Indice

Capitolo 9

Trattabilità ed intrattabilità dei problemi

Dal punto di vista delle applicazioni pratiche l'analisi della complessità dei problemi è volta ad individuare per quali problemi sia possibile determinare le relative soluzioni in modo efficiente, cioè con un costo computazionale che cresce al più in modo polinomiale rispetto alla taglia dell'input, e per quali problemi, al contrario, è invece necessario utilizzare algoritmi di costo esponenziale, o comunque super-polinomiale. I problemi del primo tipo sono detti *trattabili* e quelli del secondo tipo *intrattabili*.

In questo capitolo prendiamo in considerazione, in particolare, alcune classi di complessità più direttamente legate alla determinazione della trattabilità o intrattabilità computazionale: le classi P, NP e PSPACE. Come vedremo, anche se (vedi Sezione 8.7) è possibile mostrare facilmente delle relazioni di inclusione tra tali classi, tuttavia, per ogni coppia di tali classi, il problema di determinare se l'inclusione è propria o meno, vale a dire se le due classi coincidono o no, è un problema aperto. Il più importante di tali problemi, probabilmente il più rilevante problema aperto nella teoria degli algoritmi, si chiede se le classe P dei problemi decidibili in tempo polinomiale mediante una macchina di Turing deterministica e la classe NP dei problemi accettabili in tempo polinomiale da una macchina di Turing non deterministica coincidono o meno.

9.1 La classe P

La classe P viene tradizionalmente considerata come una ragionevole caratterizzazione formale dei problemi trattabili.

Ciò è dovuto alle seguenti osservazioni:

1. La crescita di funzioni non polinomiali (ad esempio esponenziali) è così rapida che i valori diventano rapidamente troppo grandi. Questo può essere osservato in Figura 9.1, dove viene mostrato il numero di passi eseguiti da algoritmi con diverse complessità temporali, al crescere della dimensione dell'istanza.

   Assumendo che sia utilizzata una macchina che esegue un passo di computazione in 10 nanosecondi (10^{-8} secondi), otteniamo per gli stessi algoritmi i tempi di esecuzione in Figura 9.2 (si noti che l'età dell'universo viene stimata tra 10^{10} e $2 \cdot 10^{10}$ anni).

2. Per n sufficientemente piccolo, un algoritmo polinomiale può eseguire più passi di uno esponenziale, implicando quindi che eventualmente un algoritmo esponenziale potrebbe fornire migliori prestazioni di uno polinomiale su tutte le istanze trattate "in pratica". Comunque, tale fenomeno è piuttosto raro, anche in conseguenza del fatto che in quasi tutti i casi gli algoritmi polinomiali individuati hanno una complessità limitata da un polinomio di grado basso (raramente maggiore di 5) e con costanti moltiplicative piccole. In conseguenza di ciò, per praticamente tutte le istanze di interesse pratico, un algoritmo polinomiale richiede molto meno tempo di calcolo di uno non polinomiale.

3. Gli algoritmi polinomiali sono gli unici per i quali eventuali avanzamenti tecnologici comportano la possibilità di trattare, negli stessi tempi, istanze di dimensioni significativamente maggiori. In Figura 9.3 viene illustrato come varino le dimensioni delle istanze trattabili nello stesso tempo di computazione per algoritmi di diversa complessità, nei casi in cui il singolo passo di computazione possa essere eseguito in tempi più rapidi di un fattore (10, 1000, 10^6, 10^9) o, equivalentemente, se si assume che vengano usate macchine parallele con 10, 1000, 10^6, 10^9 processori (senza costi aggiuntivi di comunicazione tra i processori stessi).

	$n = 2$	$n = 5$	$n = 10$	$n = 100$	$n = 1000$
n	2	5	10	100	1000
n^2	4	25	100	10^4	10^6
n^3	8	125	1000	10^6	10^9
n^6	64	15625	10^6	10^{12}	10^{18}
2^n	4	32	$\approx 10^3$	$\approx 10^{30}$	$\approx 10^{300}$
2^{n^2}	16	$\approx 3 \cdot 10^7$	$\approx 10^{30}$	$\approx 10^{3000}$	$\approx 10^{300000}$

Figura 9.1 Passi eseguiti da algoritmi con diverse complessità

Tipici problemi di complessità polinomiale, che vengono quindi efficientemente risolti dai calcolatori nelle applicazioni quotidiane, sono i problemi di ordina-

	$n = 2$	$n = 5$	$n = 10$	$n = 100$	$n = 1000$
n	20 ns	50 ns	100 ns	1 μs	10 μs
n^2	40 ns	250 ns	1 μs	100 μs	10 ms
n^3	80 ns	1.25 μs	10 μs	10 μs	10 sec
n^6	640 ns	150 μs	10 ms	$\approx$ 3 ore	$\approx$ 300 anni
2^n	40 ns	320 ns	$\approx$ 10 μs	$\approx 10^{14}$ anni	$\approx 10^{284}$ anni
2^{n^2}	160 ns	0.3 sec	$\approx 10^{14}$ anni	$\approx 10^{3000}$ anni	$\approx 10^{300000}$ anni

Figura 9.2 Tempi di calcolo di algoritmi con diverse complessità

	t	$t/10$	$t/1000$	$t/10^6$	$t/10^9$
n	N	$10 \cdot N$	$1000 \cdot N$	$N \cdot 10^6$	$N \cdot 10^9$
n^2	N	$\approx 3.16 \cdot N$	$\approx 31.6 \cdot N$	$1000 \cdot N$	$\approx N \cdot 31600$
n^3	N	$\approx 2.15 \cdot N$	$10 \cdot N$	$100 \cdot N$	$1000 \cdot N$
n^6	N	$\approx 1.46 \cdot N$	$\approx 3.16 \cdot N$	$10 \cdot N$	$\approx 31.6 \cdot N$
2^n	N	$\approx N + 3.32$	$\approx N + 9.96$	$\approx N + 19.93$	$\approx N + 29.89$
2^{n^2}	N	$< N + 1.82$	$< N + 3.15$	$< N + 4.46$	$< N + 5.46$

Figura 9.3 Aumento della dimensione delle istanze trattabili in uno stesso limite di tempo ottenuto da incrementi di velocità di calcolo

mento, di interrogazione di basi di dati, di ricerca di percorsi di costo minimo in grafi, di programmazione lineare, etc.

Si noti che, allo stato attuale delle conoscenze, è un problema aperto se tutti i problemi appartenenti alla classe P abbiano effettivamente un livello di complessità equivalente. Ad esempio, mentre sappiamo che alcuni problemi in P sono risolubili in spazio logaritmico nella dimensione dell'input, per altri problemi, che potrebbero quindi essere più difficili, non si sa se tale proprietà sia verificata, nel senso che, anche se non è stato derivato un lower bound super-logaritmico sulla quantità di spazio necessario per la soluzione, non è stato però individuato alcun algoritmo di soluzione che usi spazio logaritmico.

Le riducibilità polinomiali rappresentano tuttavia uno strumento eccessivamente potente per riuscire a discriminare la difficoltà di soluzione di problemi diversi in P. Infatti, non è difficile rendersi conto che, per ogni coppia di problemi $\mathcal{P}_1, \mathcal{P}_2 \in$ P, $\mathcal{P}_1 \leq_m^p \mathcal{P}_2$ e $\mathcal{P}_2 \leq_m^p \mathcal{P}_1$.

Al fine di caratterizzare la complessità relativa di problemi in P, facciamo riferimento ad un tipo di riducibilità meno potente, una versione di Karp-riducibilità vincolata ad operare in spazio logaritmico.

Definizione 9.1 *Dati due problemi $\mathcal{P}_1, \mathcal{P}_2$, diciamo che $\mathcal{P}_1$ è log-space Karp-riducibile a $\mathcal{P}_2$ ($\mathcal{P}_1 \leq_m^{logsp} \mathcal{P}_2$) se e solo se $\mathcal{P}_1$ è Karp–riducibile a $\mathcal{P}_2$ e la riduzione relativa $\mathcal{R}$ è un algoritmo calcolabile in spazio logaritmico.*

Dato che un algoritmo che termina utilizzando spazio logaritmico non può richiedere tempo più che polinomiale: da ciò deriva che una Karp-riduzione *log*-space è anche polinomiale (ma non viceversa), per cui la Karp-riducibilità *log*-space è in effetti meno potente della Karp-riducibilità polinomiale.

È quindi possibile individuare problemi P-completi (rispetto alla Karp-riducibilità *log*-space): tali problemi, pur essendo risolubili comunque in tempo polinomiale, sono comunque in un qualche senso, in virtù della loro completezza, i più "difficili" tra i problemi in P. Si noti che la maggiore difficoltà rappresentata dalla relazione di Karp-riducibilità *log*-space non ha nulla a che fare con la complessità di risoluzione dei problemi: in altri termini, il fatto che un problema $\mathcal{P}_1$ è più difficile di un altro problema $\mathcal{P}_2$ in quanto $\mathcal{P}_2 \leq_m^{logsp} \mathcal{P}_1$ è cosa ben diversa dalla maggiore difficoltà rappresentata dal fatto che $\mathcal{P}_1$ è risolubile in tempo $\Theta(n^k)$, mentre $\mathcal{P}_2$ è risolubile in tempo $\Theta(n^h)$, con $k > h$.

Esercizio 9.1 Dimostrare che la relazione di Karp-riducibilità *log*-space è transitiva.

Come corollario immediato di quanto detto sopra consegue la proprietà che se un problema P-completo è risolubile in spazio logaritmico, allora P = LOGSPACE.

Un'altra caratteristica rilevante dei problemi P-completi deriva dalla seguente osservazione: definito un problema come *efficientemente parallelizzabile* se ogni istanza di dimensione n è risolubile da $O(n^k)$ processori operanti in parallelo in tempo $O(\log^h n)$, con h, k costanti opportune, allora è possibile mostrare che, dati due problemi $\mathcal{P}_1, \mathcal{P}_2$, se $\mathcal{P}_1$ è efficientemente parallelizzabile e $\mathcal{P}_2 \leq_m^{logsp} \mathcal{P}_1$, allora anche $\mathcal{P}_2$ è efficientemente parallelizzabile.[1]

Da tale proprietà deriva che il mostrare che un qualche problema P-completo è efficientemente parallelizzabile comporta provare che *ogni* problema in P è parallelizzabile in modo efficiente.

9.2 P-completezza

La P-completezza di un problema di decisione $\mathcal{P} \in$ P viene generalmente mostrata individuando una Karp riduzione *log*-space da qualche altro problema P-completo $\mathcal{P}_1$. Infatti, dato che per ogni $\mathcal{P}_2 \in$ P si ha per definizione che $\mathcal{P}_2 \leq_m^{logsp} \mathcal{P}_1$, mostrare che $\mathcal{P}_1 \leq_m^{logsp} \mathcal{P}$ comporta, per la transitività della Karp-riducibilità *log*-space (vedi Esercizio 9.1), che $\mathcal{P}_2 \leq_m^{logsp} \mathcal{P}$, e quindi che $\mathcal{P}$ è P-completo. È allora necessario identificare, mediante qualche altra tecnica, un problema P-completo *iniziale*.

Prima di introdurre il problema Valore Calcolato da Circuito, che utilizzeremo come problema P-completo iniziale (vedi Teorema 9.2), consideriamo alcune definizioni preliminari.

[1] La classe dei problemi efficientemente parallelizzabili è comunemente indicata come NC, e gode della proprietà di chiusura rispetto alla Karp-riducibilità *log*-space.

Definizione 9.2 *Un* circuito booleano *è un grafo orientato aciclico in cui ogni nodo ha* $d_{in} \leq 2$ *archi entranti. In particolare:*

1. *i nodi aventi* $d_{in} = 0$ *sono detti* nodi di input*;*
2. *i nodi aventi* $d_{in} = 1$ *sono detti* porte NOT*;*
3. *ogni nodo avente* $d_{in} = 2$ *è una* porta OR *oppure una* porta AND.

Tutti i nodi hanno $d_{out} > 0$ *archi uscenti, eccetto un solo* nodo di output.

La *dimensione* del circuito è data dal numero di nodi del circuito stesso, mentre la sua *profondità* è pari alla lunghezza del più lungo cammino da un nodo di input al nodo di output.

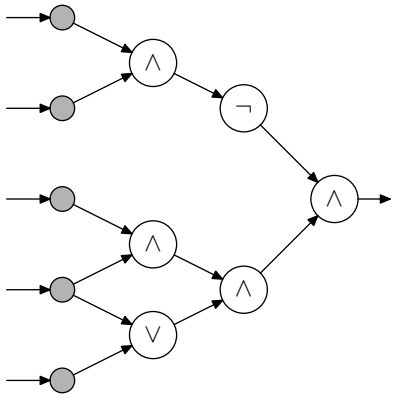

FIGURA 9.4 Esempio di circuito.

Sia $I(\mathcal{C})$, con $|I(\mathcal{C})| = n$, l'insieme dei nodi input di un circuito $\mathcal{C}$: data allora una qualunque assegnazione $\langle b_0, b_1, \ldots, b_{n-1}\rangle \in \{0,1\}^n$ di valori booleani[2] agli n nodi di input, il circuito calcola, applicando su ogni porta l'operatore ad essa associato ai relativi input, i valori booleani in uscita su tutti i nodi del circuito stesso. In particolare , $\mathcal{C}$ calcola la funzione $f_{\mathcal{C}} : \{0,1\}^n \mapsto \{0,1\}$ tale che $f_{\mathcal{C}}(b_0, b_1, \ldots, b_{n-1}) = 1$ se e solo se, con input $b_0, b_1, \ldots, b_{n-1}$, il circuito $\mathcal{C}$ restituisce il valore 1 in uscita al nodo di output.

Esempio 9.1 Il circuito booleano in Figura 9.4 ha 5 nodi di input e 6 porte, delle quali la più a destra è anche nodo di output. Se indichiamo con $b_0, \ldots, b_4$ i valori booleani associati ai nodi di input (dall'alto in basso), il circuito booleano calcola la funzione $(\overline{b_0 \wedge b_1}) \wedge ((b_2 \wedge b_3) \wedge (b_3 \vee b_4))$.

[2]Nel seguito di questa sezione, indicheremo i valori booleani come 0 (per FALSO) e (per VERO).

Definizione 9.3 *Dato un circuito $\mathcal{C}$ con $|I(\mathcal{C})|= n$, il* linguaggio deciso *da $\mathcal{C}$ è l'insieme delle stringhe in $w \in \{0,1\}^n$ tali che $f_{\mathcal{C}}(w) = 1$.*

Possiamo ora definire il problema di decisione Valore Calcolato da Circuito, il quale chiede, dato un circuito booleano ed una sua stringa di input, se il circuito restituisce il valore 1.

Valore Calcolato da Circuito
Istanza: Circuito booleano $\mathcal{C}$ con nodi di input $I(\mathcal{C})$ ($\|I(\mathcal{C})\|= n$), stringa $w \in \{0,1\}^n$.
Predicato: Si ha $f_{\mathcal{C}}(w) = 1$?

Al fine di dimostrare che Valore Calcolato da Circuito è P-completo, mostriamo ora il seguente teorema, che ci dice che ogni linguaggio in P può essere accettato da circuiti di dimensione polinomiale.

Teorema 9.1 *Sia $L \in \{0,1\}^n$ un linguaggio tale che $L \in P$; allora esiste un circuito $\mathcal{C}$ con n nodi di input e di dimensione e profondità polinomiali in n che decide L.*

Dimostrazione. Se $L \in$ P allora esiste una macchina di Turing $\mathcal{M} = \langle \Gamma, \b, Q, q_0, F, \delta \rangle$, con alfabeto di input $\Sigma = \{0,1\} \subseteq \Gamma = \{a_1, a_2, \ldots, a_{\|\Gamma\|}\}$, che decide ogni stringa $w \in L$ in al più $p(n)$ passi, dove $p(n)$ è un polinomio. Assumiamo che $\mathcal{M}$ operi su un solo nastro semi-infinito (vedi Teorema 5.3), nelle cui prime n celle l'input è inizialmente contenuto.

Consideriamo, senza perdita di generalità, la macchina di Turing $\mathcal{M}' = \langle \Gamma, \b, Q, q_0, F, \delta' \rangle$, la cui funzione di transizione è definita nel modo seguente:

$$\delta'(q,a) = \begin{cases} \delta(q,a) \text{ se } \delta(q,a) \text{ è definita} \\ (q,a,i) \text{ altrimenti.} \end{cases}$$

La macchina di Turing $\mathcal{M}'$ ha lo stesso comportamento di $\mathcal{M}$ eccetto che ad ogni computazione massimale di $\mathcal{M}$ corrisponde una computazione di $\mathcal{M}'$ che cicla indefinitamente sulla stessa configurazione. Quindi, ad ogni computazione accettante di $\mathcal{M}$ corrisponde una computazione di $\mathcal{M}'$ che cicla su una configurazione di accettazione, mentre ad ogni computazione non accettante di $\mathcal{M}$ corrisponde una computazione di $\mathcal{M}'$ che cicla su una configurazione non di accettazione. Dato che $\mathcal{M}$ decide ogni stringa in al più $p(n)$ passi, ne consegue che, data una stringa w in input, la configurazione raggiunta da $\mathcal{M}'$ dopo avere eseguito $p(n)$ passi è di accettazione se $w \in L$ e di non accettazione altrimenti.

Evidentemente, in $p(n)$ passi la testina di $\mathcal{M}'$ può al massimo posizionarsi sulle prime $p(n)+1$ celle di nastro, che quindi rappresentano l'unica parte del nastro di interesse nella computazione.

Si consideri allora una matrice (*tableau*) T di dimensioni $(p(n)+1) \times (p(n)+1)$, dove ogni riga corrisponde ad un istante di tempo (e quindi ad una configurazione) ed ogni colonna ad una particolare cella di nastro. Ogni elemento di T può contenere una coppia in $\bar{\Gamma} \times (Q \cup \{\perp\})$, dove $\perp \notin Q$.

La riga i-esima del tableau descrive nel modo seguente la configurazione di $\mathcal{M}'$ dopo che è stato eseguito l'i-esimo passo di computazione. Sia $\alpha_j \in \bar{\Gamma}$ il contenuto della cella j-esima di nastro $(0 \leq j \leq p(n))$, sia q_k lo stato attuale di $\mathcal{M}'$ e si assuma che la testina sia posizionata sulla cella t-esima: la riga i-esima di T ha allora la forma

$$\langle \alpha_0, \perp \rangle, \langle \alpha_1, \perp \rangle, \ldots, \langle \alpha_{t-1}, \perp \rangle, \langle \alpha_t, q_k \rangle, \langle \alpha_{t+1}, \perp \rangle, \ldots, \langle \alpha_{p(n)}, \perp \rangle.$$

Si noti che, dato il tableau T rappresentante la computazione effettuata da $\mathcal{M}'$ su input w, è possibile determinare se $w \in L$ semplicemente osservando se la configurazione rappresentata nell'ultima riga di T è di accettazione, e quindi se per l'unica cella in tale riga che contiene una coppia il cui secondo elemento è diverso da $\perp$, tale elemento è un qualche stato finale $q_F \in F$. È inoltre importante osservare che, per come è definita la funzione di transizione di una macchina di Turing, il contenuto della cella $T[i,j]$ del tableau è determinato (attraverso la funzione di transizione stessa) soltanto dal contenuto delle tre celle $T[i-1, j-1]$, $T[i-1, j]$ e $T[i-1, j+1]$.

Il circuito $\mathcal{C}$ essenzialmente si limita a codificare, attraverso opportuni blocchi di porte, tale dipendenza del contenuto di una cella di T dalle tre celle della riga precedente. Il circuito calcola, in particolare, il seguente insieme di funzioni:

1. $S(i,j,k)$ $(0 \leq i \leq p(n), 0 \leq j \leq p(n), 0 \leq k \leq |\, Q \,|)$, definita come $S(i,j,k) = 1$ se e solo se il secondo componente del contenuto di $T[i,j]$ è $\perp$, se $k = |\, Q \,|$, o q_k se $k < |\, Q \,|$.

2. $C(i,j,k)$ $(0 \leq i \leq p(n), 0 \leq j \leq p(n), 0 \leq k \leq |\, \Gamma \,|)$, definita come $S(i,j,k) = 1$ se e solo se il primo componente del contenuto di $T[i,j]$ è $\bar{b}$, se $k = 0$, o a_k se $k > 0$.

Il circuito $\mathcal{C}$ ha dimensione $O((p(n))^2)$ e profondità $O(p(n))$, ed è strutturato su tre sezioni (di input, di calcolo, di output) nel modo seguente (vedi Figura 9.5).

1. La sezione di input è composta da una riga di n nodi di input corrispondenti agli n caratteri $\{0,1\}$ in input, e da una seconda riga comprendente $p(n)+1$ blocchi di porte, ognuno dei quali è associato ad una cella del nastro di $\mathcal{M}'$. Il primo blocco $B_{0,0}$ è collegato al primo nodo di input, corrispondente al primo carattere b_0 di w, e fornisce in output i valori delle funzioni $S(0,0,k)$ (dove $S(0,0,0) = 1$ e $S(0,0,k) = 0$ per $k > 0$) e delle funzioni $C(0,0,k)$ (dove $C(0,0,k) = 1$ se $b_0 = a_k$ e $C(0,0,k) = 0$ altrimenti).

Ogni blocco $B_{0,j}$, con $1 \leq j \leq n-1$, è collegato al nodo di input corrispondente al j-esimo carattere b_0 di w, e fornisce in output i valori delle funzioni $S(0,j,k)$ (dove $S(0,j,k) = 0$ per $k \geq 0$) e delle funzioni $C(0,j,k)$ (dove $C(0,j,k) = 1$ se $b_j = a_k$ e $C(0,j,k) = 0$ altrimenti).

Ogni blocco $B_{0,j}$, con $n \leq j \leq p(n)$, è collegato al nodo di input corrispondente all'n-esimo carattere b_{n-1} di w, e fornisce in output, in modo del tutto indipendente dal valore di b_{n-1}, i valori delle funzioni $S(0,j,k)$ (dove $S(0,j,k) = 0$ per $k \geq 0$) e delle funzioni $C(0,j,k)$ (dove $C(0,j,0) = 1$ e $C(0,j,k) = 0$ per $k > 0$).

