
# *Progetto e analisi di algoritmi*

Roberto Cordone

DTI - Università degli Studi di Milano

Polo Didattico e di Ricerca di Crema

Tel. 0373 / 898**089**

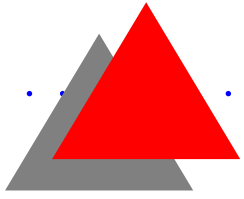
E-mail: **cordone@dti.unimi.it**

Ricevimento: **su appuntamento**

Web page: **http://www.dti.unimi.it/~cordone**

Lezioni: **Martedì dalle 11.00 alle 13.00**

**Giovedì dalle 11.00 alle 13.00**




# *Completezza*

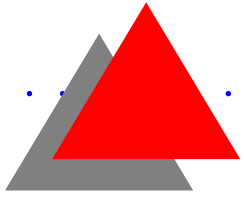
Data una classe di problemi $\mathcal{C}$

- problema $\mathcal{C}$-difficile rispetto a $\preceq_R$ è un problema $Q$ a cui tutti i problemi di $\mathcal{C}$ si riducono

$$P \preceq_R Q \quad \forall P \in \mathcal{C}$$

- problema $\mathcal{C}$-completo rispetto a $\preceq_R$ è un problema $Q$
  1. $\mathcal{C}$-difficile
  2. appartenente a $\mathcal{C}$

I problemi $\mathcal{C}$-completi sono i problemi più difficili di $\mathcal{C}$:
risolto uno, sono risolti tutti!



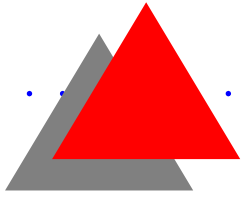
# SAT e $\mathcal{NP}$-completezza


*SAT* è $\mathcal{NP}$-completo (Cook, 1971): con un oracolo per *SAT* si risolve in tempo polinomiale ogni problema in $\mathcal{NP}$

Risolto *SAT*, sono risolti tutti i problemi in $\mathcal{NP}$!

Per la transitività di $\preceq_R$

- se un problema completo si riduce a $Q$, $Q$ è difficile
- i problemi completi si riducono tutti l'uno all'altro



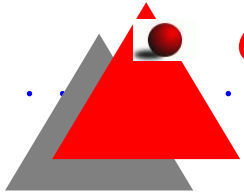
# SAT è $\mathcal{NP}$-completo (1)


Se una NTM risolve $w \in P$ in tempo polinomiale, allora in tempo polinomiale si può trasformare $w$ in una CNF $f$, soddisfacibile se e solo se $w \in P^+$

- La computazione dura $p(|w|)$, con $p$ polinomio
- Computazione: sequenza di $p(n) + 1$ configurazioni
- Configurazione: stringa $\alpha q \beta$
  - $\alpha$: contenuto del nastro prima della testina
  - $q$: stato corrente
  - $\beta$: contenuto del nastro dalla testina in poi

Definiamo le variabili booleane ($q$ stato, $t$ passo, $i$ cella, $j$ simbolo):

- $S(q,t)$ = *Vero* se in $t$ la NTM è nello stato $q$, *Falso* altrimenti;
- $L(i,t)$ = *Vero* se in $t$ la testina è in $i$, *Falso* altrimenti.
- $C(i,j,t)$ = *Vero* se in $t$ il nastro contiene $a_j$ in $i$, *Falso* altrimenti;



# SAT è $\mathcal{NP}$-completo (2)


Un assegnamento di valori descrive una computazione accettante se e solo se

1. la macchina si trova in uno e un solo stato in ogni istante:
   per ogni $t$ esiste uno e un solo $q$: $S(q,t) =$ *Vero*;
2. la macchina legge una e una sola cella in ogni istante:
   per ogni $t$ esiste una e una sola $i$: $L(i,t) =$ *Vero*;
3. ogni cella contiene uno e un solo simbolo in ogni istante:
   per ogni $t$ e ogni $i$ esiste una e una sola $j$: $C(i,j,t) =$*Vero*
4. le variabili con $t = 0$ rappresentano la configurazione iniziale;
5. la macchina termina in uno stato accettante:
   esiste una variabile $S(q, p(|w|)) =$ *Vero* tale che $q_u \in A$;
6. il contenuto delle celle non usate in un istante resta invariato:
   per ogni $t$ e ogni $i$, se $L(i,t) =$ *Falso*, $C(i,j,t) = C(i,j,t+1)$;
7. se $L(i,t) =$ *Vero*, $S$, $L$ e $C$ rispettano la funzione di transizione.

