La classe **NP**

Indice

La classe **NP** è già stata introdotta nella Dispensa 6 come la classe dei linguaggi accettati in tempo polinomiale da una macchina di Turing non deterministica. In questa dispensa studieremo la struttura della classe **NP**, ossia, ne individueremo alcune sottoclassi di particolare interesse ai fini della Teoria della Complessità Computazionale: la classe dei linguaggi **NP**-completi, la classe dei linguaggi **NP**-intermedi, e la classe dei linguaggi **NP**-completi in senso forte. La nostra analisi sarà, comunque, specificatamente rivolta ai problemi decisionali, invece che ai linguaggi, secondo quanto discusso nella Dispensa 7.
Prima di analizzarne la struttura, ci chiariremo le idee circa il significato del concetto di non determinismo, e, alla luce di ciò, forniremo di **NP** una definizione alternativa presentando anche esempi di problemi in **NP** e di algoritmi non deterministici.

9.1 Ma cosa è il non determinismo?

I problemi decisionali *trattabili computazionalmente* sono quei problemi che sono decisi da un algoritmo (tradizionalmente inteso come tale) la cui esecuzione su input x richiede tempo polinomiale in $|x|$. Come sappiamo, la classe dei problemi trattabili computazionalmente è denotata con **P**.

La classe **NP** è stata introdotta considerando un classe di algoritmi più potenti di quelli che possono essere eseguiti sui normali calcolatori, e, dunque, un modello di calcolo più potente del modello di Von Neumann: la macchina di Turing non deterministica.
Ricordiamo, infatti, che **NP** è la classe dei linguaggi *accettati* in tempo polinomiale da una macchina di Turing non deterministica; pertanto, se $L \in$ **NP** sappiamo che esistono una macchina di Turing non deterministica NT e un intero $k \in \mathbb{N}$ tali che, per ogni $x \in L$, la computazione $NT(x)$ accetta in $|x|^k$ passi. Se, invece $x \in \Sigma^* - L$ (dove Σ è l'alfabeto sul quale L è definito), sappiamo soltanto che la computazione $NT(x)$ non accetta. In effetti, però, poiché sappiamo che, su input $x \in \Sigma^*$, se $NT(x)$ accetta lo fa in $|x|^k$ passi, se al passo $|x|^k$ la computazione $NT(x)$ non ha ancora raggiunto lo stato di accettazione, possiamo concludere che non lo raggiungerà mai: ossia, possiamo concludere che $x \notin L$. Allora, possiamo derivare da NT una nuova macchina NT' che si comporta come NT ma che termina *sempre*: utilizzando un nasto aggiuntivo come contatore del numero di istruzioni eseguite, con input x esegue esattamente le stesse istruzioni della computazione $NT(x)$ con l'unica differenza che, se la computazione $NT(x)$ arriva ad eseguire l'istruzione numero $|x|^k + 1$, allora $NT'(x)$ entra nello stato di rigetto. In conclusione, sembrerebbe che possiamo affermare che **NP** è la classe dei linguaggi decisi da una macchina di Turing non deterministica. Allora, perché si parla di linguaggi *accettati* invece che decisi?

