \leq B$ (vincolo di budget)
    - $0 \leq x_i \leq 1$
- Appartenenza a **NP**: non banale (e non la vediamo)

Esercizio e commenti

- Dare prova alternativa di hardness riducendo SAT
  - le variabili x_i corrispondono alle proposizioni di ϕ

- È interessante notare che se togliamo il vincolo che la soluzione sia intera, allora il problema risultante (Linear programming) è in **P**!

Esercizio e commenti

- Dare prova alternativa di hardness riducendo SAT
  - le variabili x_i corrispondono alle proposizioni di ϕ

- È interessante notare che se togliamo il vincolo che la soluzione sia intera, allora il problema risultante (Linear programming) è in **P**!

Knapsack

Il problema della zaino

Definizione di Knapsack

Dati n oggetti $i = 1, ..., n$
ciascuno con un valore v_i e un peso (weight) w_i
e dati un limite di peso $W \in \mathbb{N}$ e un intero $K \in \mathbb{N}$,
dire se esiste $S \subseteq \{1, ..., n\}$ tale che

- $\sum_{i \in S} w_i \leq W$ (il limite di peso è rispettato)
- $\sum_{i \in S} v_i \geq K$ (il valore raggiunge la soglia specificata)

- È la versione decisionale del problema di ottimizzazione:
  massimizzare il valore rispettando il limite di peso

Knapsack

Il problema della zaino

- È un caso particolare di Integer programming:
- Siano $\mathbf{w} = (w_1, ..., w_n)$ e $\mathbf{v} = (v_1, ..., v_n)$
- Il sistema è
  - $0 \leq x_i \leq 1$ $(i = 1, ..., n)$ $[x_i = 1$ sse $i \in S]$
  - $\mathbf{w}x \leq W$
  - $\mathbf{v}x \geq K$

KNAPSACK

Il problema della zaino

- C'è un caso particolare di KNAPSACK che è già **NP** completo:
- $w_i = v_i$ $(i = 1, ..., n)$ e $K = W$
  - quindi $K \leq \sum_{i \in S} v_i = \sum_{i \in S} w_i \leq W = K$
- Il problema diventa:
  - dati gli interi $w_1, ..., w_n$
  - dire se esiste $S \subseteq \{w_1, ..., w_n\}$ tale che
  - $\sum \{w_i \mid w_i \in S\} = K$
- ovvero: esiste un sottinsieme di $\{w_1, ..., w_n\}$ che somma esattamente a K?

KNAPSACK

Il problema della zaino

- C'è un caso particolare di KNAPSACK che è già **NP** completo:
- $w_i = v_i$ $(i = 1, ..., n)$ e $K = W$
  - quindi $K \leq \sum_{i \in S} v_i = \sum_{i \in S} w_i \leq W = K$
- Il problema diventa:
  - dati gli interi $w_1, ..., w_n$
  - dire se esiste $S \subseteq \{w_1, ..., w_n\}$ tale che
  - $\sum \{w_i \mid w_i \in S\} = K$
- ovvero: esiste un sottinsieme di $\{w_1, ..., w_n\}$ che somma esattamente a K?

KNAPSACK è **NP**-completo

Preliminari: rappresentare unioni di insiemi con somme di interi

- Rappresentiamo unioni di insiemi con somme di interi. Esempio:
  - $U = \{1, 2, 3, 4\}$
  - $S_1 = \{1\}$, $S_2 = \{2\}$, $S_3 = \{3\}$, $S_4 = \{3, 4\}$, $S_5 = \{1, 3\}$
- Rappresentiamo gli S_i con vettori di bit (che sono anche interi)
  - $S_1 = 0001 = 1 = w_1$, $S_2 = 0010 = 2 = w_2$, $S_3 = 0100 = 8 = w_3$, $S_4 = 1100 = 12 = w_4$, $S_5 = 0101 = 5 = w_5$
- Rappresentazione di $S_i \cup S_j$ con $w_i + w_j$

