Complementi ed Esercizi di Informatica Teorica

Vincenzo Bonifaci

14 aprile 2009

2 Problemi P-completi

Ricordiamo che un linguaggio A è detto P-completo (rispetto a riduzioni logspace) se:

i. $A \in \mathsf{P}$;

ii. A è P-difficile, ovvero per ogni linguaggio $B \in \mathsf{P}$, si ha che $B \leq_{\mathrm{L}} A$.

È importante che la riduzione di cui al punto ii) sia una riduzione logspace; se si permettessero riduzioni dal tempo polinomiale, ogni linguaggio in P sarebbe banalmente P-completo (perché?). I linguaggi P-completi sono i linguaggi "più difficili" della classe P e in particolare sono *intrinsecamente sequenziali*, nel senso che se uno di questi linguaggi appartenesse alla classe dei linguaggi efficientemente parallelizzabili NC, allora si avrebbe $\mathsf{NC} = \mathsf{P}$, ovvero *tutti* i linguaggi in P sarebbero efficientemente parallelizzabili.

Un problema di decisione è P-completo se lo è il corrispondente linguaggio. Un tipico esempio di problema P-completo è quello di decidere il valore calcolato da un circuito booleano conoscendo i valori degli ingressi. Usando questo risultato come punto di partenza, dimostriamo in questo capitolo la P-completezza di tutta una serie di altri problemi.

2.1 Valore calcolato da un circuito monotono

Il problema del *valore calcolato da un circuito monotono* (VCCM) è il seguente: dato un circuito di soli AND e OR insieme con i valori dei suoi ingressi, decidere se l'output del circuito è `true`. Ovviamente, dato che il valore calcolato da un circuito qualsiasi (VCC) può essere deciso in tempo polinomiale, anche questo problema può essere risolto in tempo polinomiale. Meno banalmente, mostriamo che in effetti anche questo problema, come il problema del valore calcolato da un circuito, è P-completo, attraverso la riduzione VCC $\leq_{\mathrm{L}}$ VCCM dove la L indica il fatto che si tratta di una riduzione logspace.

Per fare questo passiamo da una istanza di VCC ad un'istanza di VCCM basata su una logica "a doppio binario": per ogni variabile v del circuito originale manteniamo nel nuovo circuito la coppia $(v, \overline{v})$. Sostituiamo ogni porta AND del tipo $x \wedge y$ con una coppia di porte $(x \wedge y, \overline{x} \vee \overline{y})$ e ogni porta OR del tipo $x \vee y$ con una coppia $(x \vee y, \overline{x} \wedge \overline{y})$. Infine, una porta NOT avente ingresso x è sostituita dallo "scambio" $(\overline{x}, x)$. Le sostituzioni calcolano correttamente i valori delle variabili in base alle leggi di De Morgan e sono abbastanza semplici da poter essere implementate in spazio logaritmico. Il circuito così ottenuto è monotono ed è equivalente al circuito dato.

Esercizio 2.1. Il problema *valore calcolato da circuito di NAND* è il seguente: dato un circuito costituito da sole porte NAND e dati i valori in ingresso, decidere se l'output del circuito è `true`. Dimostrare che questo problema è P-completo.

2.2 Soddisfacibilità di formule di Horn

Una *clausola di Horn* è una clausola logica con al più un letterale positivo, ovvero una clausola del tipo $\overline{x}_1 \vee \overline{x}_2 \vee \cdots \vee \overline{x}_k \vee y$ oppure $\overline{x}_1 \vee \overline{x}_2 \vee \cdots \vee \overline{x}_k$ con $k \geq 0$. Il problema della *soddisfacibilità di formule di Horn* (HORNSAT) è il seguente: dato un insieme di clausole di Horn, decidere se esiste una assegnazione di valori di verità alle variabili che soddisfi tutte le clausole.

