Progetto e analisi di algoritmi

Roberto Cordone

DTI - Università degli Studi di Milano

Polo Didattico e di Ricerca di Crema

Tel. 0373 / 898**089**

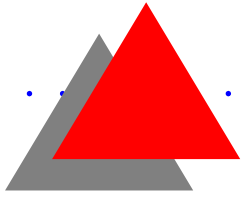
E-mail: **cordone@dti.unimi.it**

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



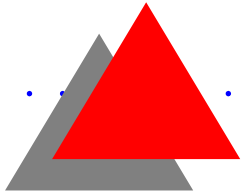
Oracoli (1)

Oracolo per un problema P è una macchina che risolve P in tempo unitario

Non è necessario definire la struttura interna dell'oracolo
L'importante è che funziona!

Una Macchina di Turing equipaggiata con oracolo (*OTM*) è una macchina di Turing che comunica con un oracolo

- lo consulta su stringhe da essa prodotte
- evolve in base al responso dell'oracolo



Oracoli (2)

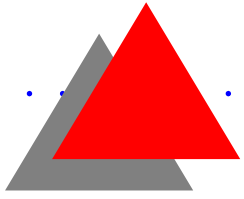
Una *OTM* cambia stato, scrive e si muove sul nastro. . .

- scrive su un nastro supplementare
- consulta l'oracolo, che interpreta il contenuto di tale nastro come istanza di un problema e lo risolve in tempo unitario
- dopo una consultazione agisce anche in base al responso

Quindi, un algoritmo che

- consulta l'oracolo un numero polinomiale di volte
- esegue un numero polinomiale di altre operazioni

diventa un algoritmo polinomiale



Trasformazioni

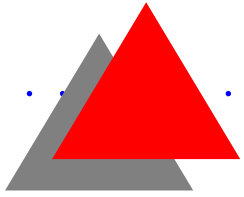
Trasformazione polinomiale del problema P nel problema P' ($P \preceq_T P'$) è una *OTM* che risolve P in tempo polinomiale consultando una sola volta un oracolo per P'

Equivalentemente, è un algoritmo polinomiale che traduce ogni istanza I di P in un'istanza I' di P' e la soluzione S' di I' nella corrispondente soluzione S di I

Un problema P polinomialmente trasformabile in un problema polinomiale è anch'esso polinomiale

$$P \preceq_T P' \text{ e } P' \in \mathcal{P} \Rightarrow P \in \mathcal{P}$$

P' è più difficile di P



Esempi (1)

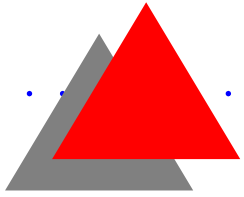
Primalità $\preceq_T$ Compostezza

Dato n, la macchina consulta una sola volta l'oracolo per la "compostezza", inverte il risultato e si ferma

Compostezza $\preceq_T$ Primalità

Dato n, la macchina consulta una sola volta l'oracolo per la "primalità", inverte il risultato e si ferma

Compostezza $\equiv_T$ Primalità



Esempi (2)

Fattorizzazione

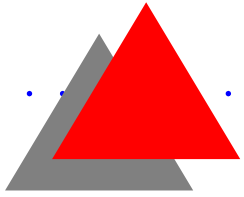
dati n e m, esiste un fattore di n che sia $< m$?

Compostezza $\preceq_T$ Fattorizzazione

Dato n, la macchina consulta una sola volta l'oracolo per la fattorizzazione (con $m = n$), copia il risultato, si ferma

Non si sa se Fattorizzazione $\preceq_T$ Compostezza

Infatti, Compostezza e Primalità sono in $\mathcal{P}$ (dal 2002), mentre non si sa se Fattorizzazione sia in $\mathcal{P}$



Esempi (3)

MinCut $\equiv_T$ MaxFlow

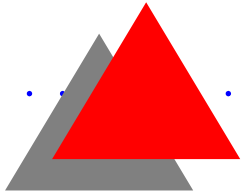
Dati il grafo orientato $G = (N, A)$, i nodi s e t, la funzione capacità $k : A \to \mathbb{N}$ e il numero K, esiste un taglio fra s e t di capacità $\leq K$?