KNAPSACK è **NP**-completo

Preliminari: rappresentare unioni di insiemi con somme di interi

- Rappresentiamo unioni di insiemi con somme di interi. Esempio:
  - $U = \{1, 2, 3, 4\}$
  - $S_1 = \{1\}$, $S_2 = \{2\}$, $S_3 = \{3\}$, $S_4 = \{3, 4\}$, $S_5 = \{1, 3\}$
- Rappresentiamo gli S_i con vettori di bit (che sono anche interi)
  - $S_1 = 0001 = 1 = w_1$, $S_2 = 0010 = 2 = w_2$,
    $S_3 = 0100 = 8 = w_3$, $S_4 = 1100 = 12 = w_4$,
    $S_5 = 0101 = 5 = w_5$
- Rappresentazione di $S_i \cup S_j$ con $w_i + w_j$

$$S_1 \cup S_2 = \{1, 2\} \qquad \begin{array}{rcccccc} S_1 & \rightsquigarrow & 0001 & = & 1 & = & w_1 \\ S_2 & \rightsquigarrow & 0010 & = & 2 & = & w_2 \\ \hline \text{somma} & = & 0011 & = & 3 & = & w_1 + w_2 \end{array}$$

KNAPSACK è **NP**-completo

Preliminari: rappresentare unioni di insiemi con somme di interi

- Rappresentiamo unioni di insiemi con somme di interi. Esempio:
  - $U = \{1, 2, 3, 4\}$
  - $S_1 = \{1\}$, $S_2 = \{2\}$, $S_3 = \{3\}$, $S_4 = \{3, 4\}$, $S_5 = \{1, 3\}$
- Rappresentiamo gli S_i con vettori di bit (che sono anche interi)
  - $S_1 = 0001 = 1 = w_1$, $S_2 = 0010 = 2 = w_2$,
    $S_3 = 0100 = 8 = w_3$, $S_4 = 1100 = 12 = w_4$,
    $S_5 = 0101 = 5 = w_5$
- Rappresentazione di $S_i \cup S_j$ con $w_i + w_j$

$S_1 \cup S_3 = \{1, 3\}$

S_1	$\rightsquigarrow$	0001	$=$	1	$=$	w_1
S_3	$\rightsquigarrow$	0100	$=$	4	$=$	w_3
somma	$=$	0101	$=$	5	$=$	$w_1 + w_3$

KNAPSACK è **NP**-completo

Preliminari: rappresentare unioni di insiemi con somme di interi

- Rappresentiamo unioni di insiemi con somme di interi. Esempio:
  - $U = \{1, 2, 3, 4\}$
  - $S_1 = \{1\}$, $S_2 = \{2\}$, $S_3 = \{3\}$, $S_4 = \{3, 4\}$, $S_5 = \{1, 3\}$
- Rappresentiamo gli S_i con vettori di bit (che sono anche interi)
  - $S_1 = 0001 = 1 = w_1$, $S_2 = 0010 = 2 = w_2$,
    $S_3 = 0100 = 8 = w_3$, $S_4 = 1100 = 12 = w_4$,
    $S_5 = 0101 = 5 = w_5$
- Rappresentazione di $S_i \cup S_j$ con $w_i + w_j$

$S_2 \cup S_4 = \{2, 3, 4\}$

S_2	$\rightsquigarrow$	0010	$=$	2	$=$	w_2
S_4	$\rightsquigarrow$	1100	$=$	12	$=$	w_4
somma	$=$	1110	$=$	14	$=$	$w_1 + w_4$

KNAPSACK è **NP**-completo

Preliminari: rappresentare unioni di insiemi con somme di interi

- Rappresentiamo unioni di insiemi con somme di interi. Esempio:
  - $U = \{1, 2, 3, 4\}$
  - $S_1 = \{1\}$, $S_2 = \{2\}$, $S_3 = \{3\}$, $S_4 = \{3, 4\}$, $S_5 = \{1, 3\}$
- Rappresentiamo gli S_i con vettori di bit (che sono anche interi)
  - $S_1 = 0001 = 1 = w_1$, $S_2 = 0010 = 2 = w_2$,
    $S_3 = 0100 = 8 = w_3$, $S_4 = 1100 = 12 = w_4$,
    $S_5 = 0101 = 5 = w_5$
- Rappresentazione di $S_i \cup S_j$ con $w_i + w_j$ ok, ma *senza riporti*...