# *Altri problemi $\mathcal{NP}$-completi*


Dato un problema $\mathcal{NP}$-completo $P$, è facile trovarne altri

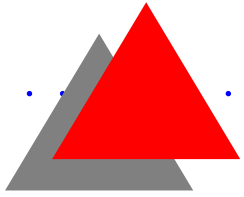
$$\begin{cases} P \text{ è } \mathcal{NP} - \text{completo} \\ P \preceq_R Q \\ Q \in \mathcal{NP} \end{cases} \Rightarrow Q \text{ è } \mathcal{NP} - \text{completo}$$

Se manca l'ipotesi $Q \in \mathcal{NP}$, $Q$ è $\mathcal{NP}$-difficile

Oggi, si conoscono migliaia di problemi $\mathcal{NP}$-completi,

per ora nessun problema in $\mathcal{P}$ è $\mathcal{NP}$-completo

Un algoritmo polinomiale per un problema $\mathcal{NP}$-completo implicherebbe $\mathcal{P} = \mathcal{NP}$



# *Utilità pratica*

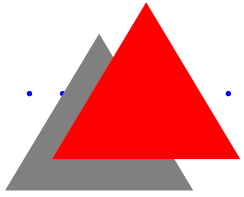

Classificare i problemi suggerisce come risolverli

Per problemi $\mathcal{NP}$-completi o $\mathcal{NP}$-difficili, si giustificano

- metodi esponenziali più raffinati dell'algoritmo esaustivo (es.: *branching*)
- metodi polinomiali euristici (che non garantiscono una soluzione corretta)

È opportuno concentrarsi sui problemi $\mathcal{NP}$-completi cui è più semplice ridurre gli altri, perché buoni algoritmi per loro danno immediatamente buoni algoritmi per gli altri

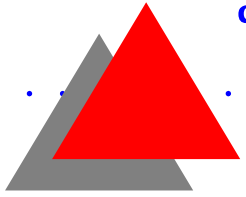
Esempi: *SAT*, la programmazione a numeri interi...



# *Tecniche di riduzione*

$$P \preceq_R Q$$

1. restrizione: l'intero insieme di istanze di $P$ corrisponde a un sottoinsieme di istanze di $Q$ (in questo caso $P \preceq_T Q$)
2. sostituzione locale: ogni istanza di $P$ si trasforma in un'istanza di $Q$ sostituendo ogni sottosistema (di una certa famiglia) con un altro opportuno sottosistema
3. progetto di componenti: si definiscono sottosistemi che "fanno scelte" e sottosistemi che "testano proprietà"; si combinano in modo da comunicare le scelte fatte dai primi ai secondi, che ne verificano la validità



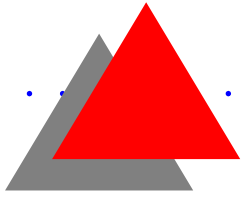

# *Restrizione (1)*

*SAT $\preceq_T$ Programmazione binaria*

- Si associa ogni variabile logica $x_i$ a una variabile binaria $y_i$
- Si sostituiscono
  - i letterali $x_i$ con $y_i$ e $\bar{x}_i$ con $1 - y_i$
  - le somme logiche con somme algebriche
  - i singoli fattori con vincoli di disuguaglianza $\geq 1$

Ogni soluzione ammissibile $y$ per la programmazione binaria corrisponde a un assegnamento di verità $x$ che soddisfa la CNF

($y_i = 1$ sta per $x_i = Vero$, $y_i = 0$ sta per $x_i = Falso$)