Si osservi che i blocchi $B_{0,j}$ con $1 \leq j \leq n-1$ calcolano la stessa funzione e quindi sono identici tra loro, così come i blocchi $B_{0,j}$ con $n \leq j \leq p(n)$; inoltre, la dimensione di tali blocchi è chiaramente indipendente da n. Da ciò deriva che la sezione di input ha dimensione $O(p(n))$ e profondità $O(1)$.

2. La sezione di calcolo è composta da $p(n)$ righe (una per ogni passo di computazione di $\mathcal{M}'$). La riga i-esima è composta da $p(n)+1$ blocchi identici (uno per ogni cella del nastro di $\mathcal{M}'$): il blocco $B_{i,j}$ determina il contenuto della cella $T[i,j]$ del tableau in funzione del contenuto delle tre celle $T[i-1,j-1]$, $T[i-1,j]$ e $T[i-1,j+1]$.[3]

   In particolare, il blocco $B_{i,j}$ ha una struttura dipendente dalla funzione di transizione δ' e riceve in input i valori delle funzioni $S(i-1,t,k)$ $(j-1 \leq t \leq j+1,\ 0 \leq k \leq |\, Q \,|)$ e $C(i-1,t,k)$ $(j-1 \leq t \leq j+1,\ 0 \leq k \leq |\, \Gamma \,|)$. A partire da tali valori, $B_{i,j}$ calcola le funzioni $S(i,j,k)$ $(0 \leq k \leq |\, Q \,|)$ e $C(i,j,k)$ $(0 \leq k \leq |\, \Gamma \,|)$.

   Anche la dimensione dei blocchi nella sezione di calcolo è indipendente da n, per cui abbiamo che la sezione di calcolo ha dimensione $O(p(n)^2)$ e profondità $O(p(n))$.

3. La sezione di output è composta da una prima riga di $p(n)+1$ blocchi identici, ognuno associato ad una cella del nastro di $\mathcal{M}'$. Il blocco $B_{p(n)+1,j}$ restituisce un solo valore booleano, pari ad 1 se e solo se il secondo componente del contenuto di $T[p(n),j]$ è uno stato finale $q_F \in F$. Il blocco $B_{p(n)+1,j}$ riceve quindi in input i valori delle funzioni $S(p(n),j,k)$ $(0 \leq k \leq |\, Q \,|)$ e restituisce un valore pari a $\bigvee_{q_k \in F} S(p(n),j,k)$. Si osservi che tutti i blocchi su tale riga sono identici ed hanno dimensione indipendente da n; si osservi inoltre che la configurazione raggiunta è di accettazione se e solo se esiste uno dei blocchi $B_{p(n)+1,j}$ che restituisce valore 1.

   La seconda riga della sezione di output è composta da $\lceil n/2 \rceil$ porte OR $B_{p(n)+2,j}$ ognuna delle quali riceve in input i valori calcolati dai due

[3] Naturalmente, se $j = 0$ la dipendenza è solo dalle celle $T[i-1,0]$ e $T[i-1,1]$, mentre se se $j = p(n)$ la dipendenza è solo dalle celle $T[i-1,p(n)-1]$ e $T[i-1,p(n)]$.

blocchi $B_{p(n)+1,2j}$ e $B_{p(n)+1,2j+1}$. La terza riga, composta da $\lceil n/4 \rceil$ porte OR opera allo stesso modo sui valori restituiti dai blocchi della riga precedente, e così via. La $\lceil \log_2(p(n)+1) \rceil$-esima riga ha una sola porta OR, definita come nodo di output del circuito, che calcola il valore 1 se e solo se la stringa viene accettata da $\mathcal{M}$ in tempo al più $p(n)$.

Chiaramente, la sezione di output ha dimensione $O(p(n))$ e profondità $O(\log(p(n))) = O(\log n)$.

Per concludere, osserviamo che, come si voleva dimostrare, il circuito $\mathcal{C}$ così costruito ha dimensione $O(p(n)^2)$ e profondità $O(p(n))$, quindi polinomiali in n, e, ricevendo in input una stringa $w \in \{0,1\}^n$, restituisce valore 1 se e solo se $w \in L$. □

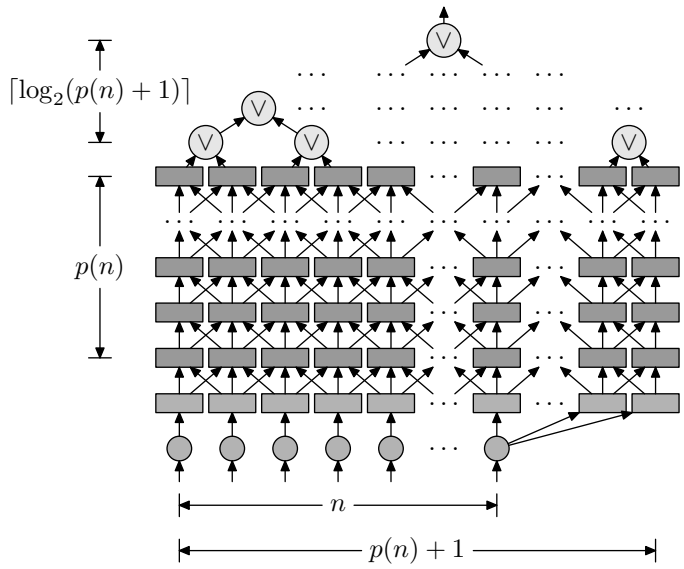

FIGURA 9.5 Struttura del circuito nella dimostrazione del Teorema 9.1.

Possiamo ora mostrare che il problema VALORE CALCOLATO DA CIRCUITO è tale che ogni problema in P può essere ridotto ad esso mediante una Karp-riduzione *log*-space.

Teorema 9.2 *Il problema* VALORE CALCOLATO DA CIRCUITO *è* P*-completo rispetto alla Karp-riducibilità* log*-space.*

Dimostrazione. Per mostrare che VALORE CALCOLATO DA CIRCUITO è in P basta osservare che, per la aciclicità del circuito, è possibile, mediante una

visita in ordine topologico dei nodi, determinare il valore in uscita da tutte le porte, e quindi anche il valore in output, in tempo lineare nella dimensione del circuito.

Il Teorema 9.1 mostra poi una riduzione da ogni problema in P a VALORE CALCOLATO DA CIRCUITO. L'ultimo passo della dimostrazione, che lasciamo come esercizio (vedi Esercizio 9.2) comporta la verifica che tale riduzione è *log*-space. □

Esercizio 9.2 Dimostrare che la riduzione da un qualunque problema in P a VALORE CALCOLATO DA CIRCUITO mostrata nel Teorema 9.1 richiede spazio logaritmico.

Dalla P-completezza di VALORE CALCOLATO DA CIRCUITO sopra dimostrata deriva quindi che se tale problema fosse efficientemente parallelizzabile[4] allora così sarebbe ogni problema risolubile in tempo polinomiale.

Un aspetto interessante nella dimostrazione precedente è che, come vedremo, la sua struttura di fondo viene ripresa nella dimostrazione del Teorema di Cook (Teorema 9.4) che enuncia un corrispondente risultato per la classe NP, identificando un primo problema NP-completo, così come il Teorema 9.2 identifica un primo problema P-completo.

Un ulteriore esempio di problema P-completo è fornito dal seguente problema SODDISFACIBILITÀ DI FORMULE DI HORN, per introdurre il quale forniamo dapprima la seguente definizione.

Definizione 9.4 *Una clausola di Horn su un insieme X di variabili booleane è una disgiunzione $C = \overline{x}_1 \vee x_2 \vee \ldots x_t$ di termini relativi a tutte o a parte delle variabili in X, con il vincolo che in esattamente uno di tali termini la variabile corrispondente compare vera, mentre in tutti gli altri compare negata.*

Si osservi che, dato che per l'implicazione logica vale l'equivalenza tra $x \vee \overline{y}$ e $y \Rightarrow x$, abbiamo che ogni clausola di Horn corrisponde ad una implicazione.

Osserviamo inoltre che una implicazione del tipo $x \wedge y \wedge z \Rightarrow w$ corrisponde alla clausola di Horn $\overline{x} \vee \overline{y} \vee \overline{z} \vee w$, e che una implicazione $x \vee y \vee z \Rightarrow w$ corrisponde alla formula $(\overline{x} \wedge \overline{y} \wedge \overline{z}) \vee w$ e quindi, equivalentemente, all'insieme di clausole di Horn $\overline{x} \vee w$, $\overline{y} \vee w$, $\overline{z} \vee w$.

Definizione 9.5 *Una formula di Horn $\mathcal{F}_H$ su un insieme X di variabili booleane è una congiunzione $\mathcal{F}_H = C_1 \wedge C_2 \wedge \ldots \wedge C_t$ di clausole di Horn definite su X.*

Si noti che una formula di Horn non è altro che u caso speciale di formula CNF, ed in effetti il problema SODDISFACIBILITÀ DI FORMULE DI HORN non

[4] Si osservi che VALORE CALCOLATO DA CIRCUITO è un problema che per sua natura si presenta come "parallelo", in quanto le porte in un circuito booleano operano in condizioni di parallelismo. La profondità del circuito descritto in una istanza di VALORE CALCOLATO DA CIRCUITO non è però necessariamente polinomiale nel logaritmo della sua dimensione.

è altro che il problema SODDISFACIBILITÀ ristretto al caso particolare in cui si considerino formule di Horn.

SODDISFACIBILITÀ DI FORMULE DI HORN

ISTANZA: Una formula di Horn $\mathcal{F}_H$ su un insieme X di variabili booleane

PREDICATO: Esiste una assegnazione $f : X \mapsto \{\text{VERO}, \text{FALSO}\}$ che soddisfa $\mathcal{F}_H$?

Teorema 9.3 *Il problema* SODDISFACIBILITÀ DI FORMULE DI HORN *è* P*-completo rispetto alla Karp-riducibilità* log*-space.*

Dimostrazione. Come prima cosa mostriamo che il problema SODDISFACIBILITÀ DI FORMULE DI HORN è in P: a tal fine, è possibile derivare un algoritmo polinomiale che verifica se una formula di Horn $\mathcal{F}_H$ è soddisfacibile (e che anzi, in tal caso, costruisce l'assegnamento di verità corrispondente) semplicemente osservando che:

- se una clausola ha un solo termine, che quindi è una variabile vera, allora tale variabile deve essere posta a VERO;
- se in una clausola con più di un termine tutte le variabili che compaiono negate sono state poste a VERO, allora la variabile che compare vera deve anch'essa essere posta a VERO.

Per mostrare la P-completezza di SODDISFACIBILITÀ DI FORMULE DI HORN introduciamo il seguente problema, restrizione di VALORE CALCOLATO DA CIRCUITO al caso in cui si considerino circuiti privi di porte NOT.

VALORE CALCOLATO DA CIRCUITO MONOTONO

ISTANZA: Circuito booleano monotono (privo di porte NOT) $\mathcal{C}$, con nodi di input $I(\mathcal{C})$ ($|I(\mathcal{C})| = n$), stringa $w \in \{0,1\}^n$.

PREDICATO: Si ha $f_{\mathcal{C}}(w) = 1$?

Il problema VALORE CALCOLATO DA CIRCUITO MONOTONO può essere facilmente mostrato essere P-completo (vedi Esercizio 9.4).

Mostriamo ora che esiste una Karp-riduzione *log*-space da VALORE CALCOLATO DA CIRCUITO MONOTONO a SODDISFACIBILITÀ DI FORMULE DI HORN. La riduzione opera nel modo seguente: l'insieme X delle variabili corrisponde all'insieme dei nodi del circuito. L'insieme delle clausole viene derivato nel modo seguente:

1. ad ogni nodo di input, cui è associata ad esempio la variabile x_i, se al nodo viene assegnato il valore 1 in input, introduciamo la clausola x_i, altrimenti, se al nodo viene assegnato il valore 0 in input, introduciamo la clausola $\overline{x}_i$;

2. per ogni porta AND, siano x_i e x_j le variabili associate alle due porte da cui essa riceve i propri input: se x_k è la variabile ad essa associata, alla porta corrispondo le tre clausole $\overline{x}_i \vee \overline{x}_j \vee x_k$, $x_i \vee \overline{x}_k$ e $x_j \vee \overline{x}_k$;

3. per ogni porta OR, siano x_i e x_j le variabili associate alle due porte da cui essa riceve i propri input: se x_k è la variabile ad essa associata, alla porta corrispondono le tre clausole $x_i \vee x_j \vee \overline{x}_k$, $\overline{x}_i \vee x_k$ e $\overline{x}_j \vee x_k$;

4. sia x_t la variabile associata al nodo di output: introduciamo allora la clausola x_t.

Non è difficile verificare che la formula di Horn derivata in tal modo è soddisfacibile se e solo l'istanza corrispondente di Valore Calcolato da Circuito è una istanza positiva del problema. In particolare, il valore assunto da ogni variabile nell'assegnazione di verità che soddisfa la formula specifica il valore calcolato dalla porta corrispondente del circuito.

Infine, per verificare che la riduzione è *log*-space basta osservare che per derivare la formula di Horn è sufficiente considerare i nodi del circuito uno alla volta, tenendo traccia dell'indice della relativa variabile e degli indici di eventuali variabili associate ai suoi input. Dato che ognuno di tali indici occupa spazio logaritmico nella dimensione del circuito, e che ogni volta che viene considerato un nodo è necessario rappresentare un numero costante di indici, ne deriva che la riduzione richiede spazio logaritmico. □

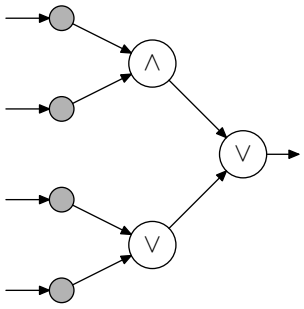

Figura 9.6 Esempio di circuito monotono.

Esempio 9.2 Si consideri l'istanza di Valore Calcolato da Circuito Monotono rappresentata dal circuito in Figura 9.6 e dai valori in input 1001. Se consideriamo la riduzione a Soddisfacibilità di formule di Horn descritta nella dimostrazione del Teorema 9.3 possiamo associare ai nodi di input le variabili x_1, x_2, x_3, x_4, ed alle 3 porte le variabili x_5, x_6, x_7. Per quanto riguarda le clausole, otteniamo quanto segue:

1. i quattro nodi di input danno luogo alle quattro clausole x_1, $\overline{x}_2$, $\overline{x}_3$, x_4;
2. la porta AND al primo livello dà luogo alle tre clausole $\overline{x}_1 \vee \overline{x}_2 \vee x_5$, $x_1 \vee \overline{x}_5$ e $x_2 \vee \overline{x}_5$;
3. la porta OR al primo livello dà luogo alle tre clausole $x_3 \vee x_4 \vee \overline{x}_6$, $\overline{x}_3 \vee x_6$ e $\overline{x}_4 \vee x_6$;
4. la porta AND al secondo livello dà luogo alle tre clausole $\overline{x}_5 \vee \overline{x}_6 \vee x_7$, $x_5 \vee \overline{x}_7$ e $x_6 \vee \overline{x}_7$;
5. il nodo di output corrisponde alla porta AND sul secondo livello, il che dà luogo alla clausola x_7.

Si può verificare che l'unico assegnamento di verità che soddisfa la formula di Horn risultante è dato da $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 1, x_5 = 0, x_6 = 1, x_7 = 1$, che, come si può vedere, fornisce i valori calcolati dalle varie porte del circuito in presenza dell'input considerato.

Esercizio 9.3 Definire un algoritmo operante in tempo polinomiale che risolva il problema SODDISFACIBILITÀ DI FORMULE DI HORN.

Esercizio 9.4 Dimostrare che la P-completezza del problema VALORE CALCOLATO DA CIRCUITO MONOTONO.
[Suggerimento: Utilizzare la regola di De Morgan $\overline{x \wedge y} = \overline{x} \vee \overline{y}$, $\overline{x \vee y} = \overline{x} \wedge \overline{y}$ per definire una riduzione da VALORE CALCOLATO DA CIRCUITO].

9.3 La classe NP

Molti problemi di decisione di significativa importanza presentano un insieme di caratteristiche comuni. Per tali problemi, infatti, non sono stati individuati algoritmi polinomiali per la loro soluzione, ma d'altra parte non è stata trovata alcuna dimostrazione di una loro esponenzialità: al tempo stesso, tutti questi problemi sono decidibili in tempo polinomiale da macchine di Turing non deterministiche.

Ad esempio, si può dimostrare che il problema SODDISFACIBILITÀ, definito nell'Esempio 8.8 è risolubile in tempo polinomiale da una macchina di Turing non deterministica, che accetta tutte e sole le istanze relative a formule soddisfacibili.

In generale, per ogni problema di decisione $\mathcal{P}$ si vuole, data una istanza x di $\mathcal{P}$ (codificata come stringa di simboli da un qualche alfabeto Σ), determinare un singolo valore binario che esprime il fatto che x appartenga o meno all'insieme $Y_{\mathcal{P}}$ delle istanze positive di $\mathcal{P}$. Come già osservato nella Sezione 8.2, nella maggior parte dei casi determinare se una istanza x appartiene a $Y_{\mathcal{P}}$ comporta l'individuare una soluzione $s(x)$ di x nel corrispondente problema di ricerca: ad esempio, determinare se una istanza di SODDISFACIBILITÀ rappresentata

da una formula CNF $\mathcal{F}$ su un insieme di variabili X è soddisfacibile equivale a chiedersi se esiste una soluzione rappresentata da una assegnazione di verità $f : X \mapsto \{\text{VERO}, \text{FALSO}\}$ che soddisfi la formula $\mathcal{F}$ stessa.

Esercizio 9.5 Mostrare che, data una istanza di SODDISFACIBILITÀ, la relativa soluzione del corrispondente problema di ricerca , se esiste, ha lunghezza polinomiale.

Osserviamo ora che, per ogni problema di decisione $\mathcal{P}$ che richiede di decidere se, data una istanza x del problema, tale istanza abbia una soluzione (e per ogni corrispondente problema di ricerca che, data x, chiede di determinare tale soluzione) è possibile considerare un corrispondente problema di *verifica* il quale, data una istanza x di $\mathcal{P}$ ed una possibile soluzione s, chiede se s sia effettivamente una soluzione di x. Nel caso di SODDISFACIBILITÀ ciò significa che, data una formula $\mathcal{F}$ su un insieme di variabili X ed una assegnazione di verità $f : X \mapsto \{\text{VERO}, \text{FALSO}\}$, si vuole determinare se f soddisfa $\mathcal{F}$.

Si osservi che, in realtà, tutte le considerazioni precedenti valgono ancora se al concetto di soluzione del problema sostituiamo quello, più generale, di certificato.

Definizione 9.6 *Dato un linguaggio $L \subseteq \Sigma^*$, una funzione $c : L \mapsto \Sigma^*$ associa ad ogni stringa $u \in L$ un* certificato *$v = c(u)$ se e solo se che esiste una macchina di Turing deterministica $\mathcal{M}$ la quale decide il linguaggio $L' = \{uv \mid u \in L, v = c(u)\}$.*

In altri termini, un certificato è una informazione aggiuntiva che "dimostra" che una data stringa appartiene al linguaggio L, e che può quindi essere utilizzata per decidere se una stringa u appartiene al linguaggio stesso. Come vedremo più avanti, l'utilizzo del concetto di certificato ci consentirà di formulare una definizione particolarmente utile di problema in NP.

In generale, verificare una soluzione sembra decisamente più semplice che risolvere il problema di decisione nella sua forma originaria. In particolare, si noti che verificare le soluzioni di un problema è equivalente a risolvere, mediante accettazione, il problema stesso in modo non deterministico: infatti, una macchina di Turing non deterministica potrebbe essere definita in modo tale da generare inizialmente, attraverso una sequenza opportuna di passi non deterministici, tutte le stringhe che potrebbero rappresentare possibili soluzioni, eventualmente in numero esponenziale, associando ad ogni possibile soluzione una distinta computazione.

Tale situazione è rappresentata in Figura 9.7, in cui, nella parte sinistra, viene mostrato come, non deterministicamente, vengono generate tutte le stringhe che potrebbero codificare possibili soluzioni (nodi ombreggiati), a partire da ognuno dei quali una computazione deterministica, nella parte destra, verifica se la stringa codifica effettivamente una soluzione dell'istanza.

Nell'ambito di ogni computazione, quindi, la macchina di Turing verifica deterministicamente la soluzione generata: evidentemente, se l'istanza del pro-

blema è una istanza positiva, esisterà una soluzione che soddisfa il predicato posto dal problema e tale soluzione corrisponderà ad uno dei cammini generati dalla macchina di Turing non deterministica, cammino che sarà quindi un cammino accettante. Inoltre, se per il problema considerato le relative soluzioni hanno dimensione polinomiale rispetto all'input la fase di generazione non deterministica di tutte le stringhe richiede tempo polinomiale e quindi, se il costo di verifica di una soluzione è anch'esso polinomiale, la macchina di Turing non deterministica opera in tempo polinomiale.

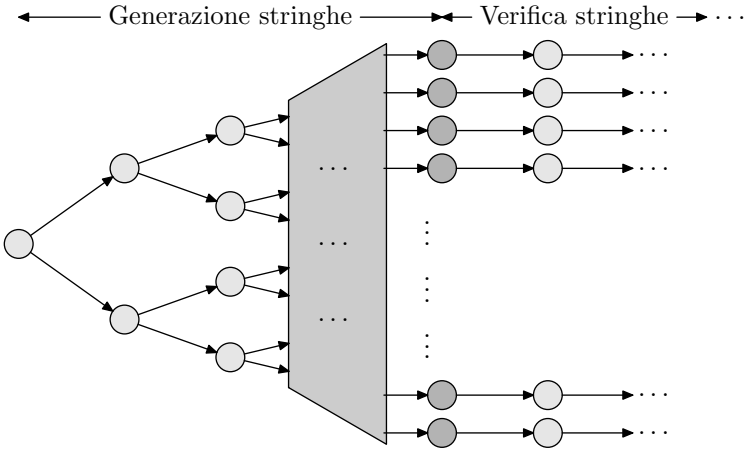

FIGURA 9.7 Schema di soluzione di un problema in NP mediante macchina di Turing non deterministica .