$S_2 \cup S_4 = \{2, 3, 4\}$

S_2	$\rightsquigarrow$	0010	$=$	2	$=$	w_2
S_4	$\rightsquigarrow$	1100	$=$	12	$=$	w_4
somma	$=$	1110	$=$	14	$=$	$w_1 + w_4$

KNAPSACK è **NP**-completo

Preliminari: rappresentare unioni di insiemi con somme di interi

- Problema: in presenza di riporti non funziona più

$$S_1 \cup S_5 = \{1, 3\}$$

S_1	$\rightsquigarrow$	0001	$=$	1	$=$	w_1
S_5	$\rightsquigarrow$	0101	$=$	5	$=$	w_5
somma	$=$	0110	$=$	6	$\rightsquigarrow$	$\{2, 3\}$

- Bugfix: cambiare base per evitare i riporti
- Abbiamo 5 insiemi S_i. Anche se tutti contenessero 1, la prima colonna sommerebbe a 5 $\Rightarrow$ usando base 6 evitiamo il riporto

$$S_1 \cup S_5 = \{1, 3\}$$

S_1	$\rightsquigarrow$	0001	$=$	1	$=$	w_1
S_5	$\rightsquigarrow$	0101	$=$	37	$=$	w_5
somma	$=$	0102	$=$	38	$=$	$w_1 + w_5$

KNAPSACK è **NP**-completo

Preliminari: rappresentare unioni di insiemi con somme di interi

- Problema: in presenza di riporti non funziona più

$$S_1 \cup S_5 = \{1, 3\} \qquad \begin{array}{rcccccc} S_1 & \rightsquigarrow & 0001 & = & 1 & = & w_1 \\ S_5 & \rightsquigarrow & 0101 & = & 5 & = & w_5 \\ \hline \text{somma} & = & 0110 & = & 6 & \rightsquigarrow & \{2, 3\} \end{array}$$

- Bugfix: cambiare base per evitare i riporti
- Abbiamo 5 insiemi S_i. Anche se tutti contenessero 1, la prima colonna sommerebbe a 5 $\Rightarrow$ usando base 6 evitiamo il riporto

$$S_1 \cup S_5 = \{1, 3\} \qquad \begin{array}{rcccccc} S_1 & \rightsquigarrow & 0001 & = & 1 & = & w_1 \\ S_5 & \rightsquigarrow & 0101 & = & 37 & = & w_5 \\ \hline \text{somma} & = & 0102 & = & 38 & = & w_1 + w_5 \end{array}$$

KNAPSACK è **NP**-completo

Preliminari: rappresentare unioni di insiemi con somme di interi

- Interpretazione delle somme:
- Le posizioni delle cifre non zero indicano i membri dell'unione

$$S_1 \cup S_3 \cup S_5 = \{1, 3\}$$

$$\begin{array}{rcccccc} S_1 & \rightsquigarrow & 0001 & = & 1 & = & w_1 \\ S_3 & \rightsquigarrow & 0100 & = & 36 & = & w_3 \\ S_5 & \rightsquigarrow & 0101 & = & 37 & = & w_5 \\ \hline \text{somma} & = & 0202 & = & 74 & \rightsquigarrow & \{1, 3\} \end{array}$$

KNAPSACK è **NP**-completo

Preliminari: rappresentare unioni di insiemi con somme di interi

- Interpretazione delle somme:
- Le cifre sono tutte $\leq 1 \Leftrightarrow$ gli insiemi sono disgiunti

$$S_1 \cup S_3 \cup S_5 = \{1, 3\}$$

$$\begin{array}{rcccccc} S_1 & \rightsquigarrow & 0001 & = & 1 & = & w_1 \\ S_3 & \rightsquigarrow & 0100 & = & 36 & = & w_3 \\ S_5 & \rightsquigarrow & 0101 & = & 37 & = & w_5 \\ \hline \text{somma} & = & 0202 & = & 74 & \rightsquigarrow & \{1, 3\} \end{array}$$