Per rispondere alla precedente domanda cerchiamo di comprendere più compiutamente in cosa consiste una computazione non deterministica. Osserviamo, intanto, che ciascuna macchina di Turing non deterministica NT ha la possibilità di valutare in parallelo, ad ogni passo, un numero costante p_{NT} di possibilità (ove, ricordiamo, p_{NT} è il grado di non determinismo di NT). Questo significa che, in una computazione di k passi di NT, vengono valutate complessivamente p_{NT}^k possibilità. O che, ancora in altri termini, una computazione di k passi di una macchina di Turing non deterministica NT con grado di non determinismo pari a p_{NT} può essere considerata come una sequenza di k scelte in cui ciascuna scelta opera in un insieme di (al più) p_{NT} possibilità. In effetti, possiamo rappresentare una computazione di NT, su input $x = x_1x_2\ldots x_n$, come un *albero della computazione* i cui nodi interni, di grado al più p_{NT}, rappresentano gli stati globali di $NT(x)$: la radice dell'albero rappresenta lo stato globale iniziale $SG_0 = (q_0,x_1)x_2\ldots x_n$, ed esiste un arco da uno stato globale SG ad uno stato globale SG' se esiste una quintupla di NT che faccia passare dallo stato globale SG allo stato globale SG': dunque, l'arco (SG,SG') dell'albero rappresenta la scelta di eseguire proprio la quintupla che permette di giungere nell' stato globale SG' fra tutte le quintuple che potevano essere eseguite dello stato globale SG.
Naturalmente, non è che la computazione di una macchina non deterministica esamini effettivamente in parallelo tutte le scelte possibili: essa, in ultima analisi, consiste in una singola sequenza di scelte, ovvero, in un percorso nell'albero della computazione che connette la radice ad una foglia. In definitiva, quindi, quello che fa una macchina non deterministica è scegliere, ad ogni passo, la quintupla da eseguire. Lo scopo di una computazione non determistica è quello di verificare se, fra le tante sequenze di scelte possibili, ne *esiste* una "buona", ossia, una che permette di affermare che l'input appartiene al linguaggio o, equivalentemente, che l'istanza è una istanza sì del problema. Un modo alternativo di considerare una computazione non deterministica è quello di assegnare alla macchina il compito di indovinare, ad ogni passo, la scelta *giusta* da fare, ossia, di scegliere la quintupla, se esiste, la cui esecuzione porterà ad uno stato globale da cui si può raggiungere uno stato globale di accettazione. Se non esiste una scelta da giusta da fare, cioè,

Input:	stringa $x_1x_2\ldots x_n$ memorizzata nell'array N, con $N[i] = x_i$ per $i = 1,\ldots,n$.
Costanti:	l'insieme delle quintuple $P = \{\langle q_{1_1}, s_{1_1}, s_{1_2}, q_{1_2}, m_1\rangle, \langle q_{2_1}, s_{2_1}, s_{2_2}, q_{2_2}, m_2\rangle, \ldots, \langle q_{k_1}, s_{k_1}, s_{k_2}, q_{k_2}, m_k\rangle\}$ che descrivono la macchina di Turing NT.

```
1   q ← q_0;
2   t ← 1;
3   primaCella ← t;
4   ultimaCella ← n;
5   while (q ≠ q_A ∧ q ≠ q_R) do begin
6       Ψ ← {⟨q_{i_1}, s_{i_1}, s_{i_2}, q_{i_2}⟩ ∈ P : q_{i_1} = q ∧ s_{i_1} = N[t]};
7       if (Ψ ≠ ∅) then begin
8           scegli ⟨q_{i_1}, s_{i_1}, s_{i_2}, q_{i_2}⟩ ∈ Ψ;
9           N[t] ← s_{i_2};
10          q ← q_{i_2};
11          t ← t + m_i;
12      end
13      if (t < primaCella) then begin
14          primaCella ← t;
15          N[t] ← □;
16      end
17      if (t > ultimaCella) then begin
18          ultimaCella ← t;
19          N[t] ← □;
20      end
21  end
22  Output: q.
```

Tabella 9.1: Algoritmo corrispondente ad una Macchina di Turing non deterministica ad un nastro NT.

se l'esecuzione di nessuna delle quintuple possibili può portare verso uno stato globale di accettazione, allora non ha alcuna importanza quale quintupla la macchina deciderà di eseguire.

Chiarito questo concetto dovrebbe apparire chiara anche la motivazione della definizione di **NP** come della classe dei linguaggi *accettati* (e non *decisi*) in tempo polinomiale da una macchina di Turing non deterministica: nel caso in cui l'input non appartenga al linguaggio (o, equivalentemente, l'istanza non sia una istanza sì del problema) la sequenza delle scelte operate ad ogni passo della computazione dalla macchina di Turing non deterministica è priva di significato.