Tali considerazioni permettono di introdurre una diversa definizione di algoritmo (o macchina di Turing) non deterministico, equivalente al modello considerato fino ad ora. Secondo tale modello, un algoritmo non deterministico è un algoritmo che, oltre a tutte le istruzioni usuali tipiche del modello di calcolo adottato, ha la possibilità di eseguire comandi del tipo "'**guess** $y \in \{0,1\}$": un'istruzione di questo tipo, eseguita in un solo passo di computazione, fa sì che y assuma come valore un elemento in $\{0,1\}$.

Essenzialmente, un algoritmo non deterministico può essere interpretato come un algoritmo che ha la possibilità aggiuntiva di *indovinare* (guess) una continuazione nell'ambito di un insieme (finito) di possibili continuazioni della computazione eseguita fino al momento dell'esecuzione dell'istruzione *guess*. Si noti che per ogni insieme S, un elemento di S può essere "indovinato" per mezzo di $O(\log |S|)$ istruzioni "guess" sull'insieme $\{0,1\}$.

A questo punto, possiamo riformulare la definizione di algoritmo non de-

deterministico che risolve un problema di decisione, nell'ambito di questo nuovo modello, nel modo seguente:

Definizione 9.7 *Dato un problema di decisione $\mathcal{P}$, un algoritmo non deterministico $\mathcal{A}$ risolve $\mathcal{P}$ se e solo se, per ogni istanza $x \in Y_{\mathcal{P}}$, esiste almeno una sequenza di* guess *che fa sì che $\mathcal{A}$ restituisca il valore* VERO *e se, per ogni istanza $x \notin Y_{\mathcal{P}}$, non esiste alcuna sequenza di* guess *che fa sì che $\mathcal{A}$ restituisca il valore* VERO.

Si osservi che, senza perdita di generalità, possiamo sempre considerare un modello semplificato di algoritmo non deterministico polinomiale che esegue una sola istruzione *guess* su un insieme di taglia esponenziale, eseguita all'inizio della sua esecuzione. Tale operazione può essere vista come una unica *guess* complessiva dei risultati della sequenza (di lunghezza polinomiale) di *guess* binarie eseguite nell'ambito di ogni computazione non deterministica. In altri termini, assumiamo che le $f(n)$ *guess* binarie eseguite nell'ambito di una computazione siano tutte eseguite all'inizio, con una sola guess su un insieme di dimensione al più $2^{f(n)}$.

Consideriamo ad esempio il seguente problema INSIEME INDIPENDENTE.

INSIEME INDIPENDENTE

ISTANZA: Grafo $G = (V, E)$, $K \in \mathbb{N}$.

PREDICATO: Esiste un insieme indipendente in G di dimensione $\geq K$, cioè un sottoinsieme $U \subseteq V$ tale che $\mid U \mid \geq K$ e $\nexists (u, v) \in E$ con $u \in U$ and $v \in U$?

Ogni istanza positiva $x \in Y_{II}$ di INSIEME INDIPENDENTE ha (almeno) una soluzione associata $s(x)$ rappresentata da un sottoinsieme $U \subseteq V$ dell'insieme dei nodi, con $|U| \geq K$.

Mentre decidere se, dato un grafo G, esiste un insieme indipendente di taglia almeno K è un problema difficile (non è noto alcun algoritmo che lo risolva in tempo polinomiale), è possibile verificare in tempo lineare se un insieme $U \subseteq V$ di nodi è un insieme indipendente di dimensione almeno K.

Un algoritmo non deterministico (polinomiale) per risolvere il probelma INSIEME INDIPENDENTE è dato in Figura 9.1. Come si può osservare, l'algoritmo essenzialmente "indovina" un possibile insieme indipendente tra i $2^{|V|}$ possibili e quindi verifica se la *guess* ha avuto successo. Si noti che la descrizione "indovinata" di un insieme indipendente ha lunghezza polinomiale (lineare, in effetti) nella dimensione dell'input e la sua generazione richiede una quantità polinomiale (lineare) di *guess* binarie. Inoltre, dato che la fase di verifica richiede tempo polinomiale, l'algoritmo non deterministico presenta complessità polinomiale.

A questo punto, la classe NP può essere definita, in modo alternativo, come la classe di tutti i problemi che possono essere risolti da algoritmi deterministici che operano in tempo polinomiale a partire da *guess* di lunghezza polinomiale.

```
input G = (V, E): graph;
output boolean;
begin
  guess U ⊆ V;
  if | U |< K then return FALSO;
  for each (u, v) ∈ E do
    if u ∈ U and v ∈ U then return FALSO;
  return VERO
end.
```

Algoritmo 9.1: Algoritmo non deterministico per INSIEME INDIPENDENTE

Come preannunciato, una diversa caratterizzazione dei linguaggi in NP può essere introdotta adattando in modo opportuno il concetto di certificato.

Definizione 9.8 *Dato un linguaggio $L \subseteq \Sigma^*$, $L \in$ NP se e solo se esiste una funzione $c : L \mapsto \Sigma^*$ che associa ad ogni stringa $u \in L$ un* certificato $v = c(u)$ di lunghezza polinomiale verificabile in tempo polinomiale*, vale a dire tale che:*

1. *v ha lunghezza polinomiale in $|u|$;*
2. *la macchina di Turing deterministica $\mathcal{M}$ decide il linguaggio $L' = \{uv \mid u \in L, v = c(u)\}$ in tempo polinomiale.*

Come è possibile osservare, nel caso di problemi in NP il concetto di soluzione di un problema non è altro che un caso particolare del concetto di certificato di lunghezza polinomiale verificabile in tempo polinomiale.

La questione "P = NP?" può essere interpretato come chiedersi se verificare efficientemente (in tempo polinomiale) un problema di decisione è equivalente come difficoltà a risolvere efficientemente lo stesso problema.

Ricordiamo ancora una volta che, come già osservato in precedenza, la presenza del non determinismo comporta una asimmetria tra istanze in $x \in Y_{\mathcal{P}}$ ed istanze in $x \in N_{\mathcal{P}} \cup D_{\mathcal{P}}$. Infatti, mentre affinché una istanza x sia riconosciuta come appartenente a $x \in Y_{\mathcal{P}}$ deve esistere *almeno una* computazione (*guess*) accettante, affinché la stessa istanza sia riconosciuta in $x \in N_{\mathcal{P}} \cup D_{\mathcal{P}}$ sarà necessario verificare che *tutte* le computazioni siano non accettanti. Anche nel caso in cui tutte le computazioni terminino, questo rende la condizione da verificare inerentemente diversa, richiedendo la verifica rispetto ad un quantificatore universale, piuttosto che rispetto ad un quantificatore esistenziale.

Consideriamo ad esempio il seguente problema di decisione FALSITÀ.

FALSITÀ

ISTANZA: Formula $\mathcal{F}$ in forma normale congiuntiva, definita su un insieme X di variabili booleane

PREDICATO: Non esiste nessuna assegnazione $f : X \mapsto \{\text{VERO}, \text{FALSO}\}$ che soddisfa $\mathcal{F}$

Chiaramente, FALSITÀ è definito sullo stesso insieme di istanze di SODDISFACIBILITÀ, ma il suo insieme di istanze positive corrisponde alle istanze negative di SODDISFACIBILITÀ, e viceversa.

Alla luce di quanto suesposto, verificare se una istanza $(X, \mathcal{F})$ di FALSITÀ è una istanza positiva, corrispondendo a verificare se la stessa istanza è istanza negativa di SODDISFACIBILITÀ, risulta essere inerentemente diverso (e probabilmente più difficile) rispetto al caso in cui si intenda verificare se $(X, \mathcal{F})$ è istanza positiva di SODDISFACIBILITÀ.

9.4 NP-completezza

Definiamo un problema $\mathcal{P} \in$ NP come NP-completo se $\mathcal{P}$ è completo in NP rispetto alla Karp-riducibilità polinomiale, vale a dire che $\mathcal{P}$ è NP-completo se, per ogni $\mathcal{P}_1 \in$ NP, $\mathcal{P}_1 \leq_m^p \mathcal{P}$.

La relazione $\mathcal{P}_1 \leq_m^p \mathcal{P}$ fa sì che saper risolvere $\mathcal{P}$ in tempo polinomiale comporti saper risolvere in tempo polinomiale anche $\mathcal{P}_1$. Infatti, sia $\mathcal{A}$ un algoritmo polinomiale che decide $\mathcal{P}$: allora, dato che per ipotesi deve esistere una Karp-riduzione polinomiale $\mathcal{R}$ da $\mathcal{P}_1$ a $\mathcal{P}$, ogni istanza x di $\mathcal{P}_1$ può essere decisa trasformandola in tempo polinomiale, applicando $\mathcal{R}$, in una istanza $\mathcal{R}(x)$ di $\mathcal{P}$ ed applicando quindi $\mathcal{A}$ a tale istanza.

Inoltre, la Karp-riducibilità polinomiale presenta l'ulteriore proprietà che la classe NP, così come le classi P e PSPACE, è chiusa rispetto ad essa (vedi Esercizio 8.7). Da ciò deriva che, dati due problemi $\mathcal{P}_1$ e $\mathcal{P}_2$, se $\mathcal{P}_1$ è NP–completo e $\mathcal{P}_2 \leq_m^p \mathcal{P}_1$, allora $\mathcal{P}_2 \in$ NP.

I problemi NP-completi possono svolgere un ruolo importante nell'ambito della determinazione della verità o della falsità della relazione P = NP: infatti, dato che ogni problema in NP è per definizione riducibile ad un problema NP-completo, la possibilità di risolvere in tempo polinomiale un problema NP-completo comporterebbe che P = NP.

Così come già osservato per la classe P, il metodo più naturale per mostrare l'NP-completezza di un problema di decisione $\mathcal{P}$ è quello di trovare una Karp riduzione polinomiale da qualche altro problema $\mathcal{P}_1$ che già si sa essere NP-completo. Infatti, dato che per ogni $\mathcal{P}_2 \in$ NP si ha per definizione che $\mathcal{P}_2 \leq_m^p \mathcal{P}_1$, mostrare che $\mathcal{P}_1 \leq_m^p \mathcal{P}$ comporta, per la transitività della Karp-riducibilità polinomiale (vedi Esercizio 8.6), che $\mathcal{P}_2 \leq_m^p \mathcal{P}$, e quindi che $\mathcal{P}$ è NP-completo.

Chiaramente, come già evidenziato nella Sezione 9.1 per la P-completezza,

questo processo richiede un problema NP-completo *iniziale*, la cui completezza deve evidentemente essere dimostrata mediante qualche altra tecnica.

L'importanza del teorema di Cook, presentato sotto, sta proprio nel fatto che tale teorema mostra, per mezzo di una tecnica di riduzione basata sul concetto di *tableau* introdotto nella dimostrazione del Teorema 9.1, che ogni problema in NP è polinomialmente Karp riducibile al problema SODDISFACIBILITÀ.

Teorema 9.4 (Cook) *Il problema* SODDISFACIBILITÀ *è* NP*-completo rispetto alla Karp-riducibilità polinomiale.*

Dimostrazione. Il problema si può facilmente mostrare essere in NP: infatti un algoritmo non deterministico per SODDISFACIBILITÀ viene presentato come Algoritmo 9.2. Come è possibile verificare, l'algoritmo genera mediante

```
input X: insieme di variabili booleane, (c_1, ..., c_n): insieme di clausole su V;
output boolean;
begin
  guess una assegnazione f : X ↦{TRUE,FALSE};
  for each clausola c_i ∈ F do
    if non esiste alcun t_{i,j} ∈ c_i soddisfatto da f
      then return NO;
  return VERO
end.
```

Algoritmo 9.2: Algoritmo non deterministico per SODDISFACIBILITÀ.

una *guess* una assegnazione f, di lunghezza polinomiale, per poi verificare, in tempo polinomiale, che f soddisfa $\mathcal{F}$. Alternativamente, SODDISFACIBILITÀ può essere dimostrato appartenere ad NP semplicemente osservando che l'assegnazione f è un certificato di lunghezza polinomiale e verificabile in tempo polinomiale.

Passiamo ora a mostrare che, per ogni problema $\mathcal{P} \in$ NP, è possibile definire una Karp-riduzione polinomiale da $\mathcal{P}$ a SODDISFACIBILITÀ.

La dimostrazione procede nel modo seguente, in cui per semplicità di esposizione facciamo riferimento al linguaggio L associato a $\mathcal{P}$ ed al linguaggio SAT associato a SODDISFACIBILITÀ.

Dato che $L \in$ NP esiste una macchina di Turing non deterministica $\mathcal{M} = \langle \Gamma, \flat, Q, q_0, F, \delta \rangle$ che accetta ogni stringa $x \in L$ con $\mid x \mid = n$ in tempo al più $p(n)$, dove p è un opportuno polinomio. Dati $\mathcal{M}$ ed $x \in \Sigma^*$, costruiremo una formula ϕ in forma normale congiuntiva tale che ϕ è soddisfacibile se e solo se $\mathcal{M}$ accetta x in tempo al più $p(n)$, cioè se e solo se $x \in L$.

Al fine di semplificare la dimostrazione, e senza perdere in generalità, facciamo le seguenti ipotesi relativamente ad $\mathcal{M}$:

1. $\mathcal{M}$ ha un solo nastro semi-infinito;

2. all'inizio, il nastro contiene la stringa in input x nelle prime n celle, mentre tutte le altre celle contengono il simbolo ƀ.

Come nel caso della dimostrazione del Lemma 9.5, consideriamo, senza perdita di generalità, la macchina di Turing non deterministica $\mathcal{M}' = \langle \Gamma, \text{ƀ}, Q, q_0, F, \delta' \rangle$, la cui funzione di transizione è definita nel modo seguente:

$$\delta'(q,a) = \begin{cases} \delta(q,a) \text{ se } \delta(q,a) \text{ è definita} \\ \{(q,a,i)\} \text{ altrimenti.} \end{cases}$$

La macchina di Turing $\mathcal{M}'$ ha lo stesso comportamento di $\mathcal{M}$ eccetto che ad ogni computazione massimale di $\mathcal{M}$ corrisponde una computazione di $\mathcal{M}'$ che cicla indefinitamente sulla stessa configurazione. Quindi, ad ogni computazione accettante di $\mathcal{M}$ corrisponde una computazione di $\mathcal{M}'$ che cicla su una configurazione di accettazione.

Ricordiamo inoltre che, dato che ogni computazione accettante ha lunghezza al più $p(n)$, non più di $p(n)+1$ celle del nastro sono coinvolte nella computazione stessa.

Ogni computazione eseguita da $\mathcal{M}'$, in presenza di una specifica *guess*, su input x può essere rappresentata, così come nella dimostrazione del Lemma 9.5, mediante un *tableau* T di dimensione $(p(n)+1) \times (p(n)+1)$, i cui elementi contengono coppie in $\bar{\Gamma} \times (Q \cup \{\perp\})$, con $\perp \notin Q$. Tale tableau deve soddisfare alcune proprietà: anzitutto, ogni riga deve rappresentare una configurazione lecita di $\mathcal{M}'$; inoltre, la prima riga deve rappresentare la configurazione iniziale con x sulle prime n celle del nastro; infine, ogni riga deve rappresentare una configurazione derivata dalla configurazione rappresentata nella riga precedente mediante l'applicazione della funzione di transizione di $\mathcal{M}'$. Si osservi poi che, per quanto detto, la condizione $x \in L$ è vera se e solo se esiste una computazione rappresentata da un tableau la cui ultima riga descrive una configurazione di accettazione.

La formula ϕ che deriviamo rappresenta in modo formale le condizioni che devono essere verificate affinché un tableau rappresenti una computazione di accettazione di x, e si basa sui seguenti predicati, rappresentati nella formula da variabili booleane:

1. $S(i,j,k)$ $(0 \leq i \leq p(n), 0 \leq j \leq p(n), 0 \leq k \leq |\, Q \,|)$, definita come $S(i,j,k)$ se e solo se il secondo componente del contenuto di $T[i,j]$ è $\perp$, se $k = |\, Q \,|$, o q_k, se $k < |\, Q \,|$.

2. $C(i,j,k)$ $(0 \leq i \leq p(n), 0 \leq j \leq p(n), 0 \leq k \leq |\, \Gamma \,|)$, definita come $S(i,j,k)$ se e solo se il primo componente del contenuto di $T[i,j]$ è ƀ, se $k = 0$, o a_k, se $k > 0$.

La formula ϕ è la congiunzione di 4 diverse formule, $\phi = \phi^M \wedge \phi^I \wedge \phi^A \wedge \phi^T$, che codificano le proprietà, sopra illustrate, che devono essere verificate da un tableau che rappresenti una computazione di accettazione. In particolare:

1. $\phi^M = \bigwedge_i \phi_i^M$, dove ϕ_i^M specifica le condizioni affinché la riga i-esima di T rappresenti una configurazione di $\mathcal{M}'$;
2. ϕ^I specifica che la prima riga di T deve rappresentare la configurazione iniziale di $\mathcal{M}'$ con input x;
3. ϕ^A specifica che l'ultima riga di T deve rappresentare una configurazione di accettazione di $\mathcal{M}'$;
4. ϕ^T specifica che ogni riga d T deve rappresentare una configurazione che deriva da quella rappresentata dalla riga precedente applicando la funzione di transizione di $\mathcal{M}$.

Vediamo ora più in dettaglio come le varie sottoformule di ϕ sopra introdotte sono definite.

1. ϕ_i^M è la congiunzione di tre formule $\phi_i^M = \phi_i^{M1} \wedge \phi_i^{M2} \wedge \phi_i^{M3}$. Dove:
   (a) ϕ_i^{M1} specifica che ogni cella $T[i,j]$ deve rappresentare uno ed un solo valore come secondo componente: ciò viene ottenuto esprimendo tale formula come congiunzione $\phi_i^{M1} = \bigwedge_j \left(\phi_{i,j}^{M11} \wedge \phi_{i,j}^{M12}\right)$ di due sottoformule, in cui:
      - $\phi_{i,j}^{M11} = \bigvee_k S(i,j,k)$ specifica che $T[i,j]$ deve rappresentare almeno un valore come secondo componente;
      - $\phi_{i,j}^{M12} = \bigwedge_{k_1 \neq k_2} \left(\overline{S(i,j,k_1)} \vee \overline{S(i,j,k_2)}\right)$ specifica che $T[i,j]$ deve rappresentare al più un valore come secondo componente.
   (b) ϕ_i^{M2} specifica che esattamente una cella $T[i,j]$ deve avere come secondo componente un valore diverso da $|\,Q\,|$: ciò viene ottenuto esprimendo tale formula come congiunzione $\phi_i^{M2} = \phi_i^{M21} \wedge \phi_i^{M22}$ di due sottoformule, in cui:
      - $\phi_i^{M21} = \bigvee_j \overline{S(i,j,|\,Q\,|)}$ specifica che almeno una cella $T[i,j]$ deve avere come secondo componente un valore diverso da $|\,Q\,|$ (corrispondente al simbolo $\perp$);
      - $\phi_i^{M22} = \bigwedge_{j_1 \neq j_2} (S(i,j_1,|\,Q\,|) \vee S(i,j_2,|\,Q\,|))$ specifica che al più una cella $T[i,j]$ deve avere come secondo componente un valore diverso da $|\,Q\,|$.
   (c) ϕ_i^{M3} specifica che ogni cella $T[i,j]$ deve rappresentare uno ed un solo valore come primo componente: ciò viene ottenuto esprimendo tale formula come congiunzione $\phi_i^{M3} = \bigwedge_j \left(\phi_{i,j}^{M31} \wedge \phi_{i,j}^{M32}\right)$ di due sottoformule, in cui:

- $\phi_{i,j}^{M31} = \bigvee_k C(i,j,k)$ specifica che $T[i,j]$ deve rappresentare almeno un valore come primo componente;
- $\phi_{i,j}^{M32} = \bigwedge_{k_1 \neq k_2} \left(\overline{C(i,j,k_1)} \vee \overline{C(i,j,k_2)}\right)$ specifica che $T[i,j]$ deve rappresentare al più un valore come primo componente.

Come si può vedere, ϕ^M è in CNF ed ha lunghezza $O(p(n)^3 \log n)$, in quanto è composta da $O(p(n)^3)$ variabili booleane, ognuna identificata mediante $O(\log n)$ bit. Inoltre, la formula è derivabile in tempo $O(p(n)^3)$.

2. ϕ^I è la congiunzione di due formule $\phi^I = \phi^{I1} \wedge \phi^{I2}$. Dove:
   (a) ϕ^{I1} specifica il contenuto iniziale del nastro di $\mathcal{M}'$, vale a dire i valori delle prime componenti in tutte le celle $T[0,j]$. Se $x = a_{i_0} a_{i_1} \ldots a_{i_{n-1}}$ è la stringa in input, allora si avrà $\phi^{I1} = C(0,0,i_0) \wedge C(0,1,i_1) \wedge \ldots \wedge C(0,n-1,i_{n-1}) \wedge C(0,n,0) \wedge \ldots \wedge C(0,p(n),0)$.
   (b) ϕ^{I2} specifica, attraverso i valori delle seconde componenti in tutte le celle $T[0,j]$, che inizialmente la macchina di Turing $\mathcal{M}'$ si trova nello stato è q_0 e la sua testina è posizionata sulla prima cella del nastro. Si ha quindi $\phi^{I2} = S(0,0,0) \wedge S(0,1,|\,Q\,|) \wedge \ldots \wedge S(0,p(n),|\,Q\,|)$.

   Chiaramente, ϕ^I è in CNF (in effetti è composta da una unica clausola), ha lunghezza $O(p(n) \log n)$ ed è derivabile in tempo $O(p(n))$.