KNAPSACK è **NP**-completo

Preliminari: rappresentare unioni di insiemi con somme di interi

- Interpretazione delle somme:
- Le cifre sono tutte $\leq 1 \Leftrightarrow$ gli insiemi sono disgiunti

$$S_1 \cup S_2 \cup S_3 = \{1,2,3\}$$

S_1	$\rightsquigarrow$	0001	$=$	1	$=$	w_1
S_2	$\rightsquigarrow$	0010	$=$	6	$=$	w_2
S_3	$\rightsquigarrow$	0100	$=$	36	$=$	w_3
somma	$=$	0111	$=$	43	$\rightsquigarrow$	$\{1,2,3\}$

KNAPSACK è **NP**-completo

Preliminari: rappresentare unioni di insiemi con somme di interi

- Interpretazione delle somme:
- Tutte le cifre sono 1 $\Leftrightarrow$ gli insiemi sono una partizione di U

$$S_1 \cup S_2 \cup S_4 = U$$

S_1	$\rightsquigarrow$	0001	$=$	1	$=$	w_1
S_2	$\rightsquigarrow$	0010	$=$	6	$=$	w_2
S_4	$\rightsquigarrow$	1100	$=$	252	$=$	w_4
somma	$=$	1111	$=$	259	$\rightsquigarrow$	$\{1, 2, 3, 4\}$

KNAPSACK è **NP**-completo

Parte I – Riduzione di EXACT COVER BY 3-SETS

- Prendiamo qualunque istanza di EXACT COVER BY 3-SETS
  - $U = \{1, 2, ..., 3m\}$, $F = \{S_1, ..., S_n\}$, $S_i \subseteq U$, $|S_i| = 3$
- Rappresentiamo gli S_i in base $(n+1)$, come nelle slide precedenti

$$w_i = \sum_{j \in S_i} (n+1)^{j-1}$$

- Le soluzioni sono insiemi che formano una partizione di U, quindi la somma, in base $(n+1)$, deve essere $\underbrace{11...1}_{n}$, cioè

$$K = \sum_{j=1}^{3m} (n+1)^{j-1}$$

- Chiaramente un exact cover esiste sse un sottinsieme di $\{w_1, ..., w_n\}$ somma esattamente a K (il caso particolare di KNAPSACK)

KNAPSACK è NP-completo

Parte I – Riduzione di EXACT COVER BY 3-SETS

- Prendiamo qualunque istanza di EXACT COVER BY 3-SETS
  - $U = \{1, 2, ..., 3m\}$, $F = \{S_1, ..., S_n\}$, $S_i \subseteq U$, $|S_i| = 3$
- Rappresentiamo gli S_i in base $(n+1)$, come nelle slide precedenti

$$w_i = \sum_{j \in S_i} (n+1)^{j-1}$$

- Le soluzioni sono insiemi che formano una partizione di U, quindi la somma, in base $(n+1)$, deve essere $\underbrace{11...1}_{n}$, cioè

$$K = \sum_{j=1}^{3m} (n+1)^{j-1}$$

- Chiaramente un exact cover esiste sse un sottinsieme di $\{w_1, ..., w_n\}$ somma esattamente a K (il caso particolare di KNAPSACK)

KNAPSACK è **NP**-completo

Parte I – Riduzione di EXACT COVER BY 3-SETS

- Prendiamo qualunque istanza di EXACT COVER BY 3-SETS
  - $U = \{1, 2, ..., 3m\}$, $F = \{S_1, ..., S_n\}$, $S_i \subseteq U$, $|S_i| = 3$
- Rappresentiamo gli S_i in base $(n+1)$, come nelle slide precedenti

$$w_i = \sum_{j \in S_i} (n+1)^{j-1}$$

- Le soluzioni sono insiemi che formano una partizione di U, quindi la somma, in base $(n+1)$, deve essere $\underbrace{11...1}_{n}$, cioè

$$K = \sum_{j=1}^{3m} (n+1)^{j-1}$$

- Chiaramente un exact cover esiste sse un sottinsieme di $\{w_1, ..., w_n\}$ somma esattamente a K (il caso particolare di KNAPSACK)