Si ha un oracolo che, dati un grafo orientato $G = (N, A)$, due nodi s e t, una funzione capacità $k : A \to \mathbb{N}$ e un numero H, dice se esiste un flusso $\geq H$ da s a t

La *OTM* passa G, s, t, $k(\cdot)$ e $H = K + 1$ all'oracolo:

- se la risposta è *Vero*, esiste un flusso $\geq K + 1$, per cui nessun taglio è $\leq K$: la macchina risponde *Falso*
- se la risposta è *Falso*, la macchina risponde *Vero*

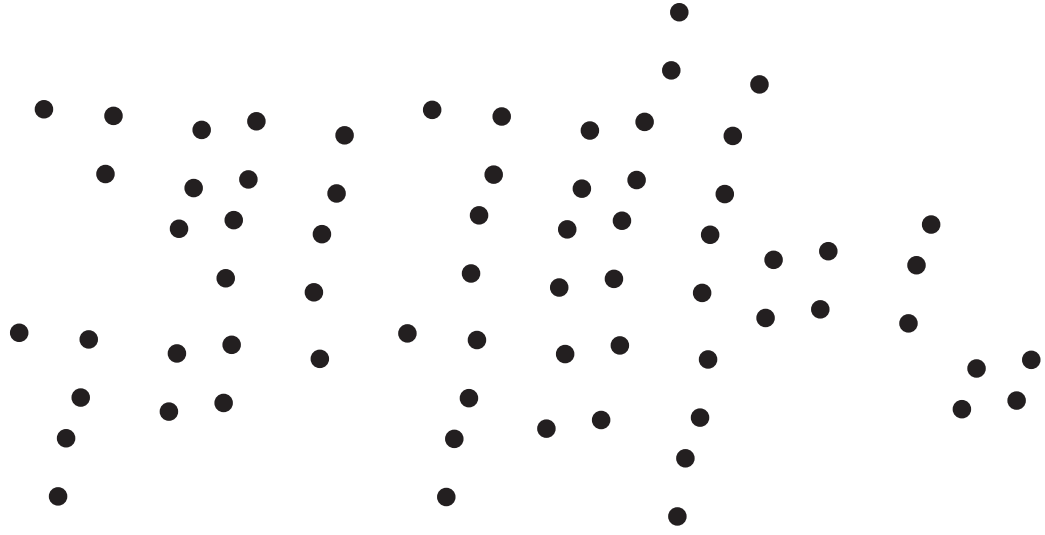
Il passaggio inverso è analogo...



Esempi (4)

InviluppoConvesso

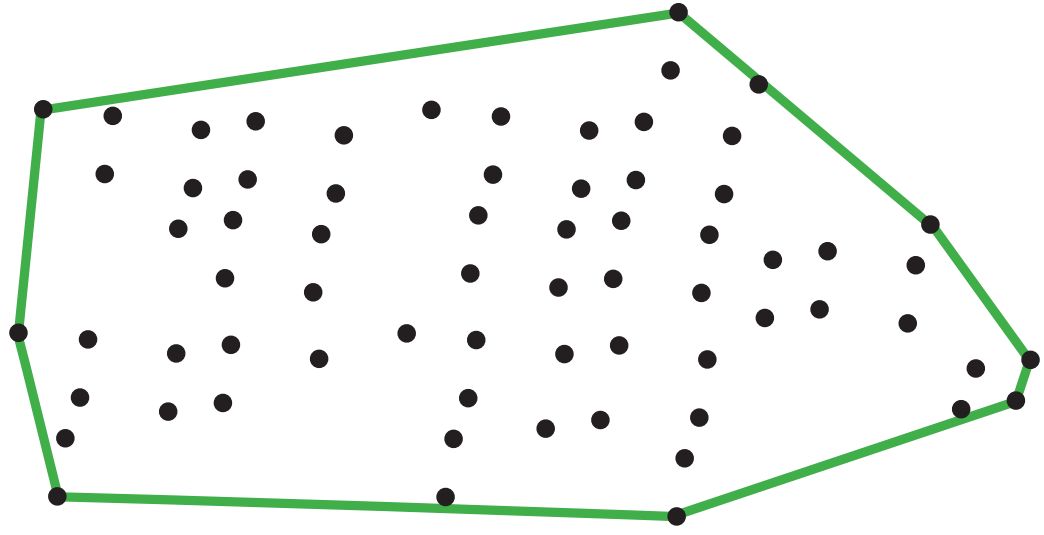
dati n punti sul piano, determinarne l'inviluppo convesso

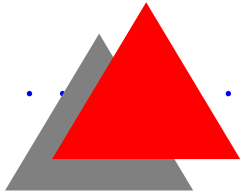


Esempi (4)

InviluppoConvesso

dati n punti sul piano, determinarne l'inviluppo convesso

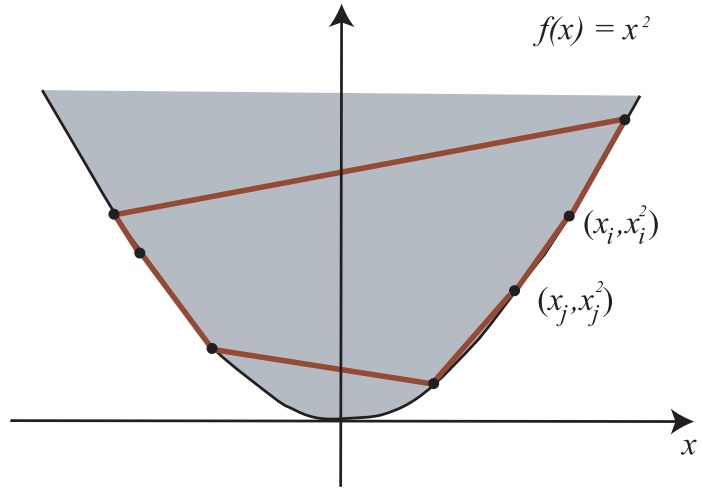




Esempi (5)