# *Restrizione (2)*

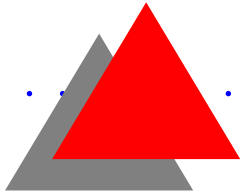
$$(x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_4) \wedge (x_1 \vee x_3 \vee \bar{x}_4)$$

diventa

$$\begin{aligned}
y_1 \quad + \quad y_2 \qquad\qquad + \quad y_4 \quad &\geq 1 \\
(1 - y_1) + (1 - y_2) \qquad\qquad\quad &\geq 1 \\
+ (1 - y_4) \quad &\geq 1 \\
y_1 \qquad + (1 - y_3) + \quad y_4 \quad &\geq 1
\end{aligned}$$

con $y_i \in \{0, 1\}$ per $i = 1, \ldots, 4$

$y = [1, 0, 0, 0]$ diventa $x = [Vero, Falso, Falso, Falso]$




# *Sostituzione locale (1)*

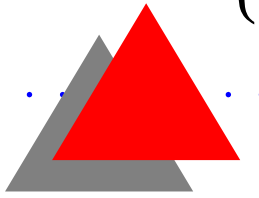
## *SAT* $\preceq_T$ $k$-Clique

Si costruisce un grafo ausiliario con

- un vertice per ogni occorrenza dell'istanza di *SAT*
- un lato per ogni coppia di occorrenze
  - che appartengano a clausole diverse
  - che non corrispondano a letterali opposti

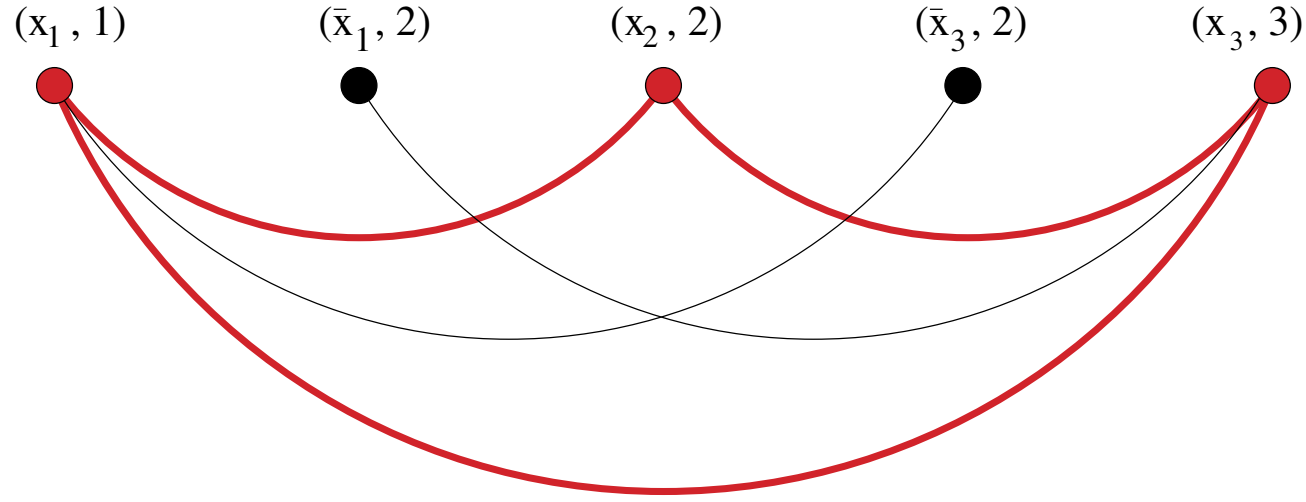
$k$-Clique: $k$ occorrenze fra loro coerenti tratte da $k$ clausole distinte

- Se esiste una $m$-clique, si soddisfano tutte le $m$ clausole
- Se non esiste, la formula non è soddisfacibile
  (un assegnamento soddisfacente determina una $m$-clique)



