APA Modulo 1 Lezione 12

Elena Zucca

1 aprile 2020

Problemi NP-completi

- i problemi NP-completi sono i “più difficili” tra i problemi in NP
- ossia: se per qualcuno di questi problemi si trovasse un algoritmo polinomiale, se ne troverebbe uno per tutti quelli in NP, e quindi si dimostrerebbe che P = NP
- infatti ognuno dei problemi in NP è riducibile a un problema NP-completo, nel senso formalizzato nel seguito

Riducibilità polinomiale

Definizione

Dati due problemi concreti $\mathcal{P}_1$ e $\mathcal{P}_2$, diciamo che $\mathcal{P}_1$ è riducibile polinomialmente a $\mathcal{P}_2$, e scriviamo $\mathcal{P}_1 \leq_{\mathsf{P}} \mathcal{P}_2$, se esiste una funzione

$$f\colon \{0,1\}^\star \to \{0,1\}^\star$$

detta funzione di riduzione, calcolabile in tempo polinomiale, tale che

per ogni $x \in \{0,1\}^\star$, $x \in \mathcal{P}_1$ *se e solo se* $f(x) \in \mathcal{P}_2$.

Riducibilità polinomiale

- nel caso di due problemi astratti $\mathcal{P}_1 \colon I_1 \to \{T, F\}$ e $\mathcal{P}_2 \colon I_2 \to \{T, F\}$
- la funzione di riduzione sarà:

  $$f \colon I_1 \to I_2$$

- NB: un predicato $\mathcal{P} \colon I \to \{T, F\}$ può essere visto equivalentemente come insieme (dei valori su cui vale T)
  $\{x \mid \mathcal{P}(x) = T\}$
- quindi $x \in \mathcal{P}$ equivale a $\mathcal{P}(x) = T$

Esempio

- il problema della soddisfacibilità è polinomialmente riducibile a quello delle formule quantificate
- infatti data ϕ con variabili $x_1, \ldots, x_n$ la formula di cui vogliamo verificare la soddisfacibilità
- la funzione f la trasforma in $\exists x_1 \ldots . \exists x_n . \phi$
- è una funzione calcolabile in tempo polinomiale, e ϕ soddisfacibile se e solo se $\exists x_1 \ldots . \exists x_n . \phi$ vera

Lemma

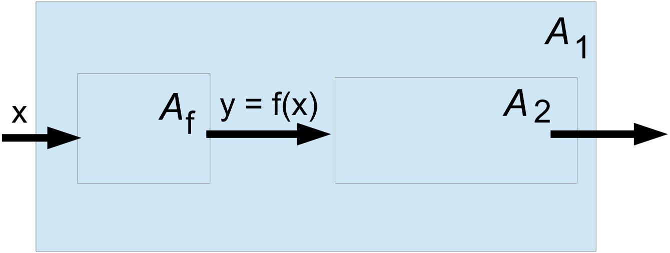
Dati due problemi $\mathcal{P}_1$ e $\mathcal{P}_2$ tali che $\mathcal{P}_1 \leq_{\mathrm{P}} \mathcal{P}_2$

se $\mathcal{P}_2 \in P$ allora anche $\mathcal{P}_1 \in P$

Dimostrazione.

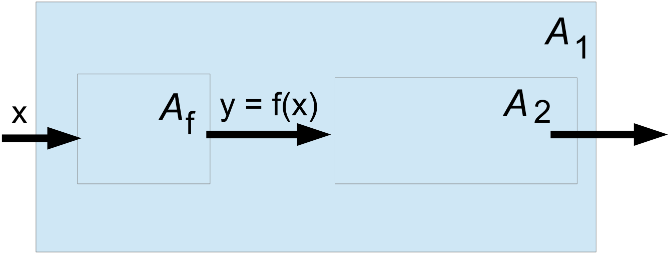
Dato un algoritmo polinomiale A_2 che risolve $\mathcal{P}_2$, un algoritmo polinomiale A_1 che risolve $\mathcal{P}_1$ può essere ottenuto eseguendo, a partire da x, prima l'algoritmo polinomiale che calcola $f(x)$, poi $A_2(f(x))$. L'algoritmo A_1 risulta polinomiale. L'algoritmo A_1 risolve $\mathcal{P}_1$ in quanto
$A_1(x) = A_2(f(x)) = \mathcal{P}_2(f(x)) = \mathcal{P}_1(x)$. □

Graficamente



- A_1 è polinomiale in quanto composizione in sequenza di due algoritmi polinomiali
- NB: A_2 è polinomiale rispetto a $|y|$
- MA $|y|$ è polinomiale rispetto a $|x|$

Graficamente



- A_1 è corretto perché
- se $x \in \mathcal{P}_1$ allora $y \in \mathcal{P}_2$
- se $x \notin \mathcal{P}_1$ allora $y \notin \mathcal{P}_2$

Problemi NP-completi

Definizione

Un problema $\mathcal{P}$ si dice

- NP-arduo o NP-difficile (in inglese NP-hard) se per ogni problema $\mathcal{Q} \in \mathrm{NP}$ si ha $\mathcal{Q} \leq_{\mathrm{P}} \mathcal{P}$
- NP-completo se appartiene alla classe NP ed è NP-arduo.
- Indichiamo con NP-C la classe dei problemi NP-completi.