Ordinamento $\preceq_T$ InviluppoConvesso

Data una sequenza di n numeri x_i, la macchina costruisce un'istanza di InviluppoConvesso con i punti (x_i, x_i^2)



- tutti i punti sono sull'inviluppo convesso
- l'inviluppo visita i punti in ordine crescente

Esempi (6)

3-*SAT*: data una *CNF* in cui ogni somma contenga al più 3 letterali, esiste un assegnamento di verità che la soddisfi?

SAT $\equiv_T$ 3-*SAT*

1. 3-*SAT* $\preceq_T$ *SAT*, perché 3-*SAT* è un caso particolare di *SAT*: si copia l'istanza, si consulta il *SAT*-oracolo e si copia il responso
2. *SAT* $\preceq_T$ 3-*SAT*: per ogni somma $(l_1 + l_2 + \ldots + l_t)$ di $t \geq 4$ letterali si introducono $t - 3$ nuove variabili y_i e si scrive il prodotto di $t - 2$ somme

$$(l_1 + l_2 + y_1)\,(l_3 + \bar{y}_1 + y_2)\ldots(l_{t-1} + l_t + \bar{y}_{t-3})$$

si consulta il 3-*SAT*-oracolo e si copia il responso

Esempi (7)

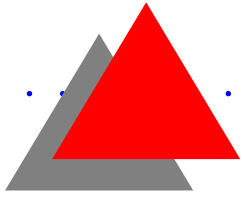
La nuova espressione è soddisfacibile se e solo se lo era quella originale

$$(l_1 + l_2 + \ldots + l_t) \ldots \tag{1}$$

$$(l_1 + l_2 + y_1)\,(l_3 + \bar{y}_1 + y_2) \ldots (l_{t-1} + l_t + \bar{y}_{t-3}) \ldots \tag{2}$$

a) Se (1) non è soddisfacibile ($l_k = 0$), anche (2) non lo è: soddisfarla richiede $y_j = 1$ per ogni j e insieme $y_{t-3} = 0$

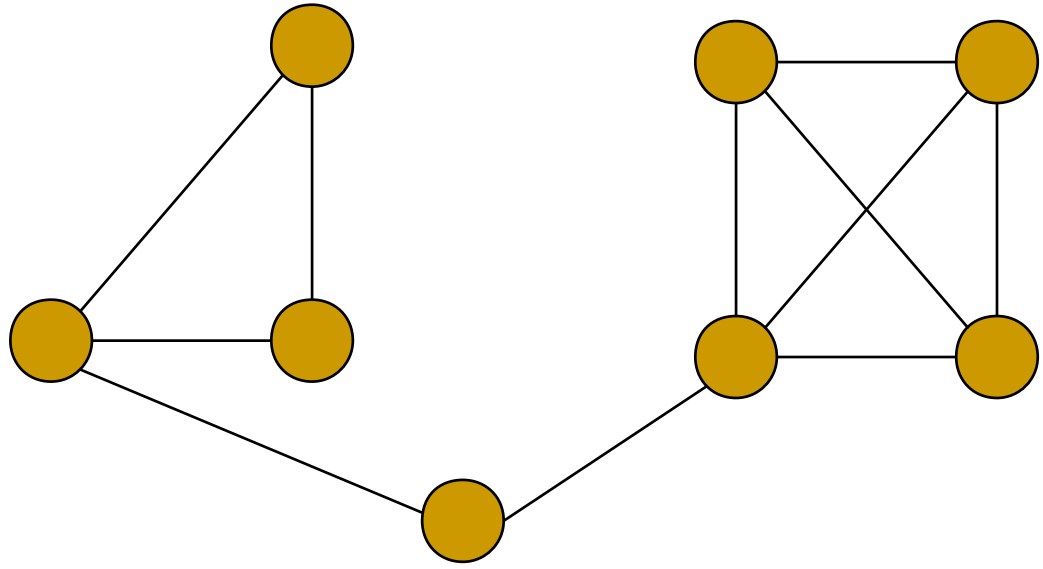
b) Se (1) è soddisfacibile ($\exists l_k = 1$), anche (2) lo è: si pone $y_j = 1$ per $j \leq k - 3$, $y_j = 0$ altrimenti



Esempi (8)

k-Clique: Dato un grafo G, esiste in esso un insieme di k vertici tutti mutuamente adiacenti?