Sulla base della discussione fin qui svolta, possiamo pensare ad un *algoritmo non deterministico* come ad un algoritmo (scritto, ad esempio, nel linguaggio **PascalMinimo** introdotto nella Dispensa 3) che utilizza la seguente istruzione

$$\textbf{scegli } y \in \Psi(x,t);$$

in cui x è l'input e $\Psi(x,t)$ è l'insieme delle *scelte possibili* ad un dato passo t della computazione relativa all'istanza x del problema. Ricordiamo che, dalla definizione di macchina di Turing non deterministica, la cardinalità dell'insieme $\Psi(x,t)$ deve essere costante, ossia, non dipendente da x e da t; questo significa che, per ogni algoritmo non deterministico $\mathscr{A}$, deve esistere una costante $k_{\mathscr{A}}$ tale che, per ogni possibile input x di $\mathscr{A}$ e per ogni passo t della computazione $\mathscr{A}(x)$, $|\Psi(x,t)| \leq k_{\mathscr{A}}$.
Così, ad esempio, il comportamento di una macchina di Turing non deterministica NT è descritto dall'algoritmo in Tabella 9.1: come si può osservare, esso è l'analogo non deterministico dell'algoritmo in Tabella 3.2 della Dispensa 3 che descrive il comportamento di una macchina deterministica. Si osservi che la cardinalità dell'insieme Ψ calcolato alla linea 6 è al più pari al grado di non determinismo di NT ed è, pertanto, costante.

Input:	un insieme di variabili booleane $X = \{x_1, x_2, \ldots, x_n\}$ e una funzione booleana f definita sulle variabili in X.
1	$i \leftarrow 1$;
2	**while** $(i \leq n)$ **do begin**
3	**scegli** $a(x_i) \in \{\texttt{vero}, \texttt{falso}\}$;
4	$i \leftarrow i+1$;
5	**end**
6	$i \leftarrow 1$;
7	**while** $(i \leq n)$ **do begin**
8	sostituisci in f ogni occorrenza di x_i con $a(x_i)$, e ogni occorrenza di $\neg x_i$ con $\neg a(x_i)$
9	$i \leftarrow i+1$;
10	**end**
11	**if** $(f(a(X)) = \texttt{vero})$ **then** $q \leftarrow q_A$;
12	**else** $q \leftarrow q_R$;
13	**Output:** q.

Tabella 9.2: Algoritmo non deterministico che accetta SAT.

9.2 Problemi in NP

In questo paragrafo cercheremo di comprendere la struttura di un problema appartenente alla classe **NP**, presentando anche due esempi particolarmente significativi.

Esempio 9.1: Abbiamo già incontrato, nella dispensa precedente, il problema SODDISFACIBILITÀ (in breve, SAT): data una funzione booleana $f : \{\texttt{vero}, \texttt{falso}\}^n \rightarrow \{\texttt{vero}, \texttt{falso}\}$ (in forma congiuntiva normale), decidere se esiste una assegnazione di verità a alle sue n variabili che soddisfa f, ossia, tale che, indicato con $X = \{x_1, \ldots, x_n\}$ l'insieme delle sue variabili, $f(a(X)) = \texttt{vero}$.

L'algoritmo che, data una funzione booleana f in n variabili, accetta f se $f \in$ SAT è illustrato in Tabella 9.2. Come si può osservare, esso è logicamente suddiviso in due parti: la prima, prettamente non deterministica, in cui viene *scelta* l'assegnazione di verità a per le variabili in f (linee 1-5), e la seconda, in cui viene verificato deterministicamente che l'assegnazione scelta soddisfi effettivamente f (linee 6-12). Poiché il numero di possibilità fra le quali scegliere ad ogni passo è pari a 2 e poiché l'algoritmo accetta se *esiste* una sequenza di scelte che soddisfa f, si tratta, effettivamente, di un algoritmo non deterministico.