Nota: Le nozioni di hard e completo si possono dare relativamente a una qualunque classe di problemi.

$P \stackrel{?}{=} NP$

Teorema

- Se un qualunque problema NP-completo appartiene alla classe P, allora si ha $P = NP$.
- Equivalentemente: se è $P \neq NP$, quindi esiste almeno un problema in NP non risolvibile in tempo polinomiale, allora nessun problema NP-completo è risolvibile in tempo polinomiale.

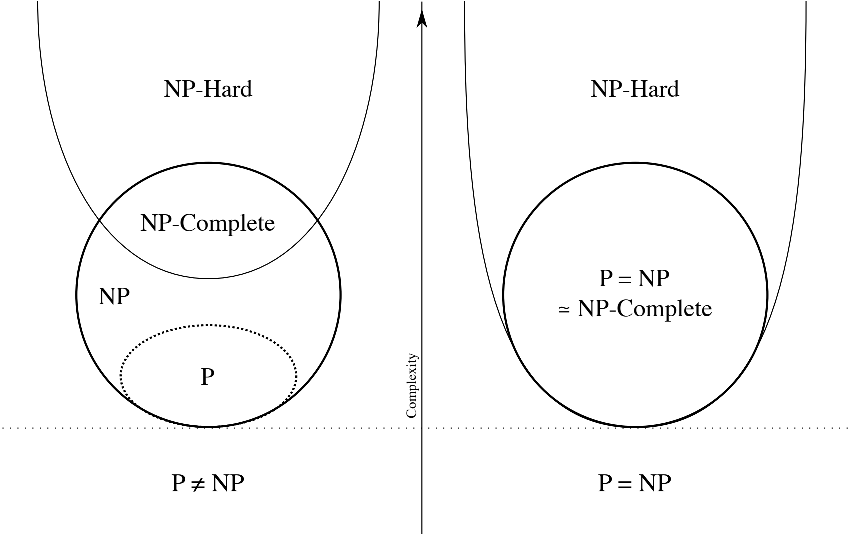
Dimostrazione.

Ovvio in base al precedente lemma. $\square$

Come risolvere $P \stackrel{?}{=} NP$

- per risolvere il problema aperto due possibilità:
- per un qualunque problema in NP-C, trovare un algoritmo polinomiale $\Rightarrow$ P = NP
- per un qualunque problema in NP-C, provare che non esiste un algoritmo polinomiale $\Rightarrow$ P $\neq$ NP

Graficamente



Come provare che un problema è NP-completo?

- per dimostrare che è in NP basta mostrare un algoritmo polinomiale che lo verifica (facile)
- come possiamo invece dimostrare che un problema in NP è NP-hard?
- punto di partenza: dimostrare che almeno un problema, sia $\hat{\mathcal{P}}$, è NP-completo, ossia $\mathcal{Q} \leq_{\mathsf{P}} \hat{\mathcal{P}}$ per ogni $\mathcal{Q} \in \mathsf{NP}$
- poi, possiamo provare che un altro, sia $\mathcal{P}$, lo è, mostrando $\hat{\mathcal{P}} \leq_{\mathsf{P}} \mathcal{P}$
- infatti per la transitività della relazione di riducibilità:
- $\mathcal{Q} \leq_{\mathsf{P}} \hat{\mathcal{P}}$ e $\hat{\mathcal{P}} \leq_{\mathsf{P}} \mathcal{P}$ implica $\mathcal{Q} \leq_{\mathsf{P}} \mathcal{P}$ per ogni $\mathcal{Q} \in \mathsf{NP}$
- nota metodologica: provare che un problema è NP-completo è utile opportuno cercare algoritmo approssimato o ridursi a sottoproblema per cui si possa trovare un algoritmo polinomiale

Teorema di Cook

- bisogna quindi trovare un "primo" problema $\hat{\mathcal{P}}$ di cui dimostrare la NP-completezza
- difficile: bisogna provare $\mathcal{Q} \leq_{\mathsf{P}} \hat{\mathcal{P}}$ per ogni $\mathcal{Q} \in \mathsf{NP}$
- storicamente, il primo problema di cui è stata provata la NP-completezza (da Stephen Cook nel 1971) è il problema della soddisfacibilità.

Teorema di Cook

SAT è NP-completo.

- idea prova: algoritmo che, dato un problema $\mathcal{Q} \in \mathsf{NP}$ e un input x per $\mathcal{Q}$, costruisce una formula in CNF che descrive un algoritmo non deterministico per $\mathcal{Q}$, ossia la formula è soddisfacibile se e solo se l'algoritmo restituisce T.