3. ϕ^A è la disgiunzione di $|\,F\,|$ formule booleane, ognuna delle quali specifica la condizione che $\mathcal{M}$ si trovi nello stato $q_k \in F$, con la testina su una delle $p(n)+1$ celle di nastro: $\phi^A = \bigvee_{q_k \in F} \left(\bigvee_j S(0,j,k)\right)$.

   Anche ϕ^A è chiaramente in CNF (è composta da $|\,F\,|\ (p(n)+1)$ clausole composte da un solo predicato), ha lunghezza $O(p(n) \log n)$ ed è derivabile in tempo $O(p(n))$.

4. $\phi^T = \bigwedge_i \phi_i^T$, dove ϕ_i^T specifica le condizioni affinché la riga i-esima di T rappresenti una configurazione di $\mathcal{M}'$ derivata, attraverso l'applicazione della funzione di transizione δ, dalla configurazione rappresentata alla riga $(i-1)$-esima. A sua volta, ogni formula ϕ_i^T ha la struttura $\phi_i^T = \bigwedge_i \phi_{i,j}^T$, dove $\phi_{i,j}^T$ specifica che il contenuto delle tre celle $T[i,j-1]$, $T[i,j]$, $T[i,j+1]$ del tableau deve poter derivare, per mezzo della funzione di transizione, dal contenuto delle celle $T[i-1,j-1]$, $T[i-1,j]$, $T[i-1,j+1]$.[5]

   In particolare, sia $S = (s_0, s_1, \ldots, s_r)$ una qualunque enumerazione dei possibili passi di computazione deterministica definiti nella funzione di

[5] In particolare, se $j = 0$ la formula $\phi_{i,0}^T$ si riferirà alle sole celle $T[i-1,0]$, $T[i-1,1]$, $T[i,0]$, $T[i,1]$, mentre $j = p(n)$ la formula $\phi_{i,p(n)}^T$ si riferirà alle sole celle $T[i-1,p(n)-1], T[i-1,p(n)], T[i,p(n)-1], T[i,p(n)]$.

transizione $\delta_{\mathcal{M}}$ di $\mathcal{M}$, vale a dire che ogni s_i corrisponde ad una 5-pla $(p_i, c_i, p'_i, c'_i, m_i)$, con $p_i, p'_i \in Q$, $c_i, c'_i \in \bar{\Gamma}$, $m_i \in \{d, s, i\}$, e $(p'_i, c'_i, m_i) \in \delta_{\mathcal{M}}(p_i, c_i)$. Ogni formula $\phi^T_{i,j}$ avrà allora la struttura $\phi^T_{i,j} = \phi^{T0}_{i,j} \vee \phi^{T1}_{i,j} \vee \phi^{T2}_{i,j} \vee \phi^{T3}_{i,j}$, dove:

- $\phi^{T0}_{i,j}$ specifica la correttezza della situazione in cui $T[i-1, j-1]$, $T[i-1, j]$, $T[i-1, j+1]$ rappresentano celle di nastro su cui non è posizionata la testina, $T[i, j]$ rappresenta anch'essa una cella su cui non si trova la testina, ed i caratteri contenuti nelle celle rimangono gli stessi nel passare dalla riga $i-1$ alla riga i. Ciò viene realizzato definendo $\phi^{T0}_{i,j}$ nel modo seguente:

$$\begin{aligned}\phi^{T0}_{i,j} = \quad & S(i-1, j-1, |\,Q\,|) \wedge S(i-1, j, |\,Q\,|) \wedge S(i-1, j+1, |\,Q\,|) \wedge \\ & \wedge S(i, j, |\,Q\,|) \wedge \left(\bigvee_{0 \leq k \leq \Gamma} (C(i-1, j, k) \wedge C(i, j, k)) \right).\end{aligned}$$

- $\phi^{T1}_{i,j}$ specifica la correttezza della situazione in cui $T[i-1, j-1]$ rappresenta una cella di nastro sulla quale è posizionata la testina (e quindi si ha $S(i-1, j-1, k)$ per qualche $k <| Q |$): in questo caso, la formula considera tutte le possibili applicazioni della funzione di transizione mediante la disgiunzione $\phi^{T1}_{i,j} = \bigvee_{s_k \in S} \phi^{T1}_{i,j,k}$ dove la formula $\phi^{T1}_{i,j,k}$ enumera tutti i contenuti delle 6 celle considerate che sono correlati mediante l'applicazione del passo di computazione $s_k \in S$.
  Ciò viene realizzato definendo $\phi^{T1}_{i,j,k}$ nel modo seguente. Sia $s_k = (p_k, c_k, p'_k, c'_k, m_k)$, e sia, senza perdita di generalità $p_k = q_{l_1} \in Q$, $p'_k = q_{l_2} \in Q$, $c_k = a_{h_1} \in \bar{\Gamma}$, $c_k = a_{h_2} \in \bar{\Gamma}$: consideriamo separatamente i tre casi in cui m_k è uguale a d, s, o i.
  (a) se $m_k = d$, allora

$$\begin{aligned}\phi^{T1}_{i,j,k} = \quad & \bigvee_{0 \leq t,u \leq \Pi} (S(i-1, j-1, l_1) \wedge C(i-1, j-1, h_1) \wedge \\ & \wedge C(i-1, j, t) \wedge C(i-1, j+1, u) \wedge C(i, j-1, h_2) \wedge \\ & \wedge C(i, j, t) \wedge C(i, j+1, u) \wedge S(i, j, h_2));\end{aligned}$$

  (b) se $m_k = s$, allora

$$\begin{aligned}\phi^{T1}_{i,j,k} = \quad & \bigvee_{0 \leq t,u \leq \Pi} (S(i-1, j-1, l_1) \wedge C(i-1, j-1, h_1) \wedge \\ & \wedge C(i-1, j, t) \wedge C(i-1, j+1, u) \wedge C(i, j-1, h_2) \wedge \\ & \wedge C(i, j, t) \wedge C(i, j+1, u));\end{aligned}$$

  (c) se, infine, $m_k = i$, allora

$$\begin{aligned}\phi^{T1}_{i,j,k} = \quad & \bigvee_{0 \leq t,u \leq \Pi} (S(i-1, j-1, l_1) \wedge C(i-1, j-1, h_1) \wedge \\ & \wedge C(i-1, j, t) \wedge C(i-1, j+1, u) \wedge C(i, j-1, h_2) \wedge \\ & \wedge C(i, j, t) \wedge C(i, j+1, u) \wedge S(i, j-1, h_2)).\end{aligned}$$

- $\phi_{i,j}^{T2}$ e $\phi_{i,j}^{T2}$ specificano la correttezza delle situazioni in cui $T[i-1,j]$ o $T[i-1,j+1]$ rappresentano celle di nastro sulla quale è posizionata la testina. La loro struttura è simile a quella descritta per $\phi_{i,j}^{T1}$, e viene lasciata per esercizio (vedi Esercizio 9.6).

Si osservi che ogni formula $\phi_{i,j}^{T}$ è composta da una quantità costante di predicati C ed S, ognuno dei quali ha dimensione $O(\log n)$. Pur non essendo in CNF, ogni formula può poi essere facilmente trasformata in una formula $\overline{\phi}_{i,j}^{T}$ in CNF equivalente, anch'essa composta da $O(1)$ predicati, e quindi anch'essa di dimensione $O(\log n)$.

Da ciò deriva che l'intera formula ϕ^T ha dimensione $O((p(n))^2 \log n)$ ed è derivabile in tempo $O((p(n))^2)$.

Se si considera ora l'intera formula ϕ, è possibile verificare che la componente dominante nella sua dimensione è quella relativa alla dimensione di ϕ_M, dal che risulta che la dimensione di ϕ è $O(p(n)^3 \log n)$, e l'intera formula può essere derivata in tempo $O(p(n)^3)$.

Lasciamo per esercizio (vedi Esercizio 9.7) la semplice verifica che la formula ϕ è soddisfacibile se e solo se esiste una computazione accettante di $\mathcal{M}$. □

Esercizio 9.6 Individuare la struttura delle formule $\phi_{i,j}^{T2}$ e $\phi_{i,j}^{T2}$ nella dimostrazione del teorema precedente.

Esercizio 9.7 Verificare che, nella dimostrazione del teorema precedente, la formula ϕ è soddisfacibile, dato un certo input x, se e solo se esiste una computazione accettante di $\mathcal{M}$ a partire da tale input.

Una volta individuato, mediante il teorema di Cook, un primo problema NP-completo è possibile ora fare riferimento alla tecnica standard per provare che un problema $\mathcal{P}$ è NP-completo. Tale tecnica, riassumiamo, è basata su due passi:

1. dimostrare che $\mathcal{P} \in$ NP;
2. esiste un problema $\mathcal{P}'$ NP-completo, mostrare che $\mathcal{P}' \leq_m^p \mathcal{P}$, fornendo una procedura che, in tempo polinomiale, trasforma ogni istanza di $\mathcal{P}'$ in una istanza equivalente di $\mathcal{P}$.

Come primo esempio di dimostrazione "standard" di NP-completezza, mostriamo che il problema 3-SODDISFACIBILITÀ, ottenuto da SODDISFACIBILITÀ considerando nella sua definizione soltanto clausole aventi 3 termini, è NP-completo.

3-SODDISFACIBILITÀ

ISTANZA: Una formula CNF $\mathcal{F}$ su un insieme X di variabili booleane, con ogni clausola composta da al più 3 termini.

PREDICATO: Esiste una assegnazione $f : X \mapsto \{\text{VERO}, \text{FALSO}\}$ che soddisfa $\mathcal{F}$?

Teorema 9.5 *Il problema* 3-SODDISFACIBILITÀ *è* NP*-completo.*

Dimostrazione. Mostriamo inizialmente che 3-SODDISFACIBILITÀ$\in$ NP: infatti, dato $f : X \mapsto \{\text{VERO}, \text{FALSO}\}$ è chiaramente un certificato di lunghezza polinomiale verificabile in tempo polinomiale.

Al fine di provare che 3-SODDISFACIBILITÀ$\leq_m^p$SODDISFACIBILITÀ mostriamo dapprima una possibile riduzione da istanze di SODDISFACIBILITÀ ad istanze di 3-SODDISFACIBILITÀ.

Consideriamo una istanza generica X, C di 3-SODDISFACIBILITÀ, dove $X = \{x_1, x_2, \ldots, x_n\}$ è un insieme di variabili booleane e $C = \{c_1, c_2, \ldots, c_m\}$ è un insieme di clausole su X. Da X, C otterremo una istanza X', C' di 3-SODDISFACIBILITÀ in cui da ogni clausola $c_i \in C$ vengono derivati un insieme di clausole a tre termini C'_i definite su un insieme di variabili X'_i: si avrà che l'istanza di 3-SODDISFACIBILITÀ risultante avrà $X' = X \cup (\cup_{i=1}^m X'_i)$ $C' = \cup_{i=1}^m C'_i$.

Tale corrispondenza dipenderà dal numero $|\, c_i \,|$ di termini nella clausola c_i:

1. $|\ c_i\ | = 1$. Sia $c_i = \{z\}$: allora, $X'_i = \{y_i^1, y_i^2\}$ e $C'_i = \{\{z_1, y_i^1, y_i^2\}, \{z_1, \overline{y}_i^1, y_i^2\}, \{z_1, y_i^1, \overline{y}_i^2\}, \{z_1, \overline{y}_i^1, \overline{y}_i^2\}\}$. Si noti che una assegnazione che soddisfa c_i (che cioè rende vero il termine z), comunque esteso a piacere a y_i^1, y_i^2 soddisfa anche C'_i; inoltre, una assegnazione che soddisfa C'_i dovrà necessariamente rendere z vero (altrimenti una delle quattro clausole non può essere soddisfatta) e, quindi, soddisfa anche c_i. Da ciò possiamo concludere che l'insieme di clausole in C'_i è soddisfatto da tutti e soli gli assegnamenti (comunque estesi) tali che $z = \text{VERO}$, gli assegnamenti cioè che soddisfano c_i.

2. $|\, c_i \,| = 2$. Sia $c_i = \{z, v\}$: allora, $X'_i = \{y_i^1\}$ e $C'_i = \{\{z, v, y_i^1\}, \{z, v, \overline{y}_i^1\}\}$. Si noti che una assegnazione che soddisfa c_i (che cioè rende vero uno almeno tra i termini z e v), comunque esteso a piacere a y_i^1 soddisfa anche C'_i; inoltre, una assegnazione che soddisfa C'_i dovrà necessariamente rendere almeno un termine tra z e v vero (altrimenti una delle due clausole non può essere soddisfatta) e, quindi, soddisfa anche c_i. In questo caso, le due clausole in C'_i sono soddisfatte da tutti e soli gli assegnamenti (comunque estesi) tali che $z = \text{VERO}$ o $v = \text{FALSO}$, e quindi da tutti e soli gli assegnamenti che soddisfano c_i.

3. $|\, c_i \,| = 3$. Sia $c_i = \{z, v, w\}$: dato che in questo caso c_i ha esattamente i tre termini richiesti nella definizione di 3-SODDISFACIBILITÀ, $X'_i = \emptyset$ e $C'_i = \{\{z, v, w\}\}$. Chiaramente, C'_i è soddisfatta da tutti e soli gli assegnamenti che soddisfano c_i.

4. $|\, c_i \,| = k \geq 4$. Sia $c_i = \{z_1, z_2, \ldots, z_k\}$: allora avremo $X'_i = \{y_i^j,\ 1 \leq j \leq k-3\}$ e $C'_i = \{\{z_1, z_2, y_i^1\},\ \{\overline{y}_i^1, z_3, y_i^2\},\ \{\overline{y}_i^2, z_4, y_i^2\},\ \ldots,\ \{\overline{y}_i^{k-4}, z_{k-2}, y_i^{k-3}\},\ \{\overline{y}_i^{k-3}, z_{k-1}, z_k\}\}$. In questo caso, si noti che una assegnazione f che soddisfa c_i dovrà rendere almeno un termine in $\{z_i \ldots, z_k\}$ vero: sia z_r il primo di tali termini (siano cioè tutti i termini z_j con $j < i$ posti a falso dall'assegnazione), allora, l'estensione f' di f in cui $y_i^1 =$ VERO, $y_i^2 =$ VERO, $\ldots$, $y_i^{r-2} =$ VERO, $y_i^{r-1} =$ VERO, $\ldots, y_i^{k-3} =$ FALSO soddisfa l'insieme di clausole C'_i. Inoltre, si osservi che una assegnazione f' che soddisfa C'_i dovrà necessariamente porre a vero almeno un termine z_r (altrimenti una delle clausole non viene soddisfatta, indipendentemente dai valori assegnati alle variabili y_i^j): evidentemente, tale assegnazione soddisferà anche c_i.

Da quanto detto, si può concludere che (1) se esiste una assegnazione f su X che soddisfa C, esso può essere esteso, come illustrato sopra, ad ottenere una assegnazione f' su X' che soddisfa C' e (2) se esiste una assegnazione f' su X' che soddisfa C', la stessa assegnazione soddisfa anche C.

Da ciò evidentemente deriva che C è soddisfacibile se e solo se lo è C'. Si osservi infine che la trasformazione può chiaramente essere effettuata in tempo polinomiale, e precisamente $O(nm)$. □

Come altro esempio di dimostrazione di NP-completezza, mostriamo ora che il seguente problema COPERTURA CON NODI è NP-completo.

COPERTURA CON NODI
ISTANZA: Grafo $G = (V, E)$, intero $K > 0$
PREDICATO: Esiste un sottoinsieme $V' \subseteq V$ tale che $\|V'\| \leq K$ e $\forall (u, v) \in E$ $u \in V'$ o $v \in V'$?

Teorema 9.6 *Il problema* COPERTURA CON NODI *è* NP*-completo.*

Dimostrazione. Mostriamo che COPERTURA CON NODI appartiene ad NP: infatti, dato un sottoinsieme $V' \subseteq V$ tale che $|\, V' \,| \leq K$ e $\forall (u, v) \in E$ $u \in V'$ o $v \in V'$ è chiaramente un certificato di lunghezza polinomiale, verificabile in tempo polinomiale.

Per provare la completezza di COPERTURA CON NODI in NP, mostriamo che 3-SODDISFACIBILITÀ$\leq_m^p$COPERTURA CON NODI. Sia data una istanza generica (X, C) di 3-SODDISFACIBILITÀ: a partire da (X, C) deriviamo una

grafo $G = (V, E)$ ed un intero $K > 0$ tali che G ha una copertura con nodi di dimensione non superiore a K se e solo se l'insieme di clausole C è soddisfacibile.

L'insieme V dei nodi di G è definito come segue:

- ad ogni variabile $x_i \in X$ sono associati due nodi $x_i, \overline{x}_i \in V$;
- ad ogni clausola $c_j \in C$ sono associati tre nodi $c_j^1, c_j^2, c_j^3 \in V$.

Per quanto riguarda l'insieme E degli archi di G (vedi Figura 9.8):

- esiste un arco $(x_i, \overline{x}_i) \in E$ per ogni $x_i \in X$;
- per ogni clausola $c_j = \{t_j^1, t_j^2, t_j^3\} \in C$, introduciamo tre archi (c_j^1, c_j^2), (c_j^2, c_j^3), (c_j^3, c_j^1). Inoltre, introduciamo un arco per ogni termine t_j^s ($s = 1, 2, 3$) in c_j nel modo seguente: sia $t_j^s = x_k \in X$: introduciamo allora un arco (c_j^s, x_k) se $t_j^s = x_k$ o un arco $(c_j^s, \overline{x}_k)$ se $t_j^s = \overline{x}_k$.

Si osservi che $\mid V \mid = 2 \mid X \mid + 3 \mid C \mid$ e che $\mid E \mid = \mid X \mid + 6 \mid C \mid$, e che la trasformazione può essere effettuata in tempo polinomiale. Poniamo inoltre $K = \mid X \mid + 2 \mid C \mid$. Ad una prima analisi, si può verificare che non esistono coperture di nodi su G di taglia inferiore a K: infatti, per ognuno degli $\mid X \mid$ archi del tipo $(x_i, \overline{x}_i)$ almeno uno tra i due nodi estremi dovrà essere inserito in una copertura, mentre per ognuna delle $\mid C \mid$ componenti c_j^1, c_j^2, c_j^3 almeno due dei tre nodi devono a loro volta entrare a far parte della copertura stessa.

Mostriamo ora che se (X, C) è soddisfacibile allora esiste una copertura V' di taglia K su G. Sia f una assegnazione che soddisfa (X, C): allora, per ogni variabile x_i, se $f(x_i) =$ VERO inseriamo in V' il nodo x_i, altrimenti inseriamo il nodo $\overline{x}_i$. Inoltre, per ogni clausola $c_j = \{t_j^1, t_j^2, t_j^3\}$, sia $t_j^s \in c_j$ un termine, che deve necessariamente esistere, tale che $f(t_j^s) =$ VERO: inseriamo allora in V' i due nodi $\{c_j^r \mid r \neq s\}$. È immediato verificare che tale insieme di K nodi rappresenta effettivamente una copertura di nodi di G.

Per quanto riguarda l'implicazione inversa, sia V' una copertura dei nodi di G con $\mid V' \mid \leq K$. Per quanto osservato sopra, dovrà essere necessariamente $\mid V' \mid = K$: in effetti, V' dovrà necessariamente includere 1 nodo per ogni arco del tipo $(x_i, \overline{x}_i)$ e due nodi per ogni componente c_j^1, c_j^2, c_j^3. Deriviamo una assegnazione f che soddisfa (X, C) nel modo seguente: per ogni $x_i \in X$, $f(x_i) =$ VERO se $x_i \in V'$, altrimenti $f(x_i) =$ FALSO. Tale assegnazione soddisferà (X, C) in quanto, per ogni clausola c_j, la corrispondente componente c_j^1, c_j^2, c_j^3 in G avrà esattamente un nodo $c_j^s \in V - V'$: tale nodo necessariamente sarà collegato ad un nodo associato al termine t_j^s, appartenente necessariamente a V'. Dato che è stato posto $f(t_j^s) =$ VERO, ne deriva che c_j è soddisfatta. □

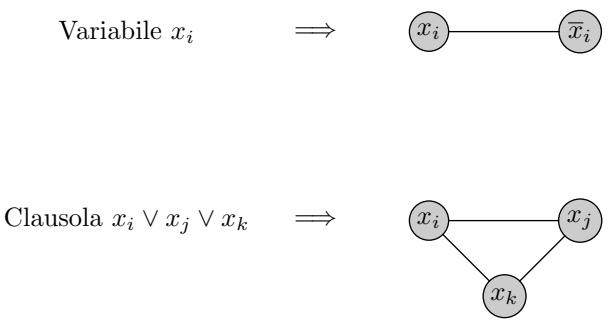

FIGURA 9.8 Riduzione da 3-SODDISFACIBILITÀ a COPERTURA CON NODI.

Esercizio 9.8 Dimostrare che il problema CRICCA introdotto nella Sezione 8.2 è NP-completo.
[Suggerimento: Definire una Karp-riduzione polinomiale da COPERTURA CON NODI.]

Introduciamo infine il seguente problema.

COPERTURA CON INSIEMI

ISTANZA: Insieme S, collezione $\mathcal{C} = \{s_1, s_2, \ldots, s_n\}$ di sottoinsiemi di S, intero $K > 0$.

PREDICATO: Esiste un sottoinsieme $\mathcal{C}'$ di $\mathcal{C}$ che copre S, vale a dire tale che $\bigcup_{s_i \in \mathcal{C}'} = S$, avente cardinalità $|\mathcal{C}'| \leq K$?

Esercizio 9.9 Dimostrare che il problema COPERTURA CON INSIEMI è NP-completo.
[Suggerimento: Definire una Karp-riduzione polinomiale da COPERTURA CON NODI.]