KNAPSACK è **NP**-completo

Parte I – Riduzione di EXACT COVER BY 3-SETS

- Prendiamo qualunque istanza di EXACT COVER BY 3-SETS
  - $U = \{1, 2, ..., 3m\}$, $F = \{S_1, ..., S_n\}$, $S_i \subseteq U$, $|S_i| = 3$
- Rappresentiamo gli S_i in base $(n+1)$, come nelle slide precedenti

$$w_i = \sum_{j \in S_i} (n+1)^{j-1}$$

- Le soluzioni sono insiemi che formano una partizione di U, quindi la somma, in base $(n+1)$, deve essere $\underbrace{11...1}_{n}$, cioè

$$K = \sum_{j=1}^{3m} (n+1)^{j-1}$$

- Chiaramente un exact cover esiste sse un sottinsieme di $\{w_1, ..., w_n\}$ somma esattamente a K (il caso particolare di KNAPSACK)

Knapsack è **NP**-completo

Parte I – Riduzione di Exact Cover by 3-sets

- Per dimostrare che la traduzione si può fare in spazio logaritmico occorre un algoritmo per l'esponenziazione in spazio log n
  - ripetere $ris = ris \cdot (n+1)$ per j volte genera numeri ris di lunghezza $O(n)$ perchè $j = O(n)$

Esponenziazione in spazio logaritmico

Proprietà 1

- Useremo un algoritmo ricorsivo
- Per mantenere lo stack di attivazione entro $O(\log n)$ dividiamo l'esponente per 2 ad ogni chiamata ricosrisiva

$$b^n = \begin{cases} \left(b^{n/2}\right)^2 & \text{se } n \text{ pari} \\ b \cdot b^{n-1} & \text{se } n \text{ dispari} \end{cases}$$

Esponenziazione in spazio logaritmico

Proprietà 2

- Poi dobbiamo elevare al quadrato in spazio logaritmico
- Problema generale: calcolare $r = a \cdot b$ in spazio logaritmico
  - $r = (r_{2n-2}...r_0)_2$, $a = (a_{n-1}...a_0)_2$, $b = (b_{n-1}...b_0)_2$
  - $a = (a_{n-1}2^{n-1} + ... + a_0 2^0)$
  - $b = (b_{n-1}2^{n-1} + ... + b_0 2^0)$
  - $r = (\sum_{j+k=2n-1} a_j a_k)2^{2n-2} + ... + (\sum_{j+k=0} a_j a_k)2^0$
- Nota: i termini di grado $\geq i+1$ non contribuiscono a r_i
- Tutti quelli di grado inferiore vi contribuiscono direttamente o attraverso i riporti

Esponenziazione in spazio logaritmico

Proprietà 2

- Calcolo di $r = a \cdot b$ in spazio logaritmico[1]
  - $r = (r_{2n-1}...r_0)_2$, $a = (a_{n-1}...a_0)_2$, $b = (b_{n-1}...b_0)_2$

$$
\begin{aligned}
h_i &= c_{i-1} + \sum_{j+k=i} a_j b_k \\
r_i &= h_i \mod 2 \\
c_i &= \lfloor h_i/2 \rfloor \quad (c_0 = 0)
\end{aligned}
$$

- Claim: $c_i \leq n$ e $h_i \leq 2n + 1$ (per induzione)

[1] https://en.wikipedia.org/wiki/Multiplication_algorithmSpace_complexity. Attenzione: mancano riferimenti

Esponenziazione in spazio logaritmico

Algoritmo

exp(b,n,i) (* base, esponente, bit prodotto*)
if n pari then
  $h = c_0 = 0$
  for $l = 0, .., i$
    $h = \lfloor h/2 \rfloor + \sum_{j+k=l} exp(b, n/2, j) \cdot exp(b, n/2, k)$
  return $h \mod 2$
else (n dispari)
  $h = c_0 = 0$
  for $l = 0, .., i$
    $h = \lfloor h/2 \rfloor + \sum_{j+k=l} b_j \cdot exp(b, n-1, k)$
  return $h \mod 2$
end