# *Sostituzione locale (2)*

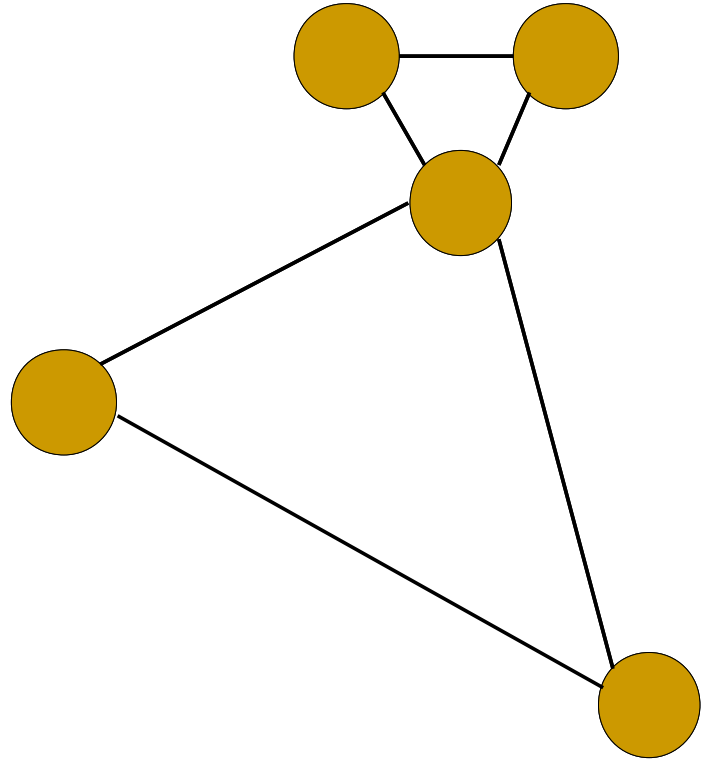
$$(x_1) \wedge (\overline{x}_1 \vee x_2 \vee \overline{x}_3) \wedge (x_3)$$




La clique di grado $m = 3$ corrisponde all'assegnamento soddisfacente $x_1 = x_2 = x_3 =$ *Vero*