Per quanto riguarda la sua complessità, osserviamo che il primo loop **while** (linee 2-5) richiede tempo non deterministico lineare in $n = |X|$. Inoltre, il secondo loop **while** (linee 7-12) richiede tempo deterministico lineare in $|f|$. In conclusione, l'algoritmo in Tabella 9.2 accetta $f \in$ SAT in tempo $\mathbf{O}(|X| + |f|)$, e questo prova che SAT $\in$ **NP**.

Esempio 9.2: Il problema CLIQUE consiste nel decidere se, dati un grafo non orientato $G = (V, E)$ ed un intero $k \in \mathbb{N}$, G contiene un sottografo completo (ossia, in cui esiste un arco per ogni coppia di nodi) di almeno k nodi. Formalmente, il problema è descritto dalla tripla

- $I_{\text{CLIQUE}} = \{\langle G = (V,E), k\rangle : G \text{ è un grafo non orientato e } k \in \mathbb{N}\}$,
- $S_{\text{CLIQUE}}(G,k) = \{V' \subseteq V\}$,
- $\pi_{\text{CLIQUE}}(G,k,S_{\text{CLIQUE}}(G,k)) = \exists V' \in S_{\text{CLIQUE}}(G,k) : \ \forall u,v \in V' \ [\ (u,v) \in E\ ] \ \wedge \ |V'| \geq k$.

L'algoritmo che, dati un grafo orientato $G = (V,E)$ ed un intero k, accetta $\langle G,k\rangle$ se $\langle G,k\rangle \in$ CLIQUE è illustrato in Tabella 9.3. Come si può osservare, esattamente come per l'algoritmo dell'Esempio 9.1, esso è logicamente suddiviso in due parti: la prima, non deterministica, *sceglie* un sottoinsieme V' di V (linee 2-6), e la seconda (linee 7-18), verifica deterministicamente che il sottoinsieme scelto soddisfi, effettivamente, il predicato

$$\eta_{\text{CLIQUE}}(G,k,S_{\text{CLIQUE}}(G,k)) = \forall u,v \in V' \ [\ (u,v) \in E\ ] \ \wedge \ |V'| \geq k.$$

Poiché il numero di scelte in cui operare ad ogni passo è pari a 2 e poiché l'algoritmo accetta se *esiste* una sequenza di scelte che soddisfa η_{CLIQUE}, si tratta, effettivamente, di un algoritmo non deterministico.

Input:	un grafo non orientato $G=(V,E)$, con $V=\{v_1,v_2,\ldots,v_n\}$ e un intero $k\in\mathbb{N}$.
1	$V'\leftarrow\emptyset$;
2	$i\leftarrow 1$;
3	**while** $(i\leq n)$ **do begin**
4	**scegli** se inserire v_i in V';
5	$i\leftarrow i+1$;
6	**end**
7	$trovata\leftarrow$ `vero`;
8	$i\leftarrow 1$;
9	**while** $(i\leq n-1)$ **do begin**
10	$j\leftarrow i+1$;
11	**while** $(j\leq n)$ **do begin**
12	**if** $(v_i\in V'\ \wedge\ v_j\in V'\ \wedge(v_i,v_j)\notin E)$ **then**
13	$trovata\leftarrow$ `falso`;
14	$j\leftarrow j+1$;
15	**end**
16	$i\leftarrow i+1$;
17	**end**
18	**if** $(trovata=$ `vero` $\wedge\ |V'|\geq k)$ **then** $q\leftarrow q_A$;
19	**else** $q\leftarrow q_R$;
20	**Output:** q.

Tabella 9.3: Algoritmo non deterministico che accetta CLIQUE.

Per quanto riguarda la sua complessità, osserviamo che il primo loop **while** (linee 3-6) richiede tempo non deterministico lineare in $n=|V|$. Inoltre, il secondo loop **while** (linee 8-16) richiede tempo deterministico quadratico in n. In conclusione, l'algoritmo in Tabella 9.3 accetta $\langle G=(V,E),k\rangle\in$ CLIQUE in tempo $\mathbf{O}(n^2)$, e questo prova che CLIQUE $\in$ **NP**.