KNAPSACK è **NP**-completo

Parte II – Appartenenza a **NP**

- Solito algoritmo generate-and-test
  1. selezionare nondeterministicamente un sottinsieme degli oggetti
  2. controllare deterministicamente se la somma dei loro pesi è $\leq W$ e se la somma dei loro valori è $\geq K$

QED

Discussione semiformale:

Algoritmi pseudopolinomiali e **NP**-completezza "forte"

Risoluzione di KNAPSACK

Teorema

KNAPSACK $\in$ **TIME**(nW), dove n è il numero di oggetti e W il limite di peso

- **Prova:** Calcoliamo $V(w, i)$: il valore massimo che si può ottenere con i primi i oggetti arrivando *esattamente* al peso w
  1. porre $V(w, 0) = 0$ per tutti i $w \leq W$
  2. notare che $V(w, i+1) = \max\{V(w, i),\ v_{i+1} + V(w - w_{i+1}, i)\}$
- La risposta è "yes" sse la tabella V contiene un valore $\geq K$
- Poiché la tabella ha $(n+1)(W+1)$ celle, il tempo richiesto è $O(nW)$

QED

Risoluzione di KNAPSACK

Teorema

KNAPSACK $\in$ **TIME**(nW), dove n è il numero di oggetti e W il limite di peso

- **Prova**: Calcoliamo $V(w,i)$: il valore massimo che si può ottenere con i primi i oggetti arrivando *esattamente* al peso w
  1. porre $V(w,0) = 0$ per tutti i $w \leq W$
  2. notare che $V(w,i+1) = \max\{V(w,i),\ v_{i+1} + V(w - w_{i+1}, i)\}$
- La risposta è "yes" sse la tabella V contiene un valore $\geq K$
- Poichè la tabella ha $(n+1)(W+1)$ celle, il tempo richiesto è $O(nW)$

QED

Risoluzione di KNAPSACK

Teorema

KNAPSACK $\in$ **TIME**(nW), dove n è il numero di oggetti e W il limite di peso

- **Prova**: Calcoliamo $V(w,i)$: il valore massimo che si può ottenere con i primi i oggetti arrivando *esattamente* al peso w
  1. porre $V(w,0) = 0$ per tutti i $w \leq W$
  2. notare che $V(w,i+1) = \max\{V(w,i),\ v_{i+1} + V(w - w_{i+1}, i)\}$
- La risposta è "yes" sse la tabella V contiene un valore $\geq K$
- Poichè la tabella ha $(n+1)(W+1)$ celle, il tempo richiesto è $O(nW)$

QED

Risoluzione di KNAPSACK

Teorema

KNAPSACK $\in$ **TIME**(nW), dove n è il numero di oggetti e W il limite di peso

- **Prova**: Calcoliamo $V(w,i)$: il valore massimo che si può ottenere con i primi i oggetti arrivando *esattamente* al peso w
  1. porre $V(w,0) = 0$ per tutti i $w \leq W$
  2. notare che $V(w,i+1) = \max\{V(w,i),\ v_{i+1} + V(w - w_{i+1}, i)\}$
- La risposta è "yes" sse la tabella V contiene un valore $\geq K$
- Poichè la tabella ha $(n+1)(W+1)$ celle, il tempo richiesto è $O(nW)$

QED

L'algoritmo per KNAPSACK è pseudopolinomiale

- Bisogna distinguere tra (il valore) W e la sua *rappresentazione* $\underline{W}$: in notazione posizionale

$$W = O(2^{|\underline{W}|})$$

- Quindi l'algoritmo usa tempo esponenziale e la questione $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ resta aperta
- Se ci restringiamo a pesi "piccoli" ($|\underline{W}| = O(\log n)$), allora KNAPSACK ricade in **P**
  - ma naturalmente la riduzione da EXACT COVER BY 3-SETS usa pesi "grossi", con $|\underline{W}| = O(n)$
  - il *valore* di W è esponenziale nella dimensione dell'istanza di EXACT COVER BY 3-SETS