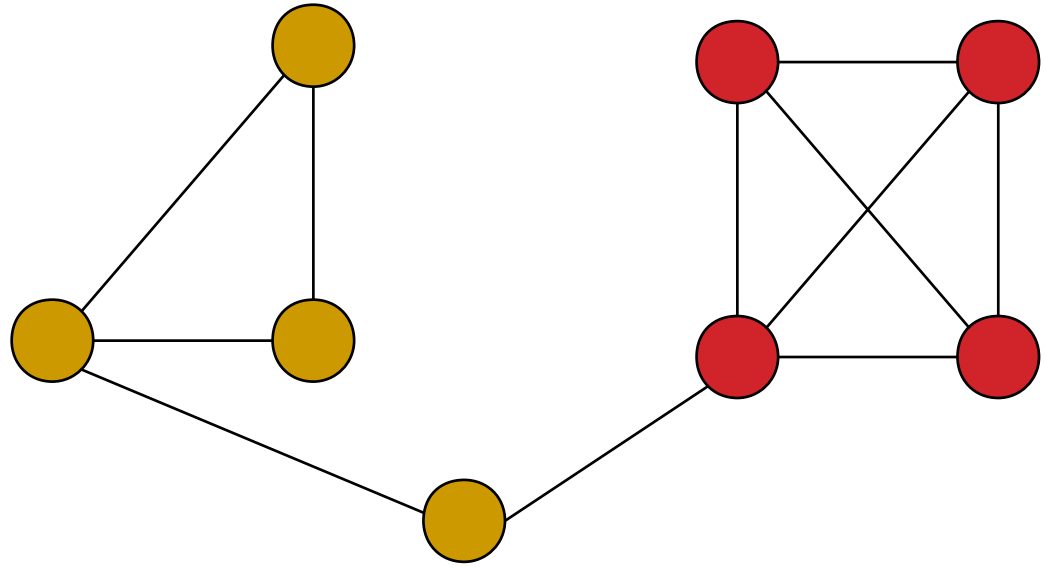
$$k = 4$$



Esempi (8)

k-Clique: Dato un grafo G, esiste in esso un insieme di k vertici tutti mutuamente adiacenti?

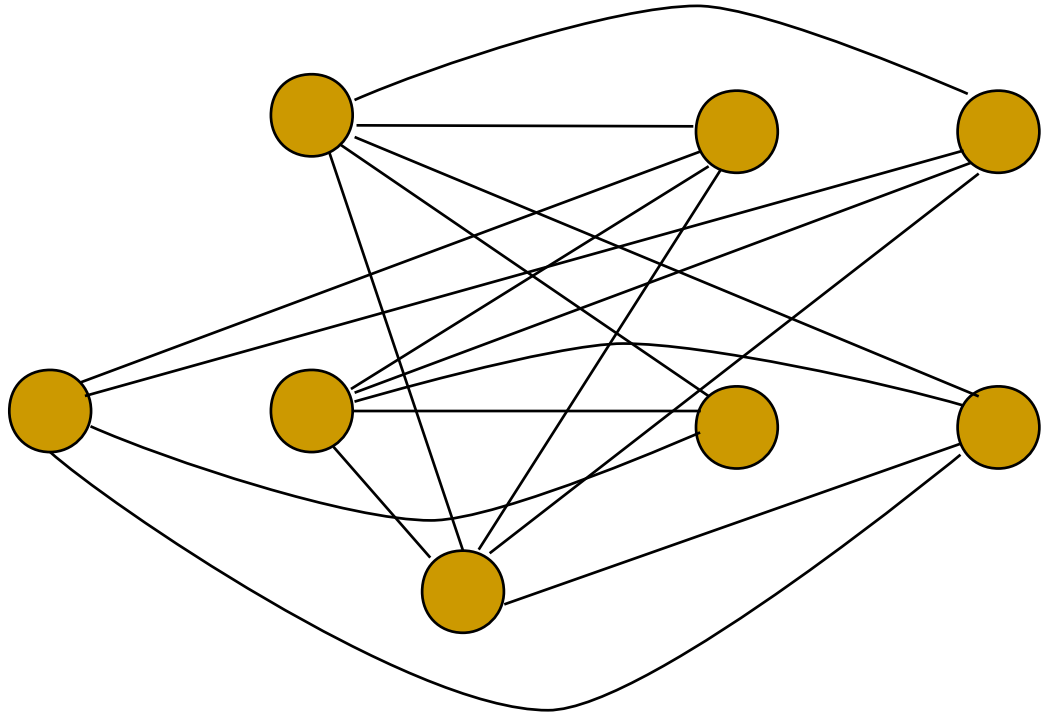
$k = 4$



Esempi (9)

k-IndependentSet: Dato un grafo G, esiste in esso un insieme di k vertici tutti mutuamente non adiacenti?