Esempio 9.3: Il problema LONG PATH (in breve, LP) consiste nel decidere, dati un grafo non orientato $G=(V,E)$, una coppia di suoi nodi $a,b\in V$ ed un intero $k\in\mathbb{N}$, se G contiene un percorso semplice (ossia, senza ripetizione di nodi) da a a b lungo almeno k archi. Formalmente, il problema è descritto dalla tripla

- $I_{\mathrm{LP}}=\{\langle G=(V,E),a,bk\rangle : G$ è un grafo non orientato, $a,b\in V$ e $k\in\mathbb{N}\}$,
- $S_{\mathrm{LP}}(G,a,b,k)=\{p=\langle p_0,p_1,\ldots,p_h\rangle : \{p_0,p_1,\ldots,p_h\}\subseteq V\}$,
- $\pi_{\mathrm{LP}}(G,k,S_{\mathrm{LP}}(G,k))=\exists p\in S_{\mathrm{LP}}(G,k):\ h\geq k\ \wedge\ p_0=a\ \wedge p_h=b$
  $\wedge\forall\, 0\leq i\leq h-1\ \big[\ (p_i,p_{i+1})\in E\ \big]\ \wedge\ \forall\, 0\leq i,j\leq h\ \big[\ i\neq j\rightarrow p_i\neq p_j\ \big]$.

L'algoritmo che, dati un grafo orientato $G=(V,E)$, due suoi nodi a e b ed un intero k, accetta $\langle G,a,b,k\rangle$ se $\langle G,a,b,k\rangle\in$ LP è illustrato in Tabella 9.4. Esattamente come per gli algoritmi degli Esempi precedenti, esso è logicamente suddiviso in due parti: la prima, non deterministica, *sceglie* un sottoinsieme p di V (linee 1-11), e la seconda (linee 7-18), verifica deterministicamente che il sottoinsieme scelto soddisfi, effettivamente, il predicato

$$\eta_{\mathrm{LP}}(G,a,b,k,S_{\mathrm{LP}}(G,a,b,k))=h\geq k\ \wedge\ p_0=a\ \wedge p_h=b$$
$$\wedge\forall\, 0\leq i\leq h-1\ \big[\ (p_i,p_{i+1})\in E\ \big]\ \wedge\ \forall\, 0\leq i,j\leq h\ \big[\ i\neq j\rightarrow p_i\neq p_j\ \big].$$

In particolare, la sequenza p di nodi è costruita in modo che il suo primo nodo sia a (linea 1) e il suo ultimo nodo sia b (linea 13). Poi, viene verificato che la sua lunghezza h sia almeno k (linee 14-15), viene verificato che essa sia, effettivamente, un percorso, ossia, esista sempre l'arco fra ogni coppia di nodi adiacenti nella sequenza (loop *while* alle linee 16-20), e, infine, che essa sia un percorso semplice, ossia, sia priva di ripetizioni di nodi (loop *while* alle linee 21-29).
Si osservi che la generazione non deterministica della sequenza p di nodi è realizzata in modo tale che potrebbe