L'algoritmo per KNAPSACK è pseudopolinomiale

- Bisogna distinguere tra (il valore) W e la sua *rappresentazione* $\underline{W}$: in notazione posizionale

$$W = O(2^{|\underline{W}|})$$

- Quindi l'algoritmo usa tempo esponenziale e la questione $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ resta aperta
- Se ci restringiamo a pesi "piccoli" ($|\underline{W}| = O(\log n)$), allora KNAPSACK ricade in **P**
  - ma naturalmente la riduzione da EXACT COVER BY 3-SETS usa pesi "grossi", con $|\underline{W}| = O(n)$
  - il *valore* di W è esponenziale nella dimensione dell'istanza di EXACT COVER BY 3-SETS

Confronto con altri problemi **NP**-completi

- Le altre riduzioni generavano numeri piccoli (polinomiali nell'input) diversamente da quella per KNAPSACK che genera numeri esponenzialmente grandi:
- Nella riduzione per SAT, le proposizioni erano in numero polinomiale quindi la loro rappresentazione era logaritmica
- Nella riduzione per TSP(D), (da HAMILTON PATH) le distanze d_{ij} valevano 1 o 2 (rappresentazione costante)
- Nella riduzione per CLIQUE, (da INDEPENDENT SET), K resta lo stesso e $K \leq n$
- Nella riduzione per TRIPARTITE MATCHING, (da SAT), il numero di boys, girls, homes è proporzionale al numero di proposizioni, quindi i valori degli indici restano in $O(n)$ (rappresentazione logaritmica)

Confronto con altri problemi **NP**-completi

- Le altre riduzioni generavano numeri piccoli (polinomiali nell'input) diversamente da quella per KNAPSACK che genera numeri esponenzialmente grandi:
- Nella riduzione per SAT, le proposizioni erano in numero polinomiale quindi la loro rappresentazione era logaritmica
- Nella riduzione per TSP(D), (da HAMILTON PATH) le distanze d_{ij} valevano 1 o 2 (rappresentazione costante)
- Nella riduzione per CLIQUE, (da INDEPENDENT SET), K resta lo stesso e $K \leq n$
- Nella riduzione per TRIPARTITE MATCHING, (da SAT), il numero di boys, girls, homes è proporzionale al numero di proposizioni, quindi i valori degli indici restano in $O(n)$ (rappresentazione logaritmica)

Confronto con altri problemi **NP**-completi

- Le altre riduzioni generavano numeri piccoli (polinomiali nell'input) diversamente da quella per KNAPSACK che genera numeri esponenzialmente grandi:
- Nella riduzione per SAT, le proposizioni erano in numero polinomiale quindi la loro rappresentazione era logaritmica
- Nella riduzione per TSP(D), (da HAMILTON PATH) le distanze d_{ij} valevano 1 o 2 (rappresentazione costante)
- Nella riduzione per CLIQUE, (da INDEPENDENT SET), K resta lo stesso e $K \leq n$
- Nella riduzione per TRIPARTITE MATCHING, (da SAT), il numero di boys, girls, homes è proporzionale al numero di proposizioni, quindi i valori degli indici restano in $O(n)$ (rappresentazione logaritmica)

Confronto con altri problemi **NP**-completi

- Le altre riduzioni generavano numeri piccoli (polinomiali nell'input) diversamente da quella per KNAPSACK che genera numeri esponenzialmente grandi:
- Nella riduzione per SAT, le proposizioni erano in numero polinomiale quindi la loro rappresentazione era logaritmica
- Nella riduzione per TSP(D), (da HAMILTON PATH) le distanze d_{ij} valevano 1 o 2 (rappresentazione costante)
- Nella riduzione per CLIQUE, (da INDEPENDENT SET), K resta lo stesso e $K \leq n$
- Nella riduzione per TRIPARTITE MATCHING, (da SAT), il numero di boys, girls, homes è proporzionale al numero di proposizioni, quindi i valori degli indici restano in $O(n)$ (rappresentazione logaritmica)