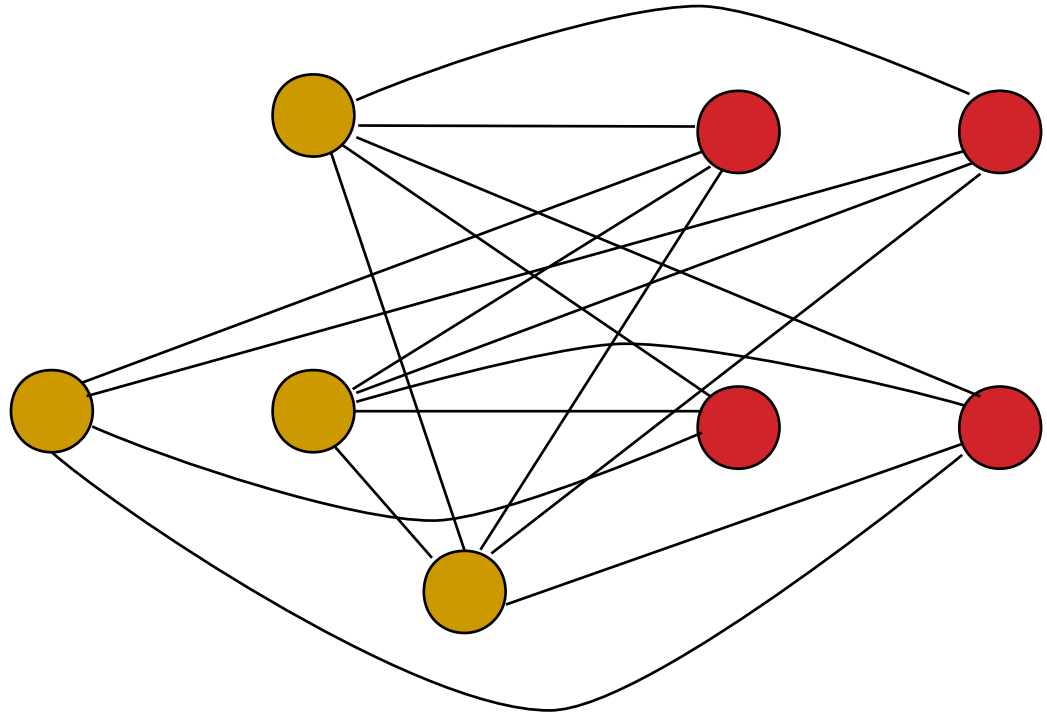
$$k = 4$$



Esempi (9)

k-IndependentSet: Dato un grafo G, esiste in esso un insieme di k vertici tutti mutuamente non adiacenti?

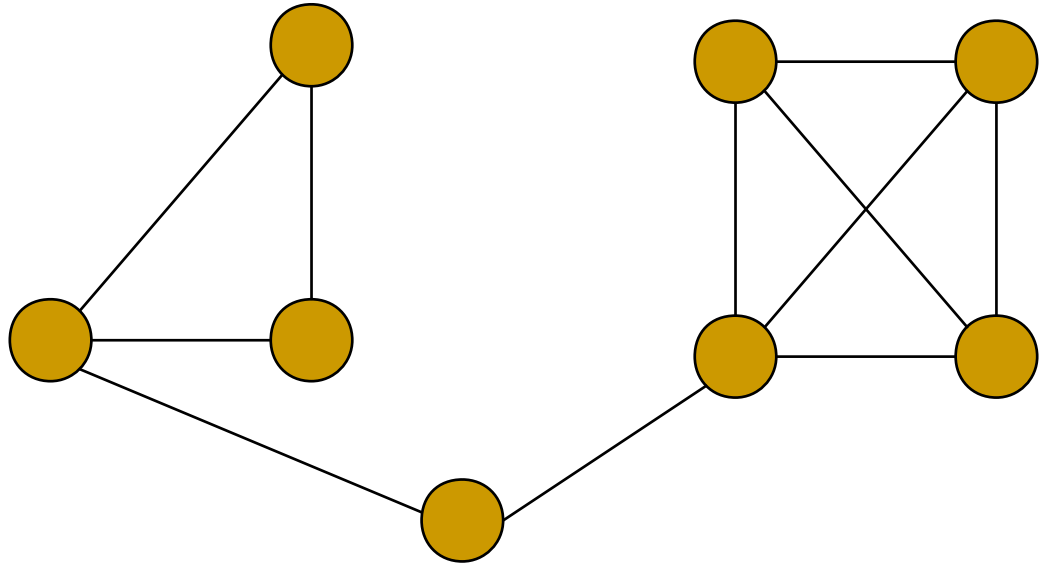
$$k = 4$$



Esempi (10)