Notiamo che una clausola di Horn con esattamente un letterale positivo può sempre essere messa nella forma $(x_1 \wedge x_2 \wedge \cdots \wedge x_k) \Rightarrow y$. Ad esempio le due clausole $\overline{x}_1 \vee \overline{x}_2 \vee y$ e z possono essere messe nella forma $x_1 \wedge x_2 \Rightarrow y$ e $\texttt{true} \Rightarrow z$. Queste clausole sono dette *implicazioni*, mentre le altre, quelle senza letterali positivi, sono dette *puramente negative*. Per determinare se un insieme di clausole di Horn ϕ è soddisfacibile, consideriamo inizialmente solo le implicazioni di ϕ. L'algoritmo assegna inizialmente il valore di verità `false` a tutte le variabili. Dopodiché, finché non si ha che tutte le implicazioni sono soddisfatte, l'algoritmo prende una qualsiasi implicazione non soddisfatta $x_1 \wedge x_2 \wedge \ldots \wedge x_k \Rightarrow y$ e pone y a `true`. Quando l'algoritmo termina, tutte le implicazioni sono soddisfatte. Inoltre l'insieme T delle variabili poste a `true` è il più piccolo possibile, nel senso che qualsiasi altra assegnazione che soddisfi tutte le implicazioni di ϕ pone a `true` un insieme di variabili $T' \supseteq T$. Ora si ha che ϕ è soddisfacibile se e solo se l'assegnazione ottenuta dall'algoritmo soddisfa ϕ. Infatti se una clausola puramente negativa $(\overline{x}_1 \vee \ldots \vee \overline{x}_k)$ di ϕ non è soddisfatta dall'algoritmo, vuol dire che $x_1, \ldots, x_k$ sono state poste tutte a `true` e che quindi una qualsiasi assegnazione che soddisfi le implicazioni di ϕ deve porre anch'essa a `true` tali variabili, non soddisfacendo così la clausola. Poiché l'algoritmo può essere chiaramente eseguito in tempo polinomiale, abbiamo mostrato che HORNSAT $\in$ P.

Per mostrare che HORNSAT è P-completo, utilizziamo una riduzione dal problema del *valore calcolato da un circuito monotono* (VCCM). Ad ogni porta logica e ad ogni porta di ingresso del circuito dato, associamo una variabile logica. Costruiremo le formule di Horn in maniera tale da rappresentare, intuitivamente, con variabili `true` quelle porte e quegli input che nel circuito erano `false`. Facciamo ciò nel seguente modo. Per ogni porta di input x con valore `false` nel circuito, aggiungiamo l'implicazione $\texttt{true} \Rightarrow x$. Per ogni porta AND del tipo $z = x \wedge y$, aggiungiamo le implicazioni $x \Rightarrow z$ e $y \Rightarrow z$ (in quanto se x o y erano `false` nel circuito, lo doveva essere anche z). Per ogni porta OR del tipo $z = x \vee y$, aggiungiamo l'implicazione $x \wedge y \Rightarrow z$ (con un ragionamento analogo). Infine, se u è la variabile legata alla porta di uscita del circuito, aggiungiamo la clausola puramente negativa $\overline{u}$.

Ora, se il circuito calcolava uscita `true`, assegnando ad ogni variabile l'opposto del valore corrispondente nel circuito abbiamo che tutte le formule sono soddisfatte. Viceversa, se per una data assegnazione tutte le formule soddisfatte, innanzitutto possiamo dire che tutte le porte che erano `false` nel circuito corrispondono a variabili `true` nell'assegnazione (questo per come abbiamo costruito le formule). Ma se tutte le formule sono soddisfatte, lo è anche la formula $\overline{u}$, per cui $u = \texttt{false}$. Questo significa che l'uscita del circuito non poteva essere `false` (altrimenti avremmo avuto $u = \texttt{true}$, per quanto detto sopra). Quindi se le formule sono soddisfacibili, l'uscita del circuito è `true`.

Poiché la trasformazione è effettuabile in spazio logaritmico, abbiamo che VCCM $\leq_{\mathrm{L}}$ HORNSAT.

2.3 Accessibilità di un sistema di cammini

Dato un insieme di nodi V, un *sistema di cammini* su V è un insieme di triple $T \subseteq V^3$. Diciamo che un nodo i è *accessibile* se $i = 1$ oppure se esistono due nodi accessibili j, j' tali che $(j, j', i) \in T$. Il problema dell'*accessibilità di un sistema di cammini* (PATH) è il seguente: dato un sistema di cammini su V, decidere se il nodo n è accessibile.

Per vedere che questo problema è in P, notiamo che è possibile calcolare l'insieme di tutti i nodi accessibili A ponendo inizialmente $A := \{1\}$ e inserendo di volta in volta in A tutti i nodi i tali che $(j, j', i) \in T$ con j e j' già presenti in A, fin quando non è più possibile aggiungere nodi ad A. A quel punto n sarà in A se e solo se n è accessibile nel sistema T.