Strong **NP**-completeness

- Problemi che restano **NP**-completi anche se i numeri/indici coinvolti sono "piccoli"
  - equivalentemente: se anche li scrivessimo in unario
  - ad es. tutti quelli della slide precedente
  - invece KNAPSACK ricadrebbe in **P**
- Se un problema "strongly **NP**-complete" avesse una soluzione pseudopolinomiale, allora sarebbe anche *effettivamente polinomiale* (conseguenza: $\mathbf{P} = \mathbf{NP}$)
- Quelli con algoritmi pseudopolinomiali si prestano ad approssimazioni "efficienti" arbitrariamente precise
  - illustrate in *Ottimizzazione combinatoria*

Strong **NP**-completeness

- Problemi che restano **NP**-completi anche se i numeri/indici coinvolti sono "piccoli"
  - equivalentemente: se anche li scrivessimo in unario
  - ad es. tutti quelli della slide precedente
  - invece KNAPSACK ricadrebbe in **P**
- Se un problema "strongly **NP**-complete" avesse una soluzione pseudopolinomiale, allora sarebbe anche *effettivamente polinomiale* (conseguenza: $\mathbf{P} = \mathbf{NP}$)
- Quelli con algoritmi pseudopolinomiali si prestano ad approssimazioni "efficienti" arbitrariamente precise
  - illustrate in *Ottimizzazione combinatoria*

Strong **NP**-completeness

- Problemi che restano **NP**-completi anche se i numeri/indici coinvolti sono "piccoli"
  - equivalentemente: se anche li scrivessimo in unario
  - ad es. tutti quelli della slide precedente
  - invece KNAPSACK ricadrebbe in **P**
- Se un problema "strongly **NP**-complete" avesse una soluzione pseudopolinomiale, allora sarebbe anche *effettivamente polinomiale* (conseguenza: $\mathbf{P} = \mathbf{NP}$)
- Quelli con algoritmi pseudopolinomiali si prestano ad approssimazioni "efficienti" arbitrariamente precise
  - illustrate in *Ottimizzazione combinatoria*

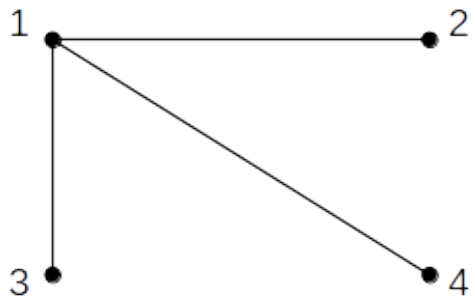
Capitolo di riferimento

Papapdimitriou

- Parte 3, Capitolo 9
  - solo i problemi e i risultati riportati esplicitamente nelle slides

Esercizio 1

Dato il grafo



1. Dire quali sono
   - gli insiemi indipendenti
   - i clique
   - i *node cover*
2. Fornire un certificato succinto per l'istanza di NODE COVER costituita da questo grafo e $K = 2$

Esercizio 2

Premessa

Frammento di Schönfinkel-Bernays

Formule della logica del primo ordine

- senza simboli di funzione
- senza eguaglianza
- in forma prenessa $\exists x_1 ... \exists x_k \forall y_1 ... \forall y_\ell \, \phi$

Teorema

SCHÖNFINKEL-BERNAYS SAT (cioè decidere se una formula del frammento è soddisfacibile) è **NEXP**-completo

[**NEXP**=Nondeterministic EXPonential time]

Esercizio 2

Dire se esistono riduzioni tra i seguenti problemi

- SAT e TSP(D)
- HAMILTON PATH e SET COVER
- CLIQUE e 2-SAT
- SCHÖNFINKEL-BERNAYS SAT e 2-SAT
- MAX FLOW e 3-COLORING

Considerare entrambe le direzioni.

Risposte possibili: "si", "no", "solo se **P** = **NP**"

Esercizio 3

- Specificare un formato per i certificati succinti del 3-COLORING
- Mostrare un certificato per il grafo