Osserviamo ora che in generale, dato un problema $\mathcal{P}$, se restringiamo il problema stesso ad un sottoinsieme delle possibili istanze otteniamo un suo *sottoproblema* $\mathcal{P}'$ (come nel caso di 3-SODDISFACIBILITÀ rispetto a SODDISFACIBILITÀ) che certamente non è più difficile da risolvere rispetto a $\mathcal{P}$: in effetti vale banalmente la relazione $\mathcal{P}' \leq_r \mathcal{P}$ per qualunque riducibilità $\leq_r$, in quanto ogni istanza di $\mathcal{P}'$ è istanza di $\mathcal{P}$.

È interessante però in tale situazione chiedersi se la restrizione introdotta sull'insieme delle istanze comporta una maggiore facilità di risoluzione del problema. Ci chiediamo cioè se $\mathcal{P} \leq_r \mathcal{P}'$: se tale relazione è vera allora, almeno rispetto alla riducibilità considerata, $\mathcal{P}$ e $\mathcal{P}'$ sono equivalenti mentre,

altrimenti, $\mathcal{P}$ è effettivamente più difficile di $\mathcal{P}'$. Ciò risulta particolarmente interessante nel caso in cui $\mathcal{P}$ sia NP-completo, in quanto ci chiediamo se il sottoproblema $\mathcal{P}'$ considerato è ancora NP-completo o meno.

Ad esempio, nel caso di SODDISFACIBILITÀ e di 3-SODDISFACIBILITÀ abbiamo già visto che la restrizione introdotta dal considerare soltanto istanze con clausole di esattamente 3 termini non rende il problema più facile, rispetto alla Karp-riducibilità polinomiale, in quanto 3-SODDISFACIBILITÀ è anch'esso NP-completo e quindi SODDISFACIBILITÀ$\leq_m^p$3-SODDISFACIBILITÀ.

Esistono però delle diverse restrizioni di SODDISFACIBILITÀ per le quali si ha un effettivo miglioramento della relativa difficoltà di soluzione. Una di tali restrizioni è quella relativa a SODDISFACIBILITÀ DI FORMULE DI HORN: chiaramente SODDISFACIBILITÀ DI FORMULE DI HORN$\leq_m^p$SODDISFACIBILITÀ (e quindi SODDISFACIBILITÀ DI FORMULE DI HORN$\in$ NP), mentre non si ha SODDISFACIBILITÀ$\leq_m^p$SODDISFACIBILITÀ DI FORMULE DI HORN, a meno che LOGSPACE = P = NP.

Un secondo sottoproblema di SODDISFACIBILITÀ che è sostanzialmente più semplice di esso è 2-SODDISFACIBILITÀ, vale a dire la restrizione al caso in cui ogni clausola è composta da al più 2 termini. Anche in questo caso è possibile mostrare che 2-SODDISFACIBILITÀ$\in$ P mostrando che esiste un algoritmo che risolve tale problema in tempo polinomiale (vedi Esercizio 9.10): ne deriva quindi che non si ha SODDISFACIBILITÀ$\leq_m^p$2-SODDISFACIBILITÀ, a meno che P = NP.

Esercizio 9.10 Trovare un algoritmo polinomiale che risolva 2-SODDISFACIBILITÀ.

In generale, possiamo considerare la relazione $<_R$ di restrizione di un problema in un suo sottoproblema, definita come $\mathcal{P} <_R \mathcal{P}'$ se e solo se $\mathcal{P}'$ è una restrizione di $\mathcal{P}$, vale a dire le sue istanze sono un sottoinsieme delle istanze di $\mathcal{P}$: come si può immediatamente verificare, tale relazione è una relazione di ordine parziale. Dato un insieme di problemi in NP, possiamo associare ad esso un grafo orientato aciclico i cui nodi corrispondono ai problemi e si ha un arco dal nodo u al nodo v se e solo se, detti $\mathcal{P}_u$ e $\mathcal{P}_v$ i problemi associati a u e v, si ha che $\mathcal{P}_u <_R \mathcal{P}_v$ e che non esiste alcun problema $\mathcal{P}_w$ tale che $\mathcal{P}_u <_R \mathcal{P}_w <_R \mathcal{P}_v$.

Tale grafo può allora essere decomposto in tre regioni (vedi Figura 9.9):

1. una regione "superiore", di tutti problemi NP-completi;
2. una regione "inferiore", di problemi in P;
3. una regione "intermedia", di problemi per i quali non si dispone di dimostrazioni né di NP-completezza, né di polinomialità.

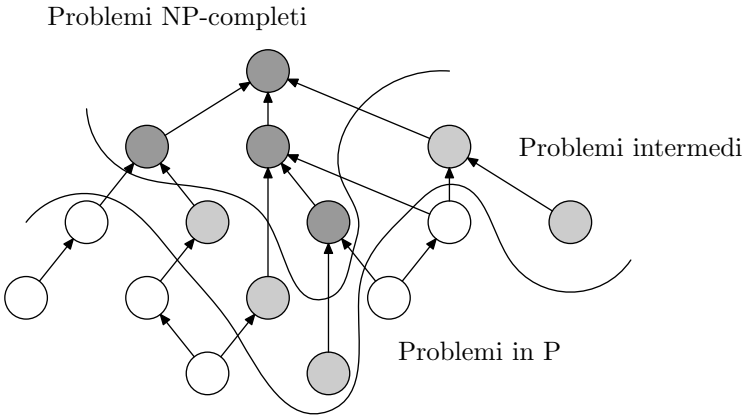

FIGURA 9.9 Relazione di restrizione fra problemi in NP.

9.5 Ancora sulla classe NP

9.5.1 La classe CO-NP

La relazione tra FALSITÀ e SODDISFACIBILITÀ osservata nella Sezione 9.3, è un tipico esempio della relazione generale di complementarietà tra problemi di decisione, definita qui sotto.

Definizione 9.9 *Dato un problema di decisione $\mathcal{P}$, il problema complementare co-$\mathcal{P}$ è un problema di decisione tale che $I_{co\text{-}\mathcal{P}} = I_{\mathcal{P}}$, $Y_{co\text{-}\mathcal{P}} = N_{\mathcal{P}}$ e $N_{co\text{-}\mathcal{P}} = Y_{\mathcal{P}}$.*

Per ogni classe di complessità $\mathcal{C}$ possiamo quindi definire la corrispondente classe complementare co-$\mathcal{C}$ come la classe contenente tutti e soli i problemi complementari a problemi in $\mathcal{C}$, co-$\mathcal{C} = \{\mathcal{P} : \text{co-}\mathcal{P} \in \mathcal{C}\}$. Coerentemente a ciò, indichiamo con CO-NP la classe complementare di NP.

Si osservi che, dato che risulta immediato che P = CO-P (in quanto per risolvere un problema complementare di un problema $\mathcal{P} \in$ P basta cambiare l'algoritmo per $\mathcal{P}$ in modo da scambiare risposte VERO con risposte FALSO), allora se NP $\neq$ CO-NP si ha che P $\neq$ NP, ma si badi che non è vero il contrario, in quanto NP = CO-NP non implica necessariamente che P = NP.

Non è difficile rendersi conto che se $\mathcal{P}$ è NP–completo, allora co-$\mathcal{P}$ è CO-NP–completo. Infatti:

1. per definizione, $\mathcal{P} \in$ NP implica co-$\mathcal{P} \in$ CO-NP (e viceversa);

2. dato che per ogni $\mathcal{P}_1 \in$ NP vale la proprietà $\mathcal{P}_1 \leq_m^p \mathcal{P}$, si ha che co-$\mathcal{P}_1 \in$ CO-NP e che co-$\mathcal{P}_1 \leq_m^p$ co-$\mathcal{P}$.

Possiamo inoltre dimostrare i seguenti teoremi. Il primo di tali teoremi ci mostra come la proprietà fondamentale dei problemi NP-completi, per la quale se un problema NP-completo è anche polinomiale allora P = NP, è valida anche per i problemi co-NP-completi.

Teorema 9.7 *Se $\mathcal{P}$ è* co-NP*–completo, allora $\mathcal{P} \in$ P implica* P = NP*.*

Dimostrazione. Dato che $\mathcal{P}$ è co-NP–completo, allora co-$\mathcal{P}$ è NP–completo. Inoltre, dato che $\mathcal{P} \in$ P, ne segue che co-$\mathcal{P} \in$ co-P = P. Per ogni $\mathcal{P}_1 \in$ NP abbiamo quindi che $\mathcal{P}_1 \leq_m^p$ co-$\mathcal{P} \in$ P, da cui deriva che $\mathcal{P}_1 \in$ P. □

I due teoremi seguenti ci mostrano una proprietà più debole della precedente, per la quale se un problema NP-completo appartiene a co-NP (o, simmetricamente, se un problema co-NP-completo appartiene a NP) allora vale la condizione NP=co-NP. Si osservi che in generale si ritiene che, così come P ≠ NP, si abbia anche NP ≠ co-NP.

Teorema 9.8 *Se un problema $\mathcal{P}$ è* co-NP*–completo, allora $\mathcal{P} \in$ NP comporta* NP = co-NP*.*

Dimostrazione. Per ipotesi, dato un qualunque $\mathcal{P}_1 \in$ co-NP, avremo $\mathcal{P}_1 \leq_m^p \mathcal{P} \in$ NP. Ne consegue, per la chiusura della classe NP rispetto alla Karp-riducibilità polinomiale (vedi Esercizio 8.7), che $\mathcal{P}_1 \in$ NP, e quindi, in generale, che co-NP ⊆ NP.

D'altro canto, in modo simmetrico, per ogni $\mathcal{P}_2 \in$ NP avremo che $\mathcal{P}_2 \leq_m^p$ co-$\mathcal{P} \in$ co-NP. Di conseguenza, per la chiusura della classe co-NP rispetto alla Karp-riducibilità polinomiale (vedi Esercizio 9.11), si avrà che $\mathcal{P}_2 \in$ co-NP, e quindi, in generale, che NP ⊆ co-NP, il che, in conseguenza delle considerazioni precedenti, comporta NP = co-NP. □

Esercizio 9.11 Dimostrare che la classe co-NP è chiusa rispetto alla Karp riduzione polinomiale.

Per simmetria con il Teorema 9.8, vale evidentemente anche il seguente teorema.

Teorema 9.9 *Se un problema $\mathcal{P}$ è* NP*–completo, allora $\mathcal{P} \in$ co-NP implica* NP = co-NP*.*

Dimostrazione. La dimostrazione deriva immediatamente, nella sua struttura, da quella del Teorema 9.8 □

In Figura 9.10 è riportata la struttura dele classi NP e co-NP nel caso in cui esse non coincidano.

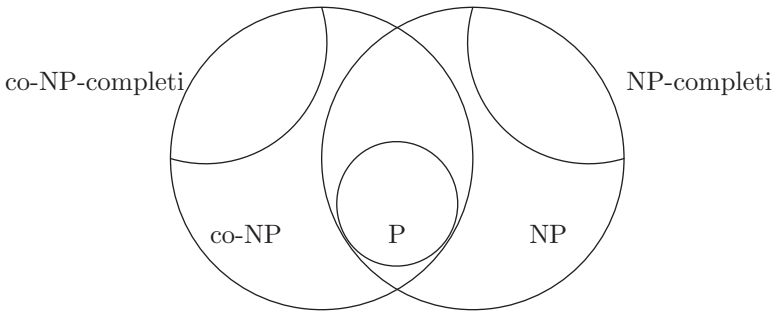

FIGURA 9.10 Struttura di NP e CO-NP se NP $\neq$ CO-NP.

9.5.2 Problemi intermedi in NP

Come osservato sopra, un problema si può trovare nella regione intermedia del grafo in Figura 9.9 in quanto non sono disponibili algoritmi polinomiali per la sua soluzione, né riduzioni da problemi NP-completi che ne mostrino la NP-completezza. Esiste anche la possibilità che il problema in questione sia inerentemente intermedio, vale a dire che, pur non essendo NP-completo, esso non sia in P. L'esistenza di problemi di questo tipo è stata dimostrata dal seguente teorema, che enunciamo senza dimostrazione.

Teorema 9.10 (Ladner) *Se* P $\neq$ NP *allora per ogni* $k \geq 1$ *esiste una sequenza di linguaggi* $L_1, L_2, \ldots, L_k$ *tali che:*

$$L_P \leq_m^p L_1 \leq_m^p L_2 \leq_m^p \ldots \leq_m^p L_k \leq_m^p L_{NPC},$$

dove $L_P \in$ P *e* L_{NPC} *è* NP*-completo.*

Tuttavia, non sono noti al momento problemi "naturali" che siano intermedi in NP.

Due importanti problemi di decisione che sono dei buoni candidati ad essere inerentemente intermedi in NP sono ISOMORFISMO TRA GRAFI e NUMERO PRIMO, definiti nel modo seguente:

ISOMORFISMO TRA GRAFI

ISTANZA: Grafi $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$.

PREDICATO: I due grafi sono isomorfi, esiste cioè una funzione biiettiva $f : V_1 \mapsto V_2$ tale che, per ogni coppia di nodi $u, v \in V_1$, $(u, v) \in E_1$ se e solo se $(f(u), f(v)) \in E_2$?

NUMERO PRIMO
ISTANZA: Intero $n > 0$.
PREDICATO: L'intero n è primo?

Mentre è immediato mostrare che ISOMORFISMO TRA GRAFI $\in$ NP, in quanto una qualunque funzione biiettiva da V_1 a V_2 può essere descritta in spazio polinomiale e la relativa verifica richiede tempo polinomiale, lo stesso non è vero per NUMERO PRIMO. In effetti, l'individuazione di un possibile modo di costruire certificati di dimensione polinomiale e verificabili in tempo polinomiale associati alle istanze positive di tale problema non è immediata e richiede l'uso del seguente risultato di teoria dei numeri.

Teorema 9.11 *Un intero $n > 1$ è primo se e solo se esiste un intero r tale che:*

1. $r^{n-1} \equiv 1$ *(mod n);*
2. *per ogni numero primo divisore di $n-1$ si ha $r^{(n-1)/q} \not\equiv 1$ (mod n).*

È possibile quindi introdurre un certificato di primalità di un intero n come un vettore $c(n) = (r, q_1, c(q_1), q_2, c(q_2, \ldots, q_k, c(q_k)))$, dove $q_1, \ldots, q_k$ sono i divisori primi di $n-1$ e $c(q_1), \ldots, c(q_k)$ sono i relativi certificati di primalità (aventi la medesima struttura). Verificare tale certificato richiederà verificare le due proprietà sopra enunciate, oltre che accertarsi che ogni q_i divide $n-1$, che $q_1, \ldots, q_k$ sono tutti i divisori di $n-1$, e che ogni q_i a sua volta è primo, verificando ricorsivamente il certificato $c(q_i)$.

Esercizio 9.12 Dimostrare che il certificato di primalità introdotto sopra ha lunghezza polinomiale (in $\log n$) ed è verificabile in tempo polinomiale.
[Suggerimento: Ricordare che il numero di divisori primi di un intero p è minore di $\log p$.]

Si consideri ora il problema complementare di NUMERO PRIMO, definito nel modo seguente.

NUMERO COMPOSTO
ISTANZA: Intero $n > 0$.
PREDICATO: L'intero n è composto?

Chiaramente, NUMERO COMPOSTO $\in$ NP, come si può concludere osservando che, data una istanza positiva di tale problema (un intero non primo n), un qualunque intero $m < n$ è un certificato di lunghezza polinomiale e verificabile in tempo polinomiale, determinando se m divide n. Dall'appartenenza di NUMERO COMPOSTO a NP deriva evidentemente che NUMERO PRIMO

$\in$ NP $\cap$ co-NP e quindi, per il Teorema 9.9, se Numero primo fosse NP-completo, allora NP = co-NP.

Un ulteriore problema correlato ai precedenti, di grande importanza sia teorica che per tutte le applicazioni legate alla sicurezza ed alla crittografia, che può essere dimostrato appartenere a NP $\cap$ co-NP è il seguente problema Fattorizzazione.

Fattorizzazione
Istanza: Interi $n, k > 0$.
Predicato: L'intero n ha un divisore $d \leq k$?

Il problema Fattorizzazione può essere dimostrato appartenere a NP osservando che un suo certificato è dato da un intero $d \leq k$ che divide n: chiaramente, il certificato ha lunghezza polinomiale ed è verificabile in tempo polinomiale, determinando che d sia effettivamente minore o uguale a k e che divida n.

L'appartenenza di Fattorizzazione a co-NP deriva osservando che un certificato del problema complementare, vale a dire che un certificato che n non ha divisori minori o uguali a k è dato dalla decomposizione di n in fattori primi $n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$, dove e_i è la molteplicità del fattore primo p_i nel decomposizione di n e dove per ogni fattore p_i si ha $p_i > k$, corredata dall'insieme dei certificati $c(p_1), \dots, c(p_k)$ di primalità dei fattori di n.

9.5.3 Problemi fortemente NP*-completi e pseudopolinomialità*

È interessante osservare che, nell'ambito dei problemi NP-completi, esistono problemi in un certo senso "strutturalmente difficili". Tali problemi hanno la caratteristica di rimanere NP-completi indipendentemente dai valori numerici eventualmente presenti nelle istanze. Consideriamo ad esempio il seguente problema Taglio.

Taglio
Istanza: Grafo $G = (V, E)$, funzione peso $w : E \mapsto \mathbb{N}, K \in \mathbb{N}$
Predicato: Esiste una partizione (V_1, V_2) di V tale che $\sum_{\substack{u \in V_1 \\ v \in V_2}} w(u, v) \geq K$?

Si può facilmente mostrare che questo problema è NP–completo. Inoltre, è possibile mostrare che Taglio rimane NP–completo anche se ristretto al caso in cui tutti gli archi in G hanno lo stesso peso, unitario. Da questo possiamo osservare che Taglio è un problema computazionalmente difficile (se P $\neq$ NP) indipendentemente dai valori dei pesi associati agli archi.

Consideriamo ora un problema NP–completo diverso, come il problema Bisaccia.

BISACCIA

ISTANZA: Insieme $I = \{e_1, e_2, \ldots, e_n\}$, funzione costo $s : I \mapsto \mathbb{N}$, funzione profitto $p : I \mapsto \mathbb{N}$, $B, K \in \mathbb{N}$

PREDICATO: Esiste un sottoinsieme $I_1 \subseteq I$ tale che $\sum_{e \in I_1} s(e) \leq B$ e $\sum_{e \in I_1} p(e) \geq K$?

Questo problema può essere risolto, come vedremo, in tempo $O(Bn)$ nel modo seguente, utilizzando una tecnica algoritmica nota come *programmazione dinamica*.

Data una istanza $\kappa(n) = \langle I, s, p, B, K \rangle$ di BISACCIA, per ogni $j \leq n$ indichiamo con $\kappa(j) = \langle I', s', p', B, K \rangle$ l'istanza (più semplice) tale che $I' = \{e_1, e_2, \ldots, e_j\}$ e le funzioni s', p' sono le restrizioni di s, p al dominio $I' \subseteq I$.

Un algoritmo di programmazione dinamica per $\kappa(n)$ consiste di n iterazioni, associate alle istanze $\kappa(1), \kappa(2), \ldots, \kappa(n)$. Per ogni istanza $\kappa(j)$ $(j > 1)$, l'algoritmo costruisce un insieme $S(j)$ di al più $B + 1$ soluzioni a partire dal corrispondente insieme $S(j-1)$, relativo all'istanza $\kappa(j-1)$.

L'insieme iniziale $S(1)$ viene definito come $S(1) = \{\{e_1\}, \emptyset\}$: cioè in $S(1)$ vengono incluse le due soluzioni rappresentate dal solo elemento e_1 e dall'insieme vuoto.

Ad ogni iterazione, l'insieme $S(j)$ viene derivato dall'insieme $S(j-1)$ considerando, per ogni soluzione $s \in S(j-1)$, le due soluzioni $s_1 = s$ e $s_2 = s \bigcup \{e_j\}$. Ciò determina la generazione di $2\,|S(j-1)|$ soluzioni.

Consideriamo ora il seguente *criterio di dominanza*, che potremo utilizzare per ridurre l'insieme delle soluzioni da mantenere: se s_1, s_2 sono due soluzioni tali che $\sum_{e \in s_1} s(e) \leq \sum_{e \in s_2} s(e)$ e $\sum_{e \in s_1} p(e) \geq \sum_{e \in s_2} p(e)$, allora possiamo eliminare s_2 da $S(j)$.

L'applicazione del criterio di dominanza consente di mantenere, per ogni possibile dimensione $1 \leq i \leq B$, soltanto al più una soluzione di massimo profitto.

Da quanto detto, deriva che, per ogni $1 \leq j \leq n$, l'algoritmo genera un insieme di al più $B + 1$ soluzioni diverse. È facile rendersi conto che la soluzione di massimo profitto generata comparirà necessariamente in $S(n)$, e sarà quindi individuabile mediante una scansione sequenziale di tale insieme. A questo punto, sarà sufficiente confrontare il profitto di tale soluzione con K per determinare se l'istanza è positiva o negativa.

È possibile visualizzare l'algoritmo di programmazione dinamica in termini di riempimento di una tabella T di n righe e $B+1$ colonne, le cui celle possono contenere o un insieme $I' \subseteq I$ o il simbolo "$-$". In particolare, se $T(j, i) = I'$ allora I' è un sottoinsieme di massimo profitto di $S(j)$ tale che $\sum_{e \in I'} s(e) = i$; se non esiste alcun $I' \subseteq I$ di dimensione i, allora $T(j, i) =$ "$-$". L'Algoritmo 9.3 riempie una tabella di $n(B + 1)$ celle utilizzando tempo costante per ognuna, da cui deriva che la sua complessità è $O(nB)$.