Per dimostrare che il problema è anche P-completo, mostriamo una riduzione dal problema della soddisfacibilità di clausole di Horn: HORNSAT $\leq_{\mathrm{L}}$ PATH. A questo scopo, dato un insieme di formule di Horn, associamo ad ogni variabile logica x un nodo. Consideriamo anche due nodi speciali chiamati `true` e `false`. Quindi se X è l'insieme delle variabili logiche poniamo $V = X \cup \{\texttt{true}, \texttt{false}\}$. Inoltre ad ogni clausola di Horn associamo un elemento di T in questo modo: ad ogni clausola del tipo $\texttt{true} \Rightarrow x$ associamo la tripla $(\texttt{true}, \texttt{true}, x)$; ad ogni clausola della forma $x \wedge y \Rightarrow \texttt{false}$ (clausola puramente negativa) associamo la tripla $(x, y, \texttt{false})$; infine, ad ogni clausola del tipo $x \wedge y \Rightarrow z$ associamo la tripla (x, y, z). Riduciamo anche gli AND a più variabili a delle sequenze di AND di due variabili introducendo nodi aggiuntivi. Infine, identifichiamo il nodo 1 del sistema con `true` e il nodo n con `false`. Dalla costruzione, abbiamo che l'insieme di clausole è soddisfacibile se e solo se il nodo n (`false`) non è accessibile nel sistema T. Infatti se il nodo n è accessibile e le clausole sono soddisfacibili, significa che esiste una sequenza di implicazioni tramite la quale è possibile "derivare" il valore logico `false` a partire dal valore `true` ottenendo la contraddizione logica $\texttt{true} \Rightarrow \texttt{false}$. D'altra parte, se il nodo n non è accessibile, possiamo soddisfare tutte le clausole ponendo a `true` tutte le variabili corrispondenti a nodi accessibili.

2.4 Disequazioni lineari

Nel problema delle *disequazioni lineari* (DL), sono dati una matrice A di dimensioni $n \times d$ e un vettore b di dimensioni $n \times 1$ a elementi interi. Il problema chiede se esiste un vettore $x \geq 0$ a elementi razionali tale che $Ax \leq b$ (ovviamente x ha dimensioni $d \times 1$). Questo problema è sostanzialmente la versione decisionale del problema della programmazione lineare e ha quindi una grande importanza.

Si tratta di un problema P-completo. Non vedremo qui perché il problema è in P; si tratta di un fatto assolutamente non banale dimostrato da Khachian nel 1979. Mostriamo invece che il problema è P-difficile esibendo una riduzione dal problema del valore calcolato da un circuito: VCC $\leq_{\mathrm{L}}$ DL. A questo scopo, associamo ad ogni porta del circuito una variabile e riduciamo il corretto funzionamento delle porte logiche a insiemi di disequazioni. Per ogni variabile u inseriamo il vincolo $u \leq 1$. Inoltre, ad ogni porta NOT con ingresso u e uscita v associamo l'equazione $v = 1 - u$, ovvero la coppia di disequazioni $u + v \leq 1$ e $-u - v \leq -1$. Ad ogni porta AND con ingressi u, v e uscita w associamo le tre disequazioni $w \leq u$, $w \leq v$, $u + v - 1 \leq w$. Realizziamo una porta OR attraverso tre NOT e un AND usando la legge di De Morgan. Infine, aggiungiamo i vincoli $u = 1$ per ogni porta di ingresso pari a `true`, $u = 0$ per ogni porta di ingresso pari a `false` e il vincolo $z = 1$ per la porta di uscita z del circuito. Notiamo che se le variabili corrispondenti agli ingressi di una porta sono tutte 0 o 1, sarà

necessariamente 0 o 1 anche la variabile all'uscita della porta. Detto questo, è facile vedere che ad un circuito che calcola il valore `true` corrisponde un sistema di disequazioni soddisfacibile e viceversa.

Esercizio 2.2. Nella dimostrazione di cui sopra, una porta OR è stata ridotta a porte AND e NOT attraverso la legge di De Morgan. Mostrare invece come associare direttamente alla porta delle disequazioni che ne specifichino il corretto funzionamento, in analogia ai casi AND e NOT.