k-Clique $\equiv_T$ k-IndependentSet

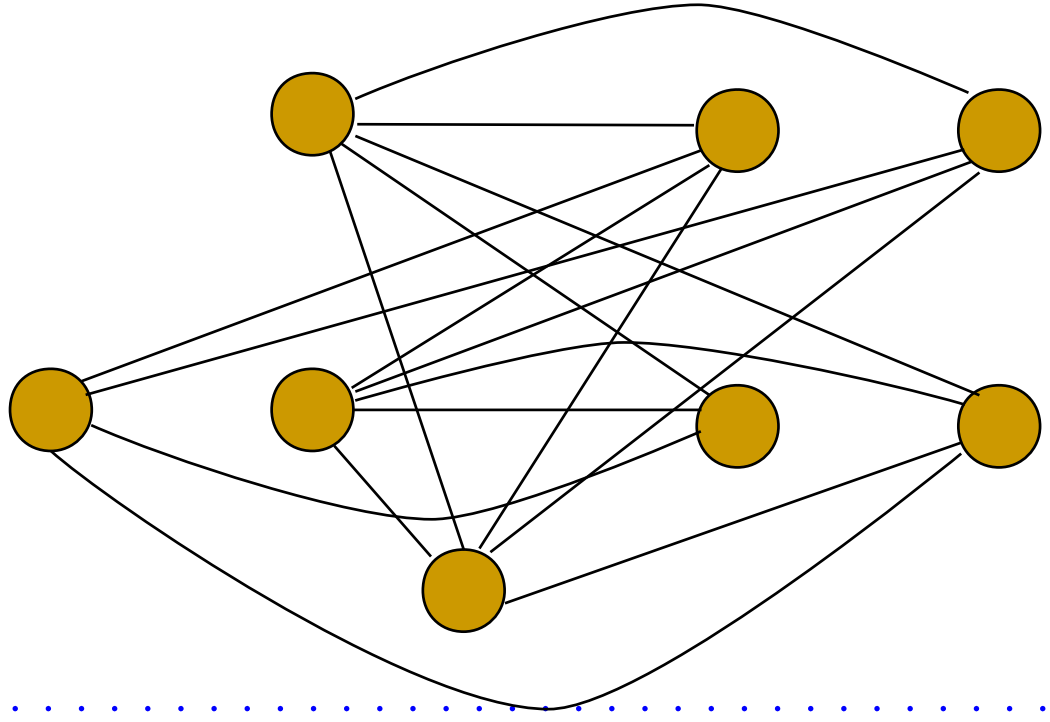
Dato il grafo G , la macchina per la k-Clique costruisce il *grafo complementare* $\bar{G}$, lo passa all'oracolo per il k-IndependentSet e copia il responso in uscita



Esempi (10)

k-Clique $\equiv_T$ k-IndependentSet

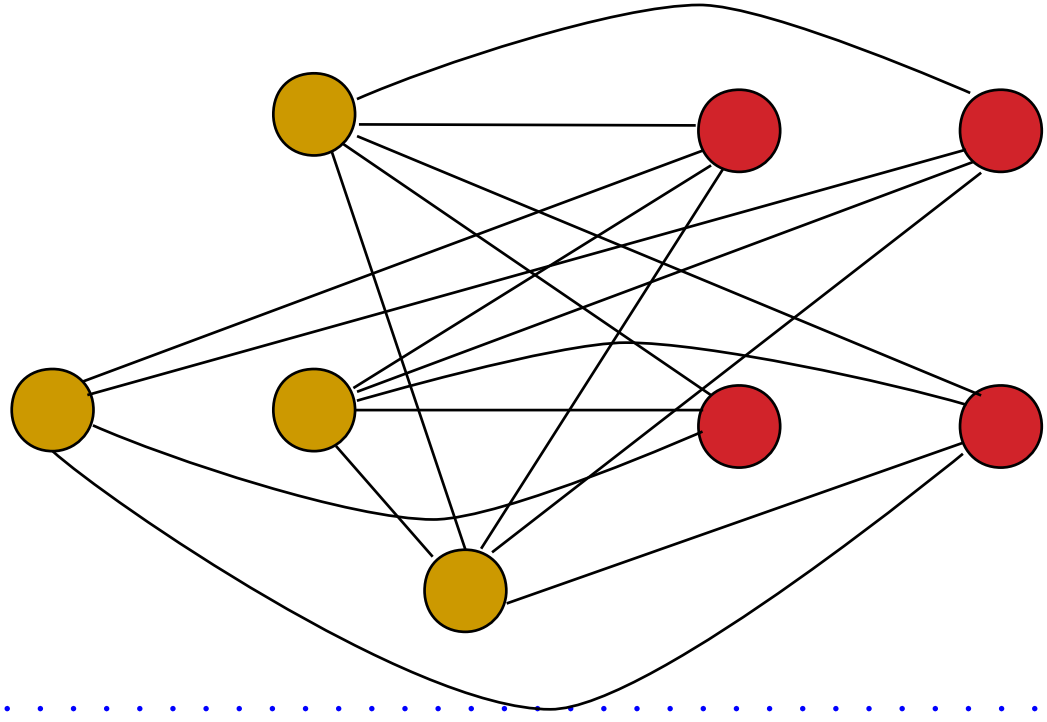
Dato il grafo G , la macchina per la k-Clique costruisce il *grafo complementare* $\bar{G}$, lo passa all'oracolo per il k-IndependentSet e copia il responso in uscita