Esempio 9.3 Consideriamo l'istanza di BISACCIA in cui:

```
input I: insieme, s funzione taglia, p funzione profitto, B, K: naturali;
output boolean;
begin
  for j := 1 to n do
    for i := 0 to B do T(j,i) := "-";
  s := s(e_1);
  T(1,0) := ∅;
  T(1,s) := {e_1};
  for j := 2 to n do begin
    s := s(e_j);
    for i := 0 to B do begin
      if T(j-1,i) ≠ "-" then begin
        if T(j,i)= "-" then T(j,i) := T(j-1,i)
          else Inserisci in T(j,i) la soluzione di massimo profitto
            tra T(j-1,i) e T(j,i);
        if i + s ≤ B
          then if T(j,i+s) = "-"
                 then T(j,i+s) := T(j-1,i) ∪ {e_j}
                 else Inserisci in T(j,i+s) la soluzione di massimo profitto
                   tra T(j-1,i) ∪ {e_j} e T(j,i+s);
      end
    end
  end;
  Sia T(n,k) la soluzione di massimo profitto nella riga n-ma di T;
  Sia p̄ il corrispondente profitto;
  if p̄ ≥ K then return VERO
    else return FALSO;
end.
```

Algoritmo 9.3: Algoritmo di programmazione dinamica per BISACCIA.

- $I = \{e_1, e_2, e_3, e_4, e_5, e_6\}$;
- $s(e_1) = 3, s(e_2) = 4, s(e_3) = 1, s(e_4) = 5, s(e_5) = 3, s(e_6) = 2\}$;
- $p(e_1) = 2, p(e_2) = 3, p(e_3) = 1, p(e_4) = 5, p(e_5) = 3, p(e_6) = 1\}$;
- $B = 8, K = 6$.

L'Algoritmo 9.3 applicato a questa istanza riempie una tabella di 6 righe e 9 colonne nel modo seguente.

$\emptyset$	-	-	$\{e_1\}$	-	-	-	-	
$\emptyset$	-	-	$\{e_1\}$	$\{e_2\}$	-	-	$\{e_1, e_2\}$	-
$\emptyset$	$\{e_3\}$	-	$\{e_1\}$	$\{e_2\}$	$\{e_1, e_3\}$	-	$\{e_1, e_2\}$	$\{e_1, e_2, e_3\}$
$\emptyset$	$\{e_3\}$	-	$\{e_1\}$	$\{e_2\}$	$\{e_4\}$	$\{e_3, e_4\}$	$\{e_1, e_2\}$	$\{e_1, e_4\}$
$\emptyset$	$\{e_3\}$	-	$\{e_5\}$	$\{e_3, e_5\}$	$\{e_4\}$	$\{e_3, e_4\}$	$\{e_2, e_5\}$	$\{e_4, e_5\}$
$\emptyset$	$\{e_3\}$	$\{e_6\}$	$\{e_5\}$	$\{e_3, e_5\}$	$\{e_4\}$	$\{e_3, e_4\}$	$\{e_2, e_5\}$	$\{e_4, e_5\}$

La soluzione di profitto massimo nell'ultima riga è $\{e_4, e_5\}$ con profitto $\overline{p} = 8 > K = 6$, per cui l'algoritmo dà risposta VERO.

Esercizio 9.13 Dimostrare che l'Algoritmo 9.3 fornisce la risposta corretta.

Contrariamente a quanto può apparire, l'Algoritmo 9.3 non è polinomiale nella lunghezza dell'istanza. Infatti, una codifica ragionevole di una istanza di BISACCIA prevede:

- $O(\log s(e_i)) = O(\log s_M)$ bit per codificare la dimensione dell'elemento e_i (dove $s_M = \max_{e \in I} s(e)$);
- $O(\log p(e_i)) = O(\log p_M)$ bit per codificare il profitto dell'elemento e_i (dove $p_M = \max_{e \in I} p(e)$);
- $\log B + \log K$ bit per codificare B e K.

Ciò ci fornisce un totale di $O(n(\log s_M + \log p_M) + \log B + \log K)$ bit e, dato che possiamo assumere senza perdita di generalità che $s_M \leq B$ e $p_M \leq K$, l'istanza può essere codificata con $O(n(\log B + \log K))$ bit.

Si noti che in generale nB non è limitata superiormente da nessuna funzione polinomiale di $n(\log B + \log K)$, per cui l'algoritmo risulta avere complessità non polinomiale.

Si noti però che, se si considerano istanze comprendenti soltanto valori "piccoli" di B e, come conseguenza, di tutti gli $s(e_i)$, allora l'Algoritmo 9.3 risulta polinomiale. Ciò vale in particolare nel caso in cui $B = O(n^r)$ per qualche costante r, nel caso cioè in cui tutte le dimensioni presenti nell'istanza sono polinomialmente limitate in n.

In questo caso, quindi, la NP-completezza di BISACCIA deriva non dalla struttura combinatoria del problema, ma dalla (possibile) presenza di numeri molto grandi nell'istanza.

Passiamo ora a generalizzare e formalizzare meglio le considerazioni precedenti, estendendo le considerazioni effettuate a proposito della valutazione della dimensione di un'istanza.

Definizione 9.10 *Dato un problema $\mathcal{P}$, definiamo come* funzione massimo *la funzione $\mu : I_{\mathcal{P}} \mapsto \mathbb{Q}$ che mappa ogni istanza di $\mathcal{P}$ in una stima del valore più grande che compare al suo interno.*

Due coppie $\langle \lambda, \mu \rangle$, $\langle \lambda', \mu' \rangle$ sono polinomialmente correlate se esistono due polinomi bivariati p, p' tali che, per ogni $x \in I_{\mathcal{P}}$:

- λ e λ' sono polinomialmente correlate
- $\mu(x) \leq p'(\lambda'(x), \mu'(x))$
- $\mu'(x) \leq p(\lambda(x), \mu(x))$

Esercizio 9.14 Si considerino, per ogni istanza di PARTIZIONE, le seguenti funzioni lunghezza:

1. $\sum_{e \in I} \log_2 e$
2. $\max_{e \in I} \log_2 e$
3. $n\lceil \log_2 e \rceil$

E le seguenti funzioni massimo:

1. $\max_{e \in I} e$
2. $\sum_{e \in I} e$
3. $\left\lceil \sum_{e \in I} e / \, |I| \right\rceil$

Mostrare che tutte le combinazioni tra funzioni lunghezza e funzioni massimo sono tra loro polinomialmente correlate.

Definizione 9.11 *Dato un problema $\mathcal{P}$ e date due funzioni $\lambda : I_{\mathcal{P}} \mapsto \mathbb{N}$ e $\mu : I_{\mathcal{P}} \mapsto \mathbb{N}$, un algoritmo che risolve $\mathcal{P}$ è detto* pseudo-polinomiale *se, per ogni istanza x, la sua complessità è polinomiale nelle due variabili $\lambda(x)$ e $\mu(x)$.*

Ad esempio l'Algoritmo 9.3 è pseudo-polinomiale, in quanto la sua complessità $O(nB)$ è polinomiale rispetto sia a $\lambda(x)$ che a $\mu(x)$ per ogni istanza x e per ogni coppia di funzioni λ, μ ragionevoli.

Nel caso di TAGLIO, per ogni istanza $x = \langle G = (V, E), w \rangle$ possiamo definire $\mu(x) = \max_{e \in E} w(e)$, mentre, nel caso di BISACCIA, possiamo porre $\mu(x) = \max\{B, K\}$.

Definizione 9.12 *Un problema $\mathcal{P} \in$ NP è detto* numerico *se non esiste alcun polinomio p tale che, per ogni istanza $x \in I_{\mathcal{P}}$, $\mu(x) \leq p(\lambda(x))$.*

Essenzialmente, un problema è numerico se all'interno di sue istanze possono comparire valori indipendenti dalla struttura combinatoria del problema.

Esempio 9.4 Il problema TAGLIO è numerico in quanto i pesi assegnati agli archi possono assumere valori arbitrariamente grandi, indipendentemente dalla struttura del grafo.

Esempio 9.5 Il problema COPERTURA CON NODI non è numerico, in quanto l'unico valore presente in una sua istanza $x = \langle G = (V, E), K \rangle$ è K, che per definizione, sarà $K \leq |V|$, cioè non superiore alla dimensione di un insieme che deve essere enumerato nella descrizione dell'istanza.

Esempio 9.6 Un altro esempio di algoritmo di programmazione dinamica può essere introdotto relativamente ad un altro problema NP-completo, il problema PARTIZIONE, definito come segue.

PARTIZIONE
ISTANZA: Insieme $I = \{e_1, e_2, \ldots, e_n\}$ di naturali
PREDICATO: Esiste una partizione $\{I_1, I_2\}$ di I tale che $\sum_{e \in I_1} e = \sum_{e \in I_2} e = \frac{1}{2} \sum_{e \in I} e$?

L'algoritmo riempie una tabella T di elementi booleani con $n = | I |$ righe e $B = \frac{1}{2} \sum_{e \in I} e$ colonne in modo tale che

$$T(i,j) = \begin{cases} \text{VERO se } \exists I' = \{e_1, e_2, \ldots, e_i\} \text{ tale che } \sum_{e \in I'} e = j \\ \text{FALSO altrimenti} \end{cases}$$

La tabella viene riempita utilizzando le relazioni seguenti:

$$T(1,j) := \text{VERO se} \begin{cases} j = 0 \\ j = e_1 \end{cases}$$

$$T(i,j) := \text{VERO se} \begin{cases} T(i-1,j) = \text{VERO} \\ T(i-1, j-e_i) = \text{VERO} \end{cases} \quad i > 1$$

L'istanza è positiva se $T(n, B) = \text{VERO}$.

Esercizio 9.15 Assumendo noto che il problema BISACCIA sia NP-completo, dimostrare l'NP-completezza di PARTIZIONE.

Chiaramente, non è possibile avere algoritmi pseudo-polinomiali che risolvono un problema non numerico NP–completo, a meno che P = NP.

Dato un problema $\mathcal{P} \in$ NP e dato un polinomio p, indichiamo con $\mathcal{P}^p$ il sottoproblema ottenuto da $\mathcal{P}$ considerando soltanto le istanze x per cui $\mu(x) \leq p(\lambda(x))$. Per ipotesi, $\mathcal{P}^p$ non è un problema numerico. Inoltre, se $\mathcal{P}$ è risolubile da un algoritmo pseudo-polinomiale allora $\mathcal{P}^p \in$ P.

Definizione 9.13 *Un problema $\mathcal{P}$ è detto* fortemente NP*–completo se $\mathcal{P} \in$ NP e se esiste un polinomio p tale che $\mathcal{P}^p$ è* NP*–completo.*

Si osservi che tutti i problemi NP–completi non numerici sono necessariamente fortemente NP–completi (si consideri il polinomio $p(n) = 1$).

Essenzialmente, possiamo dire che in un problema fortemente NP–completo la difficoltà di soluzione deriva dalla stessa complessità strutturale delle istanze del problema, al contrario dei problemi non fortemente NP–completi, risolubili mediante algoritmi pseudo-polinomiali.

Teorema 9.12 *Un problema fortemente* NP*–completo non può essere risolto da un algoritmo pseudo–polinomiale, a meno che* P = NP*.*

Dimostrazione. Supponiamo per assurdo che esista un algoritmo pseudo-polinomiale $\mathcal{A}$ che risolve il problema $\mathcal{P}$ fortemente NP–completo.

Da ciò deriva che $\mathcal{A}$ risolve una istanza generica $x \in I_{\mathcal{P}}$ in tempo $p(\lambda(x), \mu(x))$, dove p è un opportuno polinomio. Ma allora, per ogni polinomio q, $\mathcal{P}^q$ può essere risolto in tempo $p(\lambda(x), q(\lambda(x)))$, e quindi in tempo polinomiale, contrariamente alle ipotesi di NP–completezza forte di $\mathcal{P}$. □

Esempio 9.7 Il problema COMMESSO VIAGGIATORE è definito come segue.

COMMESSO VIAGGIATORE

ISTANZA: Insieme $C = \{c_1, c_2, \ldots, c_n\}$ (città), matrice simmetrica D $n \times n$ di razionali (distanze), $B \in \mathbb{Q}$

PREDICATO: Esiste un circuito che attraversa tutte le città ed ha lunghezza non superiore a B, vale a dire, esiste una permutazione $C' = \{c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(n)}\}$ di C tale che $\sum_{i=1}^{n} D(\pi(i), \pi(i+1)) + D(\pi(n), \pi(1)) \leq B$?

COMMESSO VIAGGIATORE è un problema numerico, in quanto i valori in D sono indipendenti dal resto dell'istanza. È possibile comunque provare che esso è fortemente NP–completo mostrando una Karp–riduzione polinomiale dal problema NP–completo CICLO HAMILTONIANO al sottoproblema COMMESSO VIAGGIATORE-{1,2} di COMMESSO VIAGGIATORE in cui tutti i valori in D sono ristretti ad avere dominio $\{1, 2\}$.

Data una istanza $G = (V, E)$ di CICLO HAMILTONIANO, costruiamo da essa una istanza di COMMESSO VIAGGIATORE-{1,2} nel modo seguente:

- $C = V$;
- per ogni coppia c_i, c_j, $D(i, j) = D(j, i) = 1$ se e solo se $(v_i, v_j) \in E$, altrimenti $D(i, j) = D(j, i) = 2$;
- $B = n$.

Chiaramente, esiste un ciclo hamiltoniano in G se e solo se esiste un circuito di lunghezza B tra gli elementi di C.

9.6 La gerarchia polinomiale

Riprendendo le considerazioni relative alla caratterizzazione della classe NP, fatte nella Sezione 9.3, possiamo ora mostrare come le classi NP e CO-NP costituiscanoClasse di complessità!CO-NP—(solo il primo livello di una gerarchia di classi di problemi di decisione, aventi struttura combinatoria via via più complessa.

Ricordando quanto detto nella Sezione 9.3, possiamo dire che un problema di decisione $\mathcal{P}$ è in NP se e solo se esiste un predicato $P(x, y)$ decidibile in tempo polinomiale tale che $x \in Y_{\mathcal{P}}$ se e solo se $\exists y, |y| \leq p(|x|) : P(x, y)$, dove quindi y è il certificato relativo all'istanza positiva x.

In tal caso, il non determinismo della macchina di Turing non deterministica viene utilizzato per "indovinare" un valore della variabile quantificata per il quale il predicato è vero.

Ad esempio, consideriamo il seguente problema.

INSIEME INDIPENDENTE
ISTANZA: Grafo $G = (V, E)$, $K > 0$ intero
PREDICATO: Esiste un insieme indipendente di taglia $\geq K$, cioè un $U \subseteq V$ tale che $\mid U \mid \geq K$ e $\nexists (u, v) \in E$ con $u \in U \wedge v \in U$?

Nel caso di INSIEME INDIPENDENTE la stringa x è costituita dalla rappresentazione del grafo G e del valore K, mentre il certificato y è la rappresentazione di un particolare sottoinsieme $U \subseteq V$. La verifica espressa dal predicato $P(x, y)$ richiede di accertarsi che $\mid U \mid \geq K$ e che U è un insieme indipendente. Una computazione non deterministica per questo problema consiste di un passo non deterministico di scelta (*guess*) di un sottoinsieme di V, seguito da una verifica in tempo deterministico polinomiale.

Una ulteriore, interessante interpretazione dei problemi in NP vede tali problemi come *giochi* di una sola mossa, eseguita da un giocatore (A) in cui ci si chiede se A, muovendo, può raggiungere una posizione vincente, assunte le non restrittive ipotesi che il valutare se una posizione è vincente per A richieda tempo deterministico polinomiale[6] e che una mossa sia descrivibile in modo sufficientemente succinto rispetto alla descrizione di una posizione.

Se ora consideriamo l'insieme $Y_{\mathcal{P}}$ di un qualche problema di decisione $\mathcal{P} \in$ CO-NP (istanze negative di un problema in NP), vediamo facilmente che esso potrà essere descritto mediante una formula del tipo

$$\nexists y, \mid y \mid \leq p(\mid x \mid) : P(x, y) \equiv \forall y, \mid y \mid \leq p(\mid x \mid) : \neg P(x, y) \equiv$$
$$\equiv \forall y, \mid y \mid \leq p(\mid x \mid) : P'(x, y)$$

dove $P'(x, y)$ è il predicato negato di $P(x, y)$, anch'esso, per definizione, verificabile in tempo deterministico polinomiale. Quindi, possiamo vedere i problemi in CO-NP come giochi di una mossa in cui ci si chiede se l'unico giocatore che muove (A), facendo la sua mossa, non può raggiungere in nessun modo una posizione vincente.

Al tempo stesso, come caso limite, un problema in P può chiaramente essere visto come la verifica di una formula del tipo $P(x)$, priva cioè di quantificatori, e quindi come giochi in cui ci si chiede se la posizione raggiunta è una posizione vincente per A.

A partire dalle considerazioni precedenti, possiamo allora definire due nuove gerarchie di classi di complessità, di cui P ed NP rappresentano i livelli di ordine 0 ed 1 nella prima, e P e CO-NP i livelli di ordine 0 ed 1 nella seconda..

[6] D'ora in poi, assumeremo sempre che il predicato P sia valutabile in tempo deterministico polinomiale.

Tale gerarchia è definita dalle classi di problemi i cui insiemi di istanze positive sono descrivibili mediante formule del tipo

$$\exists y_1 \forall y_2 \exists y_3 \forall y_4 \ldots Q y_k P(x, y_1, y_2, y_3, y_4, \ldots, y_k)$$

dove k è un intero non negativo, e Q è $\exists$ se k è dispari e $\forall$ se k è pari, ed in ogni caso $|\, y_i \,| \leq p(|\, x \,|)$ per un opportuno polinomio p.

In generale, una formula composta da un predicato preceduto da una sequenza di quantificatori esistenziali ed universali alternati come

$$\exists y_1 \forall y_2 \exists y_3 \forall y_4 \exists y_5 \ldots P(x, y_1, y_2, y_3, y_4, y_5, \ldots)$$

può essere interpretata come un gioco ad informazione perfetta (ad esempio come scacchi o dama) tra due giocatori A e B, in cui le variabili quantificate rappresentano le mosse eseguite a turno, la variabile non quantificata x rappresenta la posizione di partenza ed il predicato P rappresenta la condizione di vittoria per A, che assumiamo giochi per primo. Dire che A ha una strategia vincente significa affermare che:

esiste una mossa y_1 *di* A *tale che*
per ogni mossa y_2 *di* B
esiste una mossa y_3 *di* A *tale che*
per ogni mossa y_4 *di* B
esiste una mossa y_5 *di* A *tale che*
$\vdots$
il giocatore A *vince.*

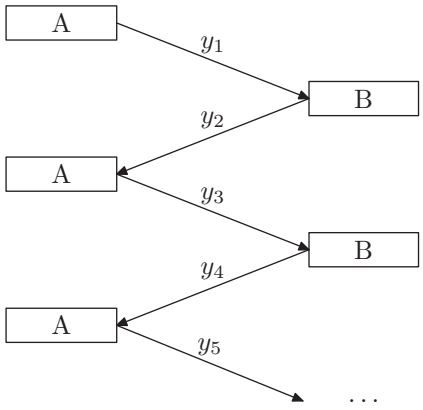

FIGURA 9.11 Gioco tra A e B

Quindi, la formula è verificata se e solo se A ha una strategia vincente nel gioco.

Si noti che se la formula è del tipo

$$\forall y_1 \exists y_2 \forall y_3 \exists y_4 \exists y_5 \ldots P(x, y_1, y_2, y_3, y_4, y_5, \ldots)$$

allora la formula è vera se e solo se B ha una strategia vincente nel gioco (tutte le strategie sono perdenti per A).

Definizione 9.14 *Per ogni problema di decisione $\mathcal{P}$, $\mathcal{P} \in \Sigma_k^p$ ($k \geq 0$) se e solo se esiste un predicato $P(x, y_1, \ldots, y_k) \in$ P ed un polinomio $p(\cdot)$ tali che, per ogni istanza x di $\mathcal{P}$, $x \in Y_{\mathcal{P}}$ se e solo se $\exists y_1 \forall y_2 \ldots Q y_k \; P(x, y_1, \ldots, y_k)$ è vera, dove $|y_i| \leq p(|x|)$, $i = 1, \ldots, k$ e $Q = \exists$ per k dispari, $Q = \forall$ per k pari.*

La seconda gerarchia che introduciamo è definita dalle classi di problemi i cui insiemi di istanze positive sono descrivibili mediante formule del tipo

$$\forall y_1 \exists y_2 \forall y_3 \exists y_4 \ldots Q s y_k P(x, y_1, y_2, y_3, y_4, \ldots, y_k)$$

dove k è un intero non negativo, e Q è $\forall$ se k è dispari e $\exists$ se k è pari.

Definizione 9.15 *Per ogni problema di decisione $\mathcal{P}$, $\mathcal{P} \in \Pi_k^p$ ($k \geq 0$) se e solo se esiste un predicato $P(x, y_1, \ldots, y_k) \in$ P ed un polinomio $p(\cdot)$ tali che, per ogni istanza x di $\mathcal{P}$, $x \in Y_{\mathcal{P}}$ se e solo se $\forall y_1 \exists y_2 \ldots Q y_k \; P(x, y_1, \ldots, y_k)$ è vera, dove $|y_i| \leq p(|x|)$, $i = 1, \ldots, k$ e $Q = \forall$ per k dispari, $Q = \exists$ per k pari.*

Si può banalmente verificare che per $k = 0$ si ha effettivamente $\Sigma_0^p = \Pi_0^p =$ P. Allo stesso modo, è immediato che $\Sigma_1^p =$ NP e $\Pi_1^p =$ co-NP.

Inoltre, vale il seguente teorema, generalizzazione di quanto già mostrato per NP e co-NP.

Teorema 9.13 *Per ogni $k \geq 0$, $\Pi_k^p =$ co-Σ_k^p.*

Dimostrazione. Per ogni problema $\mathcal{P} \in \Sigma_k^p$ ogni istanza $x \in N_{\mathcal{P}} = Y_{\overline{\mathcal{P}}}$ verifica la formula

$$\neg(\exists y_1 \forall y_2 \ldots Q y_k \; P(x, y_1, \ldots, y_k)) \equiv \forall y_1 \exists y_2 \ldots Q' y_k \; P'(x, y_1, \ldots, y_k)$$

dove $P'(\cdot)$ è il predicato negato di $P(\cdot)$ e $Q' = \exists$ se $Q = \forall$, $Q' = \forall$ se $Q = \exists$.