Input:	un grafo non orientato $G = (V,E)$, con $V = \{v_1, v_2, \ldots, v_n\}$, $a, b \in V$ e un intero $k \in \mathbb{N}$.
1	$p[0] \leftarrow a$;
2	$i \leftarrow 1$;
3	**while** $(p[i-1] \neq \texttt{NULL} \wedge i \leq n-2)$ **do begin**
4	$j \leftarrow 1$;
5	$p[i] \leftarrow \texttt{NULL}$;
6	**while** $(j \leq n \wedge p[j] = \texttt{NULL})$ **do begin**
7	**scegli** se $p[i] = v_j$;
8	$j \leftarrow j+1$;
9	**end**
10	$i \leftarrow i+1$;
11	**end**
12	$h \leftarrow i$;
13	$p[h] \leftarrow b$;
14	**if** $(h < k)$ **then** $trovato \leftarrow \texttt{falso}$;
15	**else** $trovato \leftarrow \texttt{vero}$;
16	$i \leftarrow 0$;
17	**while** $(i \leq h-1 \wedge trovato = \texttt{vero})$ **do begin**
18	**if** $((p[i], p[i+1]) \notin E)$ **then** $trovato \leftarrow \texttt{falso}$;
19	$i \leftarrow i+1$;
20	**end**
21	$i \leftarrow 0$;
22	**while** $(trovato = \texttt{vero} \wedge i \leq h-1)$ **do begin**
23	$j \leftarrow i+1$;
24	**while** $(trovato \leftarrow \texttt{vero} \wedge j \leq h)$ **do begin**
25	**if** $(p[i] = p[j])$ **then** $trovato \leftarrow \texttt{falso}$;
26	$j \leftarrow j+1$;
27	**end**
28	$i \leftarrow i+1$;
29	**end**
30	**if** $(trovato = \texttt{vero})$ **then** $q \leftarrow q_A$;
31	**else** $q \leftarrow q_R$;
32	**Output:** q.

Tabella 9.4: Algoritmo non deterministico che accetta LONG PATH.

accadere che, per qualche indice i, nessun nodo v_j venga scelto come i-esimo nodo di p: non appena questo si verifica, la generazione della sequenza termina e viene posto $p[i] = b$ (linea 13).
Poiché il numero di scelte in cui operare ad ogni passo è pari a 2 e poiché l'algoritmo accetta se *esiste* una sequenza di scelte che soddisfa η_{LP}, si tratta, effettivamente, di un algoritmo non deterministico.

Per quanto riguarda la sua complessità, osserviamo che il primo loop **while** (linee 2-11) richiede tempo non deterministico in $\mathbf{O}(n^2) = \mathbf{O}(= |V|^2)$: in effetti, poiché siamo alla ricerca di un percorso semplice in un grafo, p non può avere lunghezza superiore a $|V|$ e, quindi, il loop **while** alle linee 3-11 esegue al più $\mathbf{O}(n)$ iterazioni. Inoltre, il secondo loop **while** (linee 16-20) richiede tempo deterministico lineare in n e, infine, l'ultimo loop (linee 21-29) richiede tempo deterministico lquadratico in n. In conclusione, l'algoritmo in Tabella 9.4 accetta $\langle G = (V,E), a, b, k\rangle \in$ LP in tempo $\mathbf{O}(n^2)$, e questo prova che LP $\in$ **NP**.

Nei tre esempi che abbiamo presentato in questo paragrafo vengono affrontate questioni che si incontrano frequentemente nel progetto di algoritmi non deterministici: generazione delle assegnazioni di verità ad un insieme di variabili booleane (Esempio 9.1, generazione dei sottoinsiemi di un insieme (Esempio 9.2), generazione delle sequenze di elementi di un insieme (Esempio 9.3). In effetti, gli algoritmi proposti si prestano ad essere generallizzati ad altri contesti, come proveremo ad esemplificare in quanto segue.

- Sia $k \in \mathbb{N}$ un valore costante. Un problema decisionale Γ le cui istanze consistano di un insieme X, in cui l'insie-

me delle soluzioni possibili è l'insieme delle funzioni $a : X \to \{c_1, \ldots, c_k\}$ e il cui predicato sia, informalmente,

$$\pi_\Gamma(X, S_\Gamma(X)) = \exists a \in S_\Gamma(X) : \eta_\Gamma(X, a)$$

può essere accettato da un algoritmo il cui schema ricalca quello dell'algoritmo presentato in Tabella 9.2 in cui l'istruzione alla linea 3 diventa

scegli $a(x_i) \in \{c_1, \ldots, c_k\}$

e le istruzioni alle linee 6-11 valutano il predicato η_Γ.
Poiché k è un valore costante, l'algoritmo risultante corrisponde ad una macchina di Turing non deterministica con grado di non determinismo pari a k. Inoltre, se decidere η_Γ è un problema in **P**, tale algoritmo dimostra che $\Gamma \in$ **NP**.