Esempi (10)

k-Clique $\equiv_T$ k-IndependentSet

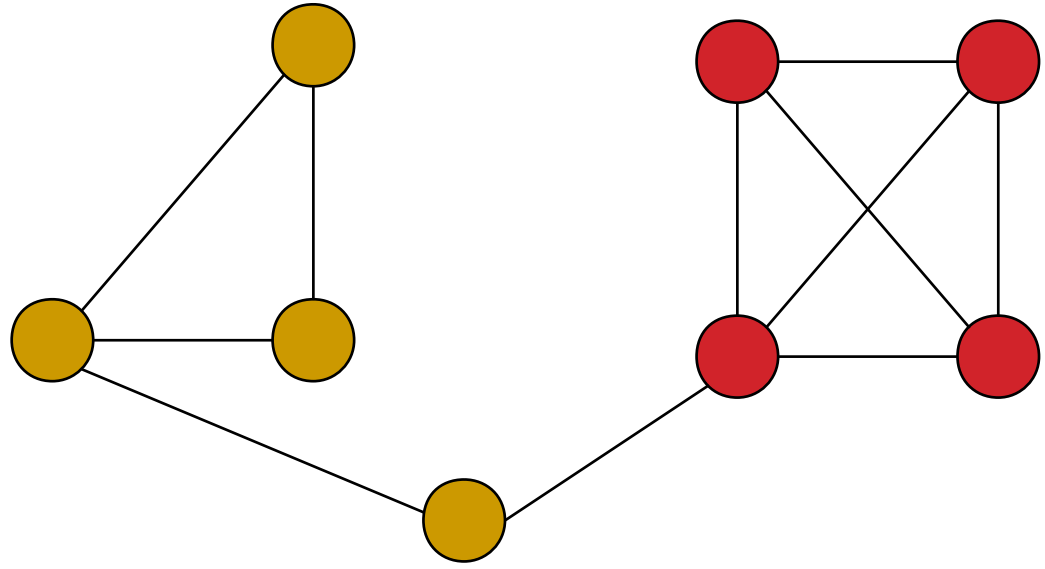
Dato il grafo G , la macchina per la k-Clique costruisce il *grafo complementare* $\bar{G}$, lo passa all'oracolo per il k-IndependentSet e copia il responso in uscita



Esempi (10)

k-Clique $\equiv_T$ k-IndependentSet

Dato il grafo G , la macchina per la k-Clique costruisce il *grafo complementare* $\bar{G}$, lo passa all'oracolo per il k-IndependentSet e copia il responso in uscita

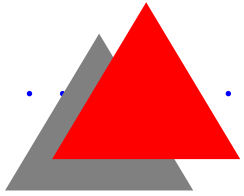


Conseguenze pratiche

In presenza di una trasformazione $P \preceq_T P'$ possiamo

- ordinare i problemi: P' è più difficile di P
- progettare algoritmi: ogni algoritmo per P' dà un algoritmo per P
- definire l'intrattabilità: se P non è polinomiale, anche P' non lo è

Non siamo più costretti a dimostrare proprietà e definire algoritmi per ciascun problema: possiamo generalizzare!



Un'altra definizione?

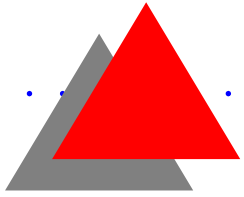
Abbiamo già definito la complessità di un problema come complessità del miglior algoritmo che lo risolve

Ora definiamo la complessità relativa di due problemi attraverso la possibilità di trasformare uno nell'altro

Le due definizioni sono potenzialmente contraddittorie:
sia $A'_{P'}$ il miglior algoritmo esistente per P' ($T_{A'_{P'}} = T_{P'}$)

$$T_{A_P} = T_{I \to I'} + T_{A'_{P'}} + T_{S' \to S} \geq T_{A'_{P'}}$$

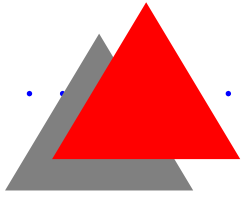
Costa più tempo risolvere il problema facile P (con A_P)
che risolvere il problema difficile P' (con $A'_{P'}$)?



Definizioni incoerenti?