Da ciò deriva che $\forall \mathcal{P} \in \Sigma_k^p : \overline{\mathcal{P}} \in \Pi_k^p$, e quindi che co-$\Sigma_k^p \subseteq \Pi_k^p$.

Inoltre, per ogni problema $\mathcal{P} \in \Pi_k^p$ ogni istanza $x \in Y_{\mathcal{P}}$ verifica la formula

$$\forall y_1 \exists y_2 \ldots Q y_k \; P(x, y_1, \ldots, y_k) \equiv \neg(\exists y_1 \forall y_2 \ldots Q' y_k) \; P'(x, y_1, \ldots, y_k)$$

dove, nuovamente, P' è il predicato negato di P e $Q' = \exists$ se $Q = \forall$, $Q' = \forall$ se $Q = \exists$. Ma l'insieme delle x tali che $\neg(\exists y_1 \forall y_2 \ldots Q' y_k) \; P'(x, y_1, \ldots, y_k)$ rappresenta, per definizione, le istanze negative di un qualche problema $\mathcal{P}' \in \Sigma_k^p$.

Da ciò consegue che $\forall \mathcal{P} \in \Pi_k^p \; \exists \mathcal{P}' \in \Sigma_k^p : \mathcal{P} = \overline{\mathcal{P}}'$, e quindi che $\Pi_k^p \subseteq$ co-Σ_k^p. □

Il seguente corollario deriva immediatamente notando che, per due qualunque classi di complessità $\mathcal{A}$, $\mathcal{B}$, $\mathcal{A} = \mathcal{B} \Rightarrow$ co-$\mathcal{A}$ = co-$\mathcal{B}$.

Corollario 9.14 *Per ogni* $k \geq 0$, $\Sigma_k^p = co\text{-}\Pi_k^p$.

Ai fini di quanto seguirà introduciamo ora la seguente notazione.

Definizione 9.16 *Data una classe di complessità* $\mathcal{C}$, *indichiamo con* $\exists\mathcal{C}$ *l'insieme dei problemi di decisione tale che* $\Pi \in \exists\mathcal{C}$ *se e solo se esistono un polinomio* $p(\cdot)$ *ed un predicato binario* P *in* $\mathcal{C}$ *tali che* $Y_\Pi = \{x \mid \exists y\, P(x,y)\}$, *dove* $|y| \leq p(|x|)$.

Allo stesso modo, possiamo definire $\forall\mathcal{C}$.

Proposizione 9.15 *Le relazioni seguenti derivano immediatamente dalle definizioni.*

1. $\Sigma_k^p = \underbrace{\exists\forall\ldots Q}_{k}\,\mathrm{P}$
2. $\Pi_k^p = \underbrace{\forall\exists\ldots Q}_{k}\,\mathrm{P}$
3. $\exists\Sigma_k^p = \Sigma_k^p$
4. $\forall\Pi_k^p = \Pi_k^p$

Teorema 9.16 *Per ogni* $k \geq 0$ $\Sigma_k^p \cup \Pi_k^p \subseteq \Sigma_{k+1}^p \cap \Pi_{k+1}^p$.

Dimostrazione. Sia $\mathcal{P} \in \Sigma_k^p$. Allora $Y_\mathcal{P} = \{x \mid \exists y_1 \ldots Q y_k\, P(y_1, \ldots, y_k, x)\}$, ma anche $Y_\Pi = \{x \mid \exists y_1 \ldots Q y_k Q' y_{k+1}\, P(y_1, \ldots, y_k, x)\}$ (dove $Q' = \exists$ se e solo se $Q = \forall$) e quindi $\mathcal{P} \in \Sigma_{k+1}^p$. Inoltre, $Y_\mathcal{P} = \{x \mid \forall y_{k+1}\exists y_1 \ldots Q y_k\, P(y_1, \ldots, y_k, x)\}$ e quindi $\mathcal{P} \in \Pi_{k+1}^p$.

Le stesse considerazioni possono essere applicate se $\mathcal{P} \in \Pi_k^p$. □

Il diagramma in Figura 9.12 mostra la struttura delle inclusioni per i primi livelli delle classi Σ_k^p e Π_k^p.

Definiamo come *gerarchia polinomiale* PH l'unione di tutte le classi Σ_k^p, $\mathrm{PH} = \cup_k \Sigma_k^p$: dalle relazioni di inclusione mostrate in Figura 9.12 risulta anche chiaro che $\mathrm{PH} = \cup_k \Pi_k^p$.

Per nessuna delle inclusioni si sa se è propria o meno, anche se delle dipendenze tra le diverse relazioni di inclusioni possono essere dimostrate.

Teorema 9.17 *Per ogni* $k \geq 1$ *le seguenti relazioni sono equivalenti:*

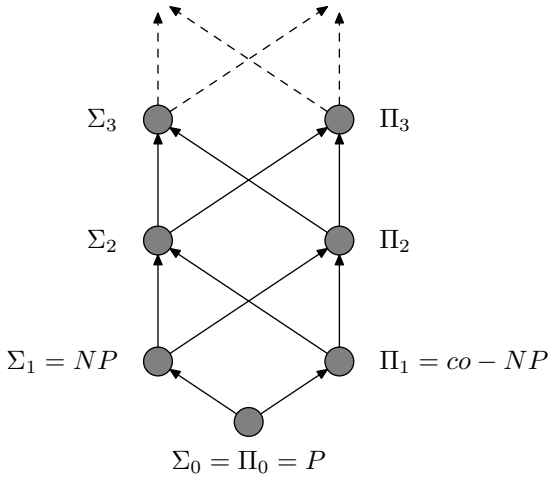

FIGURA 9.12 Relazioni di inclusione in PH

---

1. $\Sigma_k^p = \Sigma_{k+1}^p$
2. $\Pi_k^p = \Pi_{k+1}^p$
3. $\Sigma_k^p = \Pi_k^p$
4. $\Sigma_k^p = \Pi_{k+1}^p$
5. $\Pi_k^p = \Sigma_{k+1}^p$

Dimostrazione. 1 e 2 sono equivalenti per complementazione. Lo stesso vale per 4 e 5.

Mostriamo ora che 1, 3 e 4 sono equivalenti tra loro.

- 1⇒3 deriva da $\Sigma_k^p \subseteq \Pi_{k+1}^p$co-$\Sigma_{k+1}^p$ (dal Teorema 9.16) e co-Σ_{k+1}^p = co-$\Sigma_k^p = \Pi_k^p$ (da 1), che implicano $\Sigma_k^p \subseteq \Pi_k^p$, e da $\Pi_k^p \subseteq \Sigma_{k+1}^p =$ co-Π_{k+1}^p (dal Teorema 9.16) e co-$\Pi_{k+1}^p = \Sigma_{k+1}^p = \Sigma_k^p$ (da 1), che implicano $\Pi_k^p \subseteq \Sigma_k^p$;
- 3⇒4 deriva da $\Sigma_k^p = \Pi_k^p$ (da 3), $\Pi_k^p = \forall\Pi_k^p$ (dalla Proposizione 9.15), da $\forall\Pi_k^p = \forall\Sigma_k^p$ (da 3) e da $\forall\Sigma_k^p = \Pi_{k+1}^p$ per definizione;
- 4⇒1 deriva da $\Sigma_{k+1}^p =$ co-Π_{k+1}^p (per definizione), co-Π_{k+1}^p = co-Σ_k^p (da 4), co-$\Sigma_k^p = \Pi_k^p \subseteq \Pi_{k+1}^p$ (per definizione) e $\Pi_{k+1}^p = \Sigma_k^p$ (da 4).

□

Teorema 9.18 *Per ogni $k \geq 1$ le condizioni 1, 2, 3, 4, 5 del teorema 9.17 sono equivalenti a* $\mathrm{PH} = \Sigma_k^p$.

Dimostrazione. Data l'equivalenza, già mostrata, tra le condizioni nel Teorema 9.17, considereremo soltanto l'equivalenza tra $\Sigma_k^p = \Sigma_{k+1}^p$ e $\mathrm{PH} = \Sigma_k^p$.

È immediato che da $\mathrm{PH} = \Sigma_k^p$ deriva necessariamente $\Sigma_k^p = \Sigma_{k+1}^p$. L'implicazione opposta deriva osservando che, se $\Sigma_k^p = \Sigma_{k+1}^p$, allora $\Pi_k^p = \Pi_{k+1}^p$ (dal Teorema 9.17) e $\Sigma_{k+2}^p = \exists\Pi_{k+1}^p = \exists\Pi_k^p = \Sigma_{k+1}^p = \Sigma_k^p$ e, per induzione, si ha che per ogni $i \geq k$ $\Sigma_i^p = \Sigma_k^p$ e, dato che $\cup_{i \leq k}\Sigma_i^p = \Sigma_k^p$ per le relazioni di inclusione già mostrate, ne deriva $\mathrm{PH} = \Sigma_k^p$. □

È quindi sufficiente che, per un qualunque livello k della gerarchia si abbia $\Sigma_k^p = \Sigma_{k+1}^p$ (o una qualunque delle relazioni equivalenti del Teorema 9.17) perché risulti che tutta la gerarchia polinomiale sia uguale a Σ_k^p: tale situazione viene detta *collasso* della gerarchia polinomiale al livello k.

Si noti anche che, al contrario, se, per qualche k, $\Sigma_k^p \neq \Sigma_{k+1}^p$ (la gerarchia polinomiale non collassa) allora $\Sigma_0^p = \mathrm{P} \neq \Sigma_1^p = \mathrm{NP} \neq \ldots \neq \Sigma_k^p \neq \Sigma_{k+1}^p$. Ne deriva, tra l'altro, che ogni condizione $\Sigma_k^p \neq \Sigma_{k+1}^p$ (o, ad esempio, $\Sigma_k^p \neq \Pi_k^p$) con $k \geq 1$ implica $\mathrm{P} \neq \mathrm{NP}$: si osservi che, come caso particolare, ciò implica il già noto risultato che $\mathrm{NP} \neq \text{co-NP} \Rightarrow \mathrm{P} \neq \mathrm{NP}$. Inoltre, da quanto mostrato deriva anche che ogni condizione $\Sigma_k^p \neq \Sigma_{k+1}^p$ (o $\Sigma_k^p \neq \Pi_k^p$) con $k \geq 2$ implica $\mathrm{NP} \neq \text{co-NP}$.

Una definizione alternativa della gerarchia polinomiale può essere fornita facendo riferimento alla *relativizzazione* di classi di complessità.

Definizione 9.17 *Dato un problema $\mathcal{P}$, definiamo la classe relativizzata* $\mathrm{P}^{\mathcal{P}}$ *come l'insieme dei problemi risolubili in tempo polinomiale da un macchina di Turing deterministica con oracolo per $\mathcal{P}$.*

Si noti che $\mathcal{P}' \in \mathrm{P}^{\mathcal{P}}$ è equivalente a $\mathcal{P}' \leq_T^p \mathcal{P}$.

Definizione 9.18 *Dato un problema $\mathcal{P}$, definiamo la classe relativizzata* $\mathrm{NP}^{\mathcal{P}}$ *come l'insieme dei problemi risolubili in tempo polinomiale da un macchina di Turing non deterministica con oracolo per $\mathcal{P}$.*

Le definizioni precedenti possono essere estese a classi relativizzate rispetto ad altre classi.

Definizione 9.19 *Data una classe A, un problema $\mathcal{P}'$ appartiene alla classe relativizzata* P^A *se esiste un problema $\mathcal{P} \in A$ tale che $\mathcal{P} \in \mathrm{P}^{\mathcal{P}}$. Cioè,* P^A *è l'insieme dei problemi risolubili in tempo polinomiale da un macchina di Turing deterministica con oracolo per un opportuno problema $\mathcal{P} \in A$.*

Definizione 9.20 *Data una classe A, un problema $\mathcal{P}'$ appartiene alla classe relativizzata NP^A se esiste un problema $\mathcal{P} \in A$ tale che $\mathcal{P} \in \mathrm{NP}^{\mathcal{P}}$. Cioè, NP^A è l'insieme dei problemi risolubili in tempo polinomiale da un macchina di Turing non deterministica con oracolo per un opportuno problema $\mathcal{P} \in A$.*

Si noti che alcune classi già definite possono essere interpretate anche come classi relativizzate: ad esempio, $\mathrm{P} = \mathrm{P}^{\mathrm{P}}$ e $\mathrm{NP} = \mathrm{NP}^{\mathrm{P}}$.

A partire da operazioni di relativizzazione operate a partire dalle classi P e NP, possiamo definire la seguente gerarchia di classi.

Definizione 9.21 *Per ogni intero $k \geq 0$, la classe $\overline{\Sigma}_k^p$ è definita come $\overline{\Sigma}_k^p = \mathrm{P}$ se $k = 0$ e $\overline{\Sigma}_k^p = \mathrm{NP}^{\overline{\Sigma}_{k-1}^p}$ se $k > 0$.*

Definizione 9.22 *Per ogni intero $k \geq 0$, la classe $\overline{\Pi}_k^p$ è definita come co-$\overline{\Sigma}_k^p$.*

Definizione 9.23 *Per ogni intero $k \geq 0$, la classe $\overline{\Delta}_k^p$ è definita come $\overline{\Delta}_k^p = \mathrm{P}$ se $k = 0$ e $\overline{\Delta}_k^p = \mathrm{P}^{\overline{\Sigma}_{k-1}^p}$ se $k > 0$.*

Esercizio 9.16 Mostrare che dalle precedenti definizioni deriva che $\overline{\Sigma}_0^p = \overline{\Pi}_0^p = \overline{\Delta}_0^p = \mathrm{P}$, $\overline{\Sigma}_1^p = \mathrm{NP}$, $\overline{\Pi}_1^p = \text{co-NP}$, $\overline{\Delta}_1^p = \mathrm{P}$, $\overline{\Sigma}_2^p = \mathrm{NP}^{\mathrm{NP}}$, $\overline{\Pi}_2^p = \text{co-NP}^{\mathrm{NP}}$, $\overline{\Delta}_2^p = \mathrm{P}^{\mathrm{NP}}$.

L'equivalenza tra le classi Σ_k^p, Π_k^p, che definiscono la gerarchia polinomiale, e le classi $\overline{\Sigma}_k^p$, $\overline{\Pi}_k^p$ deriva dal seguente teorema, la cui dimostrazione è lasciata per esercizio.

Teorema 9.19 *Dato un linguaggio L e dato un intero $k \geq 1$, $L \in \overline{\Sigma}_k^p$ se e solo se $L \in \Sigma_k^p$.*

Dimostrazione. La dimostrazione è lasciata per esercizio (vedi Esercizio 9.17). □

Esercizio 9.17 Dimostrare il Teorema 9.19.

La caratterizzazione mediante formule booleane quantificate con alternanza di quantificatori permette di definire anche una sequenza di problemi completi (rispetto alla Karp-riducibilità) per le classi Σ_k^p.

Per ogni intero $i \geq 1$, definiamo il problema Formula Booleana Quantificata i-limitata nel modo seguente:

FORMULA BOOLEANA QUANTIFICATA i-LIMITATA

ISTANZA: Una formula booleana $\mathcal{F}$ su una sequenza $X = \{x_1, x_2, \ldots, x_i\}$ di variabili booleane.

PREDICATO: La formula booleana quantificata

$$\forall x_1 \exists x_2 \forall x_3 \ldots Q x_i \mathcal{F}(x_1, x_2, \ldots, x_i)$$

è vera?

Teorema 9.20 *Per ogni $i \geq 1$,* FORMULA BOOLEANA QUANTIFICATA i-LIMITATA *è Σ_i^p-completo rispetto alla Karp-riducibilità polinomiale.*

Dimostrazione. La dimostrazione è lasciata come esercizio (vedi Esercizio 9.18). □

Esercizio 9.18 Dimostrare il Teorema 9.20.

9.7 La classe PSPACE

La classe PSPACE, definita come l'insieme dei problemi risolubili in spazio polinomiale nella dimensione dell'istanza, viene a costituire il primo e più naturale esempio di classe di complessità che non collassa a P nel caso in cui P=NP: vale a dire che P = NP non implica PSPACE = P. Inoltre, la caratterizzazione di problemi mediante spazio di risoluzione polinomiale presenta l'interessante caratteristica di essere invariante rispetto all'introduzione del non determinismo: infatti, in conseguenza del Teorema di Savitch (Teorema 8.26), ogni problema risolubile in spazio polinomiale da macchine di Turing non deterministiche è risolubile in spazio polinomiale anche da macchine di Turing deterministiche, vale a dire che NSPACE=DSPACE.

Come vedremo, la classe PSPACE rappresenta inoltre l'insieme di linguaggi cui tende la Gerarchia Polinomiale PH al crescere indefinito del numero di quantificatori alternati nella formula, o, corrispondentemente, della lunghezza della sequenza di relativizzazioni operate.

Possiamo verificare l'inclusione di PH in PSPACE come conseguenza del Teorema seguente, che identifica, corrispondentemente a quanto effettuato nei Teoremi 9.2 e 9.4 per le classi P ed NP, un problema completo in PSPACE iniziale.

Sia FORMULA BOOLEANA QUANTIFICATA il problema definito nel modo seguente.

FORMULA BOOLEANA QUANTIFICATA

ISTANZA: Una formula booleana $\mathcal{F}$ su una sequenza $X = \{x_1, x_2, \ldots, x_n\}$ di variabili booleane.

PREDICATO: La seguente formula booleana quantificata Φ:

$$\Phi = Q_1 x_1 Q_2 x_2 Q_3 x_3 \ldots Q_n x_n \mathcal{F}(x_1, x_2, \ldots, x_n),$$

dove, per ogni i, $Q_i \in \{\exists, \forall\}$, $Q_i = \exists \Rightarrow Q_{i+1} = \forall$ e $Q_i = \forall \Rightarrow Q_{i+1} = \exists$, è vera?

Come si può immediatamente vedere, FORMULA BOOLEANA QUANTIFICATA è una generalizzazione di tutti i problemi FORMULA BOOLEANA QUANTIFICATA i-LIMITATA per ogni $i > 0$. Mostriamo ora che vale il seguente teorema.

Teorema 9.21 *Il problema* FORMULA BOOLEANA QUANTIFICATA *è* PSPACE*-completo rispetto alla Karp–riducibilità log-space.*

Dimostrazione. Per mostrare che FORMULA BOOLEANA QUANTIFICATA $\in$ PSPACE introduciamo una procedura ricorsiva *Val* che risolve il problema su una istanza generica $\Phi = \exists x_1 \forall x_2\ \exists x_3\ \ldots Q x_m \mathcal{F}(x_1, x_2, \ldots, x_n)$.

La procedura *Val* ha la struttura seguente, e viene applicata inizialmente per $\phi = \Phi$:

- se $\phi =$ TRUE allora $Val(\phi) =$ TRUE;
- $\phi =$ FALSE allora $Val(\phi) =$ FALSE;
- $\phi = \overline{\psi}$ allora $Val(\phi) = \overline{Val(\psi)}$;
- $\phi = \psi \wedge \psi'$ allora $Val(\phi) = Val(\psi) \wedge Val(\psi')$;
- $\phi = \psi \vee \psi'$ allora $Val(\phi) = Val(\psi) \vee Val(\psi')$;
- $\phi = \forall x \psi$ allora $Val(\phi) = Val(\psi\ |_{x=\text{TRUE}}) \wedge Val(\psi\ |_{x=\text{FALSE}})$;
- $\phi = \exists x \psi$ allora $Val(\phi) = Val(\psi\ |_{x=\text{TRUE}}) \vee Val(\psi\ |_{x=\text{FALSE}})$.

Con la notazione $Val(\psi\ |_{x=k})$ intendiamo la applicazione della procedura *Val* sulla formula ψ, in cui la variabile x è posta uguale a k. Chiaramente, la formula Φ rappresenta un'istanza positiva di FORMULA BOOLEANA QUANTIFICATA se e solo se $Val(\Phi)$ restituisce il valore TRUE.

Si osservi che la profondità massima della ricursione è pari al numero m di variabili, ed è limitata superiormente da n, la dimensione della formula, per cui una pila di n elementi è sufficiente ad implementare la procedura. Si osservi inoltre che ogni elemento di tale pila è sufficiente che rappresenti una

copia della formula modificata per rappresentare le assegnazioni effettuate fino ad ora nell'ambito della valutazione, per una dimensione proporzionale ad n. In definitiva, uno spazio $O(n^2)$ è sufficiente per implementare la procedura *Val*, dal che deriva che FORMULA BOOLEANA QUANTIFICATA $\in$ PSPACE.

Per mostrare la completezza di FORMULA BOOLEANA QUANTIFICATA in PSPACE è ora necessario mostrare che per ogni problema $\mathcal{P} \in$ PSPACE vale la proprietà $\mathcal{P} \leq_m^{logsp}$FORMULA BOOLEANA QUANTIFICATA.

Come nel caso del Teorema di Cook (Teorema 9.4), costruiremo, a partire da una macchina di Turing $\mathcal{M}$[7] e da una stringa di input x, una formula booleana quantificata che è soddisfacibile se e solo se x viene accettata da $\mathcal{M}$ in spazio polinomiale.

Sia $p(n)$ il polinomio che definisce la limitazione di spazio utilizzabile da $\mathcal{M}$ e si osservi che, se x ha lunghezza n, il tableau relativo ad una computazione effettuata da $\mathcal{M}$ può avere $p(n)$ colonne e $c^{p(n)}$ righe per qualche costante c,[8] il che rende ad esempio la costruzione mostrata nella dimostrazione del Teorema 9.4 non applicabile, in quanto la formula CNF risultante avrebbe dimensione esponenziale.