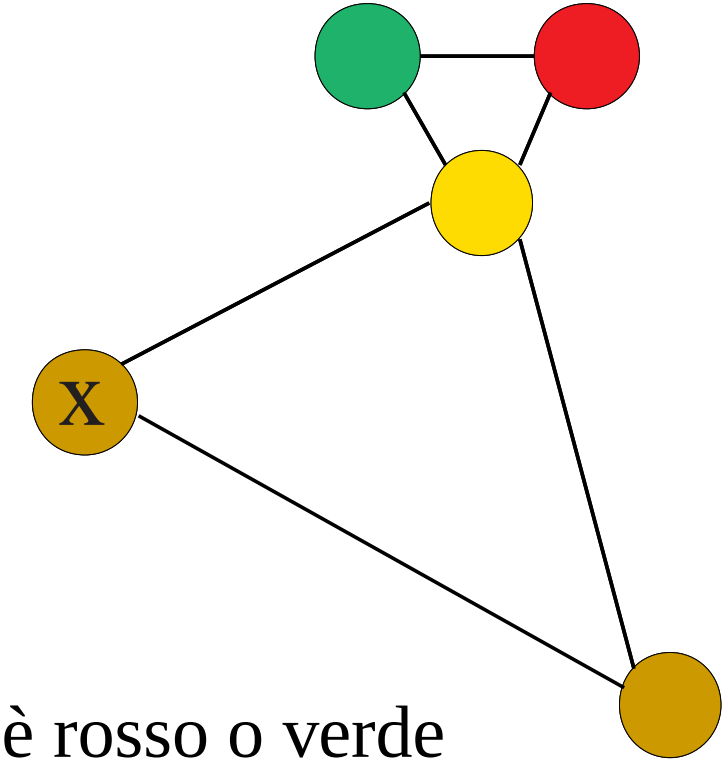
# *Progetto di componenti (1)*

$SAT \preceq_T$ 3-Colouring



# *Progetto di componenti (1)*

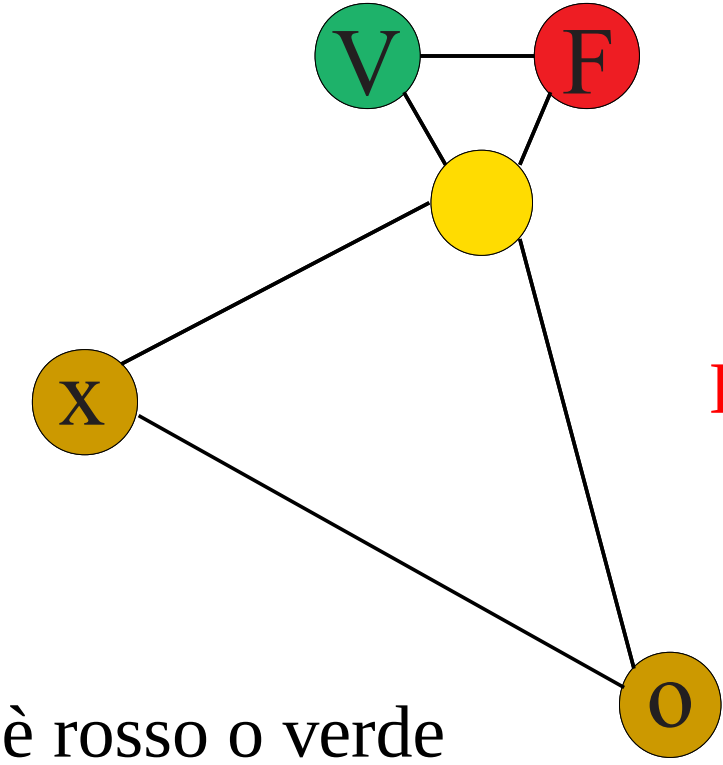
*SAT* $\preceq_T$ 3-Colouring




- Il vertice $x$ è rosso o verde

# *Progetto di componenti (1)*

*SAT* $\preceq_T$ 3-Colouring

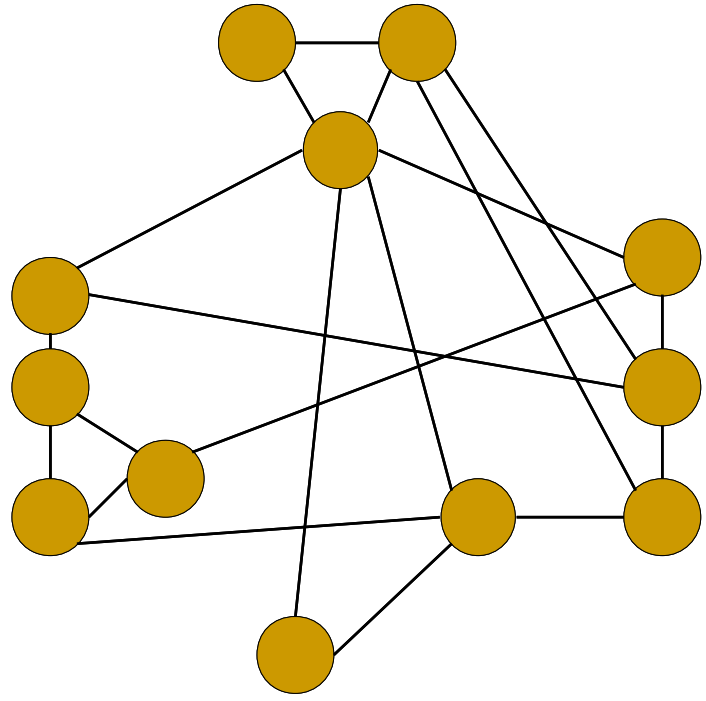


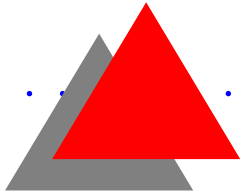

| $x$ | $o$ |
|---|---|
| $F$ | $V$ |
| $V$ | $F$ |

È una porta *NOT*!

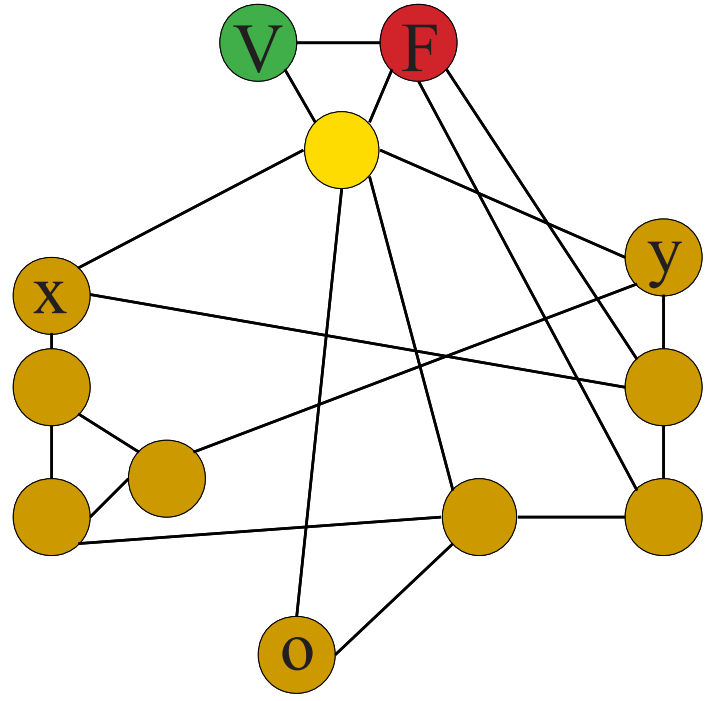
- Il vertice $x$ è rosso o verde
- L'altro vertice è del colore opposto