Le due definizioni non sono del tutto coerenti, ma

1. A_P è un algoritmo per P: T_{A_P} stima per eccesso la complessità di P (*ma potrebbe essere esatta...*)
2. spesso $T_{I \to I'}$ e $T_{S' \to S}$ sono $O(T_{A'_{P'}})$, per cui $T_{A_P} = \Theta(T_{A'_{P'}})$ (*ma non sempre...*)
3. T_{A_P} supera $T_{A'_{P'}}$ al più di un polinomio: nessun problema P esponenziale è trasformabile in un problema P' polinomiale



Vantaggi delle due definizioni

La complessità definita attraverso l'algoritmo ottimo

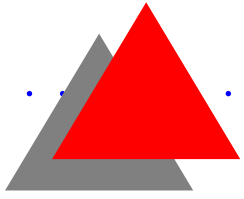
- è un preordine, ma quasi un ordine debole
  (cioè è completo, salvo casi molto particolari)
- dà una misura di complessità assoluta

  ma è quasi inutile quando il gap algoritmico è ampio!

La complessità definita attraverso le trasformazioni polinomiali

- non richiede di conoscere algoritmi per i due problemi
- estende tutti gli algoritmi di un problema all'altro

  ma è un preordine (ammette problemi non confrontabili)



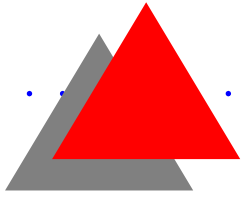
Riduzioni

Riduzione polinomiale del problema P al problema P' ($P \preceq_R P'$) è una *OTM* che risolve P in tempo polinomiale consultando un oracolo per P' un numero polinomiale di volte

P' è non molto più facile di P

Esempio:

- P = ordinamento
- P' = confronto fra numeri interi



Polinomialità

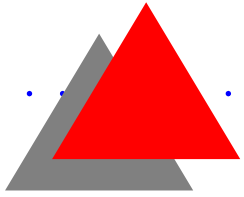
La trasformazione polinomiale e la riduzione polinomiale sono relazioni di preordine (riflessive e transitive)

$$P \preceq_T P' \text{ e } P' \preceq_T P'' \Rightarrow P \preceq_T P''$$

$$P \preceq_R P' \text{ e } P' \preceq_R P'' \Rightarrow P \preceq_R P''$$

La trasformazione polinomiale e la riduzione polinomiale trasmettono "all'indietro" la polinomialità

$$\begin{cases} P \preceq_T P' \\ P' \in \mathcal{P} \end{cases} \Rightarrow P \in \mathcal{P} \quad \text{e} \quad \begin{cases} P \preceq_R P' \\ P' \in \mathcal{P} \end{cases} \Rightarrow P \in \mathcal{P}$$



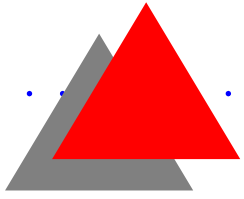
Esistenza e ottimizzazione (1)

Per ogni problema di ottimizzazione P_O ce n'è uno di esistenza P_E

- P_O: "Qual è il costo minimo degli oggetti con date proprietà?"
- Si aggiunge una soglia K al costo di tali oggetti
- P_E: "Esiste un oggetto con date proprietà di costo $\leq K$?"

Per ogni problema di esistenza P_E ce n'è uno di ottimizzazione P_O

- P_E: "Esiste un oggetto con date proprietà?"
- Si aggiunge una funzione di costo c
- P_O: "Qual è il costo minimo degli oggetti con date proprietà?"

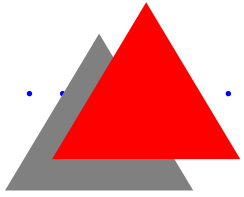


Esistenza e ottimizzazione (2)

$$P_E \preceq_T P_O$$

- L'algoritmo di ottimizzazione fa da oracolo: dato P_E, si costruisce P_O
- Si risolve il problema di ottimizzazione P_O (una sola consultazione dell'oracolo)
- Si confronta la soluzione con la soglia K
- Se l'ottimo supera la soglia, la soluzione di P_E è *Falso*; altrimenti è *Vero*

Quindi $P_O \in \mathcal{OP} \Rightarrow P_E \in \mathcal{P}$



Esistenza e ottimizzazione (3)

$$P_O \preceq_R P_E$$

- L'algoritmo di esistenza fa da oracolo: dato P_O, si costruisce P_E introducendo una soglia parametrica
- Si risolve il problema di ottimizzazione P_E variando la soglia con la ricerca binaria: se la risposta è *Falso*, si abbassa la soglia; altrimenti si alza (diverse consultazioni dell'oracolo)
- Questo determina un intervallo piccolo a piacere contenente l'ottimo (se la funzione di costo è razionale, determina l'ottimo)
- Il numero di consultazioni è polinomiale, dato che la differenza tra minimo e massimo dell'obiettivo è polinomiale in $|I|$

Quindi $P_E \in \mathcal{P} \Rightarrow P_O \in \mathcal{OP}$