Per brevità, utilizzeremo nel seguito delle variabili $X_1, X_2, \ldots$ che rappresentano codifiche di righe di tableau, effettuate secondo le modalità descritte nella dimostrazione del Teorema 9.4. Vale a dire che ognuna di tali variabili corisponde in realtà a $O(p(n))$ variabili booleane del tipo $S(i,j,k)$ e $C(i,j,k)$. Si assume inoltre che, in corrispondenza all'uso di una variabile X_i di tal tipo, nella formula sia presente una formula $\phi^M(X_i)$, della stessa struttura delle formule ϕ_i^M introdotte nella dimostrazione del Teorema 9.4, che specifica che l'assegnazione effettuata sull'insieme delle variabili $S(i,j,k)$ e $C(i,j,k)$ in X_i rappresenta una configurazione di $\mathcal{M}$.

$$\phi(x) = \exists X_1 \exists X_2 (\phi^M(X_1) \wedge \phi^M(X_2) \wedge \phi^I(X_1, x) \wedge \phi^A(X_2) \wedge \phi^T(X_1, X_2, c^{p(n)}))$$

dove valgono le proprietà seguenti:

- la formula $\phi^I(X_1, x)$ ha la stessa struttura della formula ϕ^I nella dimostrazione del Teorema 9.4, ed è verificata se e solo se si ha una assegnazione sulle variabili booleane in X_1 che codifica la prima riga del tableau con stringa x, e quindi la configurazione iniziale di $\mathcal{M}$ in cui x si trova sul nastro.

- la formula $\phi^A(X_2)$ ha la stessa struttura della formula ϕ^A nella dimostrazione del Teorema 9.4, ed è verificata se e solo se si ha una assegnazione sulle variabili booleane in X_2 che codifica una configurazione accettante di $\mathcal{M}$.

[7] Come nel Teorema 9.4 si assume che $\mathcal{M}$ abbia un solo nastro semi-infinito e che cicli indefinitamente su ogni stato finale.

[8] Nel seguito si assume, senza perdita di generalità, che $c^{p(n)}$ sia una potenza di due, vale a dire che esista un intero k tale che $c^{p(n)} = 2^k$.

- la formula $\phi^T(X_1, X_2, c^{p(n)})$ è soddisfatta da ogni assegnazione che rappresenta una computazione in cui $\mathcal{M}$ passa dalla configurazione X_1 alla configurazione X_2 in al più $c^{p(n)}$ passi.

  Applicando la costruzione utilizzata nella dimostrazione del Teorema 8.26, una formula $\phi^T(X_1, X_2, 2t)$ può essere definita nel modo seguente:

$$\phi^T(X_1, X_2, 2t) = \exists X_3(\phi^M(X_3) \wedge \phi^T(X_1, X_3, t) \wedge \phi^T(X_3, X_2, t)).$$

  Dove $\phi^T(X_1, X_2, 1)$ ha la struttura della formula ϕ^T nella dimostrazione del Teorema 9.4.

  Applicando la definizione precedente, la formula $\phi^T(X_1, X_2, c^{p(n)})$ risulta avere lunghezza esponenziale $O(2^{p(n)})$.

  Consideriamo ora una diversa definizione di $\phi^T(X_1, X_2, 2t)$, che utilizza anche quantificatori universali.

$$\phi^T(X_1, X_2, 2t) = \forall X \forall Y \exists X_3 \left(\phi^M(X_3) \wedge \right.$$

$$\left. \wedge \left[((X = X_1 \wedge Y = X_3) \vee (X = X_3 \wedge Y = X_2)) \Rightarrow \phi^T(X, Y, t)\right]\right).$$

  Come si può vedere, data questa definizione, la lunghezza della formula $\phi^T(X_1, X_2, 2t)$ è pari alla lunghezza della formula $\phi^T(X_1, X_2, t)$ più $O(p(n) \log n)$. Da ciò deriva che la lunghezza di $\phi^T(X_1, X_2, c^{p(n)})$ è $O(p(n) \log n \log c^{p(n)}) = O((p(n))^2 \log n)$, e quindi che la lunghezza della formula $\phi(x)$ è anch'essa $O((p(n))^2 \log n)$.

Si può verificare (vedi Esercizio 9.19) che la formula $\phi(x)$, oltre ad avere lunghezza polinomiale, può essere costruita in spazio logaritmico, e che esiste un'assegnazione di verità che la soddisfa se e solo se $\mathcal{M}$ accetta la stringa di input x. □

Esercizio 9.19 Dimostrare che la formula $\phi(x)$ nella dimostrazione del Teorema 9.21 può essere costruita in spazio logaritmico, e che esiste una assegnazione di verità che la soddisfa se e solo se la macchina di Turing $\mathcal{M}$ accetta la stringa x.

Dal Teorema 9.21 deriva che per ogni $i \geq 1$ $\Sigma_i^p \subseteq$ PSPACE, in quanto ogni problema Formula Booleana Quantificata i-limitata (per ogni $i \geq 1$) è immediatamente Karp-riducibile in spazio logaritmico a Formula Booleana Quantificata, e le classi Σ_i^p ($i \geq 1$) sono chiuse rispetto a tale riducibilità. Da ciò evidentemente deriva che NP = PSPACE implica il collasso dell'intera gerarchia polinomiale.

Esercizio 9.20 Dimostrare che ogni classe Σ_i^p è chiusa rispetto alla Karp-riducibilità *log*-space.

Non è difficile rendersi conto che la classe PSPACE è chiusa rispetto alla complementazione, vale a dire che PSPACE=co-PSPACE. Ciò risulta evidente applicando ad esempio le stesse considerazioni effettuate per mostrare che P=co-P, vale a dire osservando che per risolvere un problema complementare di un problema $\mathcal{P} \in$ PSPACE basta cambiare l'algoritmo per $\mathcal{P}$ in modo da scambiare risposte VERO con risposte FALSO.

Numerosi problemi interessanti risultano PSPACE-completi: la loro completezza può essere mostrata, come nel caso dei problemi NP-completi considerati in precedenza, mostrando che essi sono in PSPACE e individuando una Karp-riduzione *log*-space da un problema PSPACE-completo ad essi. Tra questi possiamo citare il problema GEOGRAFIA introdotto qui di seguito.

Definiamo il seguente gioco *Geografia* da effettuarsi tra due giocatori A e B su un grafo orientato $G = (V, A)$. I giocatori scelgono alternativamente archi da A, con il giocatore A che sceglie inizialmente un arco uscente dal nodo predefinito v_0: ad ogni turno, un giocatore deve scegliere un arco $\langle V_i, V_j \rangle$, se l'avversario al turno precedente ha scelto un arco $\langle V_k, V_i \rangle$. Il primo giocatore che non ha archi da scegliere perde.

GEOGRAFIA

ISTANZA: Grafo orientato $G = (V, A)$, vertice $v_0 \in V$

PREDICATO: Il giocatore A ha una strategia vincente nel gioco *Geografia* eseguito sul grafo G a partire dal nodo v_0?

Al fine di mostrare la PSPACE-completezza di GEOGRAFIA dimostriamo prima il seguente lemma.

Lemma 9.22 *Per ogni istanza* Φ *di* FORMULA BOOLEANA QUANTIFICATA *esiste una istanza equivalente* $\overline{\Phi}$ *con le seguenti caratteristiche:*

1. *la formula* $\overline{\mathcal{F}}$ *di* $\overline{\Phi}$ *è in CNF;*
2. *il primo quantificatore di* $\overline{\Phi}$ *è un quantificatore esistenziale;*
3. *l'ultimo quantificatore di* $\overline{\Phi}$ *è anch'esso un quantificatore esistenziale;*
4. *i quantificatori si alternano, nel senso che , per ogni* $i < n$, *se* $Q_i = \exists$ *allora* $Q_{i+1} = \forall$ *e se* $Q_i = \forall$ *allora* $Q_{i+1} = \exists$*;*
5. *l'istanza* $\overline{\Phi}$ *ha dimensione polinomiale nella dimensione dell'istanza* Φ*.*

Dimostrazione. Sia $\Phi(x_1, \ldots, x_n) = Q_1 x_1 \ldots Q_n x_n \mathcal{F}(x_1, \ldots, x_n)$ una istanza di FORMULA BOOLEANA QUANTIFICATA. A partire da Φ possiamo costruire una istanza Φ' avente la proprietà che la formula $\mathcal{F}'$ è in CNF nel modo

seguente: introduciamo una nuova variabile z_i, quantificata in modo esistenziale, per rappresentare ogni sottoespressione in $\mathcal{F}$, e definiamo $\mathcal{F}'$ per mezzo della composizione delle variabili z_i (vedi Esempio 9.8).

A partire da

$$\Phi'(z_1, \ldots, z_m, x_1, \ldots, x_n) = Q_1 z_1 \ldots Q_{m+n} x_n \mathcal{F}'(z_1, \ldots, z_m, x_1, \ldots, x_n)$$

costruiamo una istanza Φ'', avente le prime tre caratteristiche nell'enunciato, introducendo opportune variabili *dummy*, che non sono presenti nella formula $\mathcal{F}$, nel modo seguente:

- se $Q_1 = \exists$ e $Q_{m+n} = \exists$ allora $\Phi'' = \Phi'$;
- se $Q_1 = \forall$ e $Q_m = \exists$ allora poniamo

$$\Phi'' = \exists y_0 Q_1 z_1 \ldots Q_{m+n} x_n \mathcal{F}'(z_1, \ldots, z_m, x_1, \ldots, x_n);$$

- se $Q_1 = \exists$ e $Q_m = \forall$ allora poniamo

$$\Phi'' = Q_1 z_1 \ldots Q_{m+n} x_n \exists y_1 \mathcal{F}'(z_1, \ldots, z_m, x_1, \ldots, x_n);$$

- se $Q_1 = \forall$ e $Q_m = \forall$ allora poniamo

$$\Phi'' = \exists y_0 Q_1 z_1 \ldots Q_{m+n} x_n \exists y_1 \mathcal{F}'(z_1, \ldots, z_m, x_1, \ldots, x_n).$$

Al fine di ottenere, a partire da Φ'', l'istanza $\overline{\Phi}$ avente i quantificatori alternanti, possiamo procedere nel modo seguente: per ogni coppia Q_i, Q_{i+1} di quantificatori successivi dello stesso tipo introduciamo ua variabile *dummy* ed un quantificatore di tipo opposto da inserire tra Q_i e Q_{i+1}. Se ad esempio $Q_i = \forall$ e $Q_{i+1} = \forall$, la sequenza di quantificatori in Φ'' sarà $(\ldots Q_i x_i \exists y_i Q_{i+1} x_{i+1} \ldots)$: in questo modo, come si può osservare, $\mathcal{F}'' = \mathcal{F}$ e la sequenza dei quantificatori di Φ'' ha lunghezza al più doppia rispetto a quella di Φ.

È possibile verificare facilmente (vedi Esercizio 9.21) che l'istanza $\overline{\Phi}$ ha dimensione polinomiale nella dimensione di Φ. □

Esempio 9.8 Si consideri l'istanza di FORMULA BOOLEANA QUANTIFICATA rappresentata dalla formula

$$\forall x_1 \forall x_2 \exists x_3 [((x_1 \wedge \overline{x}_3) \vee x_2) \wedge x_3].$$

Definiamo la variabile z_1 e poniamo $\equiv (x_1 \wedge \overline{x}_3)$, ottenendo la formula

$$\exists z_1 \forall x_1 \forall x_2 \exists x_3 [(z_1 \equiv (x_1 \wedge \overline{x}_3)) \wedge (z_1 \vee x_2) \wedge x_3].$$

Definiamo ora la variabile z_2 e poniamo $z_2 \equiv (z_1 \vee x_2)$, ottenendo la formula

$$\exists z_1 \exists z_2 \forall x_1 \forall x_2 \exists x_3 [(z_1 \equiv (x_1 \wedge \overline{x}_3)) \wedge (z_2 \equiv (z_1 \vee x_2)) \wedge z_2 \wedge x_3].$$

Osserviamo ora che la formula $z_1 \equiv (x_1 \wedge \overline{x}_3))$ può essere scritta anche come

$$(\overline{x}_1 \vee x_3 \vee z_1) \wedge (x_1 \vee \overline{z}_1) \wedge (\overline{x}_3 \vee \overline{z}_1),$$

e che $z_2 \equiv (z_1 \vee x_2)$ può a sua volta essere scritta come

$$(\overline{z}_1 \vee \overline{z}_2 \vee x_2) \wedge (z_2 \vee \overline{z}_1) \wedge (\overline{x}_2 \vee z_2).$$

Da ciò otteniamo la formula Φ' definita come:

$$\exists z_1 \exists z_2 \forall x_1 \forall x_2 \exists x_3 [(\overline{x}_1 \vee x_3 \vee z_1) \wedge (x_1 \vee \overline{z}_1) \wedge (\overline{x}_3 \vee \overline{z}_1) \wedge$$
$$\wedge (\overline{z}_1 \vee \overline{z}_2 \vee x_2) \wedge (z_2 \vee \overline{z}_1) \wedge (\overline{x}_2 \vee z_2) \wedge z_2 \wedge x_3].$$

Applicando le altre trasformazioni descritte nella dimostrazione del Lemma 9.22, otteniamo quindi l'istanza $\overline{\Phi}$ voluta

$$\exists z_1 \forall y_1 \exists z_2 \forall x_1 \exists y_2 \forall x_2 \exists x_3 [(\overline{x}_1 \vee x_3 \vee z_1) \wedge (x_1 \vee \overline{z}_1) \wedge (\overline{x}_3 \vee \overline{z}_1) \wedge$$
$$\wedge (\overline{z}_1 \vee \overline{z}_2 \vee x_2) \wedge (z_2 \vee \overline{z}_1) \wedge (\overline{x}_2 \vee z_2) \wedge z_2 \wedge x_3].$$

Esercizio 9.21 Verificare che l'istanza $\overline{\Phi}$ ottenuta applicando il procedimento descritto nella dimostrazione del Lemma 9.22 ha in effetti dimensione polinomiale nella dimensione dell'istanza originaria Φ.

Passiamo ora a dimostrare che GEOGRAFIA è un problema completo in PSPACE.

Teorema 9.23 *Il problema* GEOGRAFIA *è* PSPACE*-completo rispetto alla Karp–riducibilità log-space.*

Dimostrazione. Mostriamo dapprima che GEOGRAFIA $\in$ PSPACE: per verificare che tale relazione è vera osserviamo che ogni partita di *Geografia* ha lunghezza polinomiale. Consideriamo inoltre l'albero di gioco associato ad una istanza di GEOGRAFIA: in tale albero ogni nodo è associato ad una posizione nella partita, con la radice associata alla posizione iniziale, ed ogni arco (u, v) corrisponde ad una mossa che viene effettuata nella posizione associata ad u e porta alla posizione associata a v. Gli archi uscenti dalla radice rappresentano possibili mosse di A, gli archi del livello sottostante possibili mosse di B, e così via. Dato che una partita corrisponde ad un cammino nell'albero, ed ha lunghezza polinomiale, ne deriva che l'altezza dell'albero è polinomiale.

Non è difficile rendersi conto (vedi Esercizio 9.22) che per verificare se il giocatore A ha una strategia vincente è sufficiente effettuare una opportuna visita in profondità dell'intero albero di gioco, e che l'effettuazione di tale visita richiede spazio polinomiale.

Per mostrare la completezza di GEOGRAFIA in PSPACE, dimostriamo ora che FORMULA BOOLEANA QUANTIFICATA $\leq_m^{logsp}$ GEOGRAFIA. A tal fine, assumiamo di avere una istanza di $\Phi = \exists x_1 \forall x_2 \ldots \exists x_n \mathcal{F}(x_1, x_2, \ldots, x_n)$ di FORMULA BOOLEANA QUANTIFICATA nella forma enunciata nel Lemma 9.22, vale a dire tale che:

1. il primo quantificatore Q_1 è un quantificatore esistenziale $\exists$;
2. l'ultimo quantificatore Q_n è anch'esso un quantificatore esistenziale $\exists$;
3. i quantificatori si alternano;
4. la formula non quantificata $\mathcal{F}$ è in CNF.

Per il Lemma 9.22, le ipotesi precedenti non limitano l'applicabilità della dimostrazione.

Sia m il numero di clausole in $\mathcal{F}$. A partire da Φ costruiamo una istanza di GEOGRAFIA nel modo seguente. Associamo ad ogni variabile x_i in Φ un grafo V_i di quattro nodi n_i, s_i, e_i, w_i e quattro archi $< n_i, e_i >$,$< n_i, w_i >$,$< e_i, s_i >$,$< w_i, s_i >$: tali grafi sono composti in una sequenza mediante archi $< s_i, n_{i+1} >$.

Alla j-esima clausola $t_{j,1} \vee t_{j,2} \vee \ldots \vee t_{j,n_j}$ di $\mathcal{F}$ associamo un grafo C_j di n_j+1 nodi $c_j, t_{j,1}, \ldots, t_{j,n_j}$ e n_j archi $< c_j, t_{j,1} >, < c_j, t_{j,2} >, \ldots, < c_j, t_{j,n_j} >$.

Nell'istanza di GEOGRAFIA il nodo s_n è collegato ad un nodo r, dal quale a sua volta esce un arco verso ogni nodo c_j, $1 \leq j \leq m$. Inoltre, se $t_{j,i} = x_k$ esiste un arco $< c_{j,i}, w_k >$, mentre se $t_{j,i} = \overline{x}_k$ esiste un arco $< c_{j,i}, e_k >$.

Sia $n_1 = v_0$ il nodo iniziale del gioco. Per la struttura del grafo risultante, la partita procede nel modo seguente: il giocatore A inizialmente sceglie uno tra i due archi $< n_0, e_1 >$, $< n_0, w_1 >$, scegliendo così un valore VERO (nel primo caso), FALSO (nel secondo caso) per la variabile x_1. La stessa scelta viene quindi effettuata da B per la seconda variabile, e così via, fino alla scelta effettuata da A per la variabile x_n. In tal modo, viene costruita una assegnazione di verità sulle variabili $x_1, \ldots, x_n$.

La scelta dell'arco uscente dal nodo r è effettuata dal giocatore B, il quale in tal modo può selezionare una possibile clausola c_j falsa secondo l'assegnazione costruita. A questo punto, il giocatore A può selezionare un termine $t_{j,i}$ di tale clausola. Per costruzione, se tale termine è vero nell'assegnazione di verità allora il giocatore B non ha archi da scegliere, ed A vince la partita. Al contrario, se il termine selezionato da A non è vero nell'assegnazione di verità, allora B può scegliere l'unico arco uscente dal nodo associato a tale termine, mentre A successivamente non ha possibilità di scelta, per cui B vince la partita.

Da quanto detto, risulta semplice verificare sia che A ha una strategia vincente per l'istanza di GEOGRAFIA se e solo se la formula nell'istanza di Φ è vera, sia che la riduzione mostrata può essere effettuata in spazio logaritmico. □

Esempio 9.9 Consideriamo l'istanza $\exists x_1 \forall x_2 \exists x_3 [(x_1 \vee \overline{x}_2) \wedge (x_1 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3) \vee \overline{x}_2]$.

Applicando la riduzione mostrata nella dimostrazione del Teorema 9.23 si ottiene l'istanza di GEOGRAFIA in Figura9.13.

Esercizio 9.22 Mostrare come sia possibile verificare se il giocatore A ha una strategia vincente in *Geografia* effettuando una visita in profondità del relativo albero di gioco.

Esercizio 9.23 Dimostrare che la riduzione illustrata nella dimostrazione del Teorema 9.23 richiede spazio logaritmico.

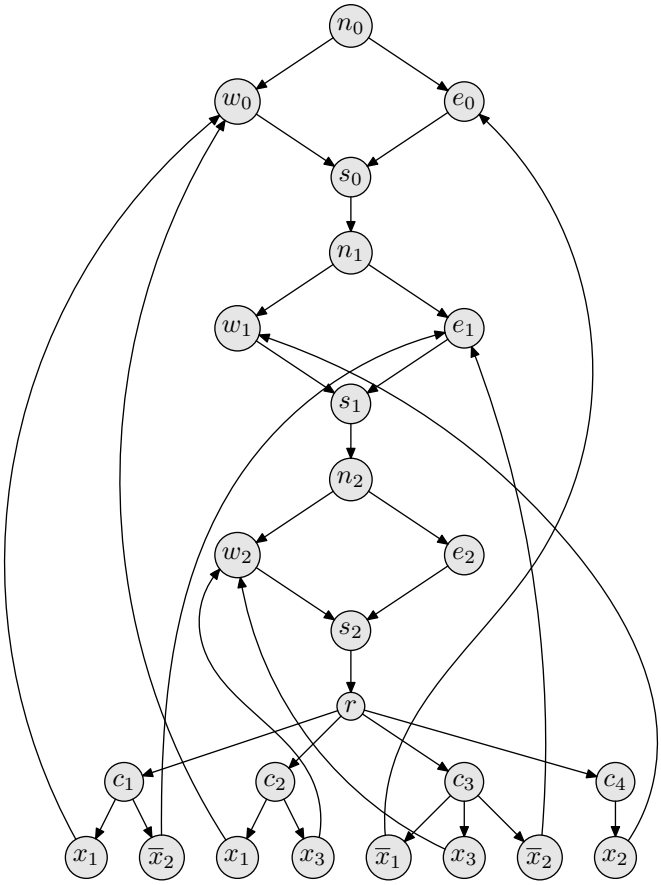

FIGURA 9.13 Istanza di GEOGRAFIA risultante dall'Esempio 9.23

Un ulteriore esempio di problema PSPACE-completo, problema originato nell'ambito della teoria dei linguaggi formali, è il seguente.

COMPLETEZZA DI ESPRESSIONI REGOLARI

ISTANZA: Espressione regolare ρ sugli operatori $+$, $\cdot$, $*$ e su un alfabeto Σ

PREDICATO: Denominato con $L(\rho)$ il linguaggio descritto da ρ, è vero che $L(\rho) = \Sigma^*$?

Teorema 9.24 *Il problema* COMPLETEZZA DI ESPRESSIONI REGOLARI *è* PSPACE*-completo rispetto alla Karp–riducibilità log-space.*

Dimostrazione. La dimostrazione è lasciata come esercizio (vedi Esercizio 9.24). □

Esercizio 9.24 Dimostrare la PSPACE-completezza del problema COMPLETEZZA DI ESPRESSIONI REGOLARI.

Indice analitico