# *Progetto di componenti (2)*





# *Progetto di componenti (2)*




| $x$ | $y$ | $o$ |
|---|---|---|
| $F$ | $F$ | $F$ |
| $V$ | $F$ | $V$ |
| $F$ | $V$ | $V$ |
| $V$ | $V$ | $V$ |

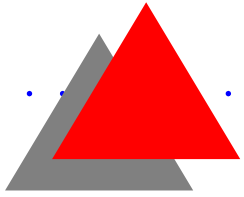
È una porta *OR*!

- Il vertice $o$ è rosso o verde
- Se uno fra i vertici $x$ e $y$ è verde, il vertice $o$ è verde (si scende la catena a destra)
- Se $x$ e $y$ sono rossi, il vertice $o$ è rosso (catena di sinistra)

# *Progetto di componenti (3)*


Realizzare la porta *AND*, osservando che

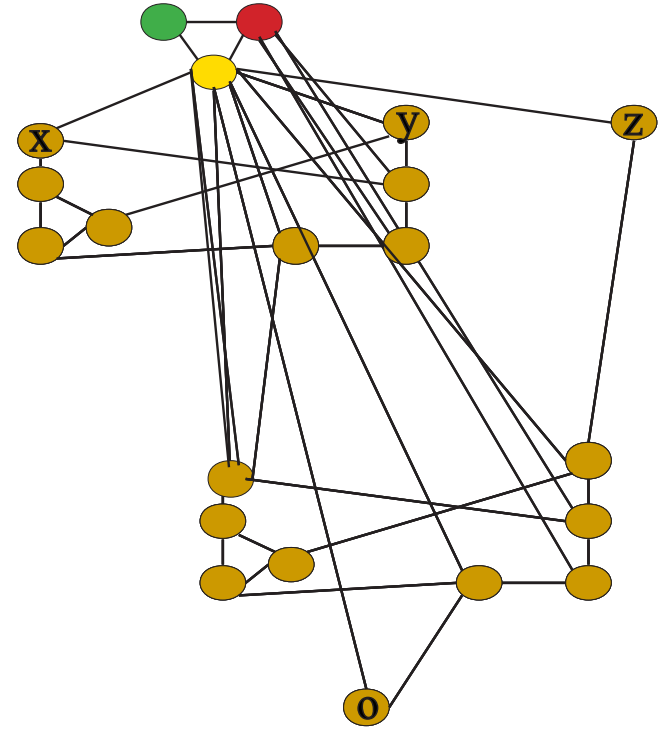
$$(NOT\ (x\ AND\ y)\ ) = (NOT\ x)\ OR\ (NOT\ y)$$



# *Progetto di componenti (4)*

Si dicono gadget i componenti prima descritti
(la clique *Vero*-*Falso*-*Giallo*, le porte *Not*, *Or* e *And*)

Combinandoli si può costruire ogni funzione logica




- i vertici di ingresso
  "fanno scelte"
- gli altri le trasmettono

# *Progetto di componenti (4)*

Si dicono gadget i componenti prima descritti (la clique *Vero-Falso-Giallo*, le porte *Not*, *Or* e *And*)

Combinandoli si può costruire ogni funzione logica

- i vertici di ingresso "fanno scelte"
- gli altri le trasmettono
- un lato dal vertice *Falso* al vertice di uscita "testa la proprietà" che si possa soddisfare la funzione colorando i vertici di ingresso