- Sia Γ un problema decisionale le cui istanze consistano di un insieme V, in cui l'insieme delle soluzioni possibili sia l'insieme dei sottoinsiemi V' di V e il cui predicato sia nella forma:

  $$\pi_\Gamma(V, S_\Gamma(V)) = \exists V' \in S_\Gamma(V) : \eta_\Gamma(V, V')$$

  può essere accettato da un algoritmo il cui schema ricalca quello dell'algoritmo presentato in Tabella 9.3 in cui le istruzioni alle linee 7-19 valutano il predicato η_Γ.
  Di nuovo, se decidere η_Γ è un problema in **P**, tale algoritmo dimostra che $\Gamma \in$ **NP**.

- Sia Γ un problema decisionale le cui istanze consistano di un insieme V, in cui l'insieme delle soluzioni possibili sia l'insieme delle sequenze p di elementi di V di lunghezza limitata superiormente da qualche valore n e il cui predicato sia nella forma:

  $$\pi_\Gamma(V, n, S_\Gamma(V, n)) = \exists p \in S_\Gamma(V, n) : \eta_\Gamma(V, n, p)$$

  può essere accettato da un algoritmo il cui schema ricalca quello dell'algoritmo presentato in Tabella 9.4 in cui le istruzioni alle linee 1 e 12-31 valutano il predicato η_Γ.
  Ancora, se decidere η_Γ è un problema in **P**, tale algoritmo dimostra che $\Gamma \in$ **NP**.

Osserviamo che tutti gli esempi presentati in questo paragrafo mostrano di avere le seguenti caratteristiche comuni:

1) indicato genericamente con S l'insieme delle soluzioni possibili per uno dei problemi decisionali che abbiamo incontrato, il suo predicato π è nella forma

   $$\exists y \in S(x) : \ \eta(x, y),$$

   ove x è una istanza del problema;

2) per ogni istanza x del problema, la costruzione di un elemento $y \in S(x)$ richiede tempo non deterministico polinomiale in $|x|$;

3) la verifica del predicato $\eta(x, y)$ richiede tempo deterministico polinomiale in $|x|$.

Ogni volta che un problema soddisfa le tre caratteristiche appena elencate, possiamo pensare ad un algoritmo che lo decide suddiviso in due fasi: una fase non deterministica, in cui viene costruito (non deterministicamente, appunto) un elemento dell'insieme delle soluzioni possibili, seguita da una fase deterministica, durante la quale viene verificato se la soluzione costruita durante la prima fase soddisfa il predicato η.

Possiamo, allora, affermare che: *se un problema decisionale soddisfa le tre caratteristiche sopra elencate, e se la costruzione di una sua soluzione possibile richiede tempo non deterministico polinomiale, allora esso è contenuto in* **NP**.
Nel prossimo paragrafo approfondiremo questo concetto.

9.3 Una definizione alternativa della classe NP

L'osservazione con la quale abbiamo concluso il precedente paragrafo ci indica, intuitivamente, una condizione sufficiente affinché un problema decisionale appartenga ad **NP**. Il prossimo teorema formalizza questo concetto, dimostrando che, in effetti, tale condizione è necessaria e sufficiente. Esso costituisce, pertanto, una definizione alternativa della classe **NP**.

Teorema 9.1: *Un linguaggio $L \subseteq \Sigma^*$ è in **NP** se e soltanto se esistono una macchina di Turing deterministica T che opera in tempo polinomiale e una costante $k \in \mathbb{N}$ tali che, per ogni $x \in \Sigma^*$,*

$$x \in L \Leftrightarrow \exists y_x \in \{0,1\}^* : \ |y_x| \leq |x|^k \ \wedge T(x, y_x) \text{ accetta.}$$

Dimostrazione: Mostriamo, inizialmente, che se $L \in$ **NP** allora esistono una macchina di Turing deterministica T che opera in tempo polinomiale e una costante $k \in \mathbb{N}$ tali che, per ogni $x \in \Sigma^*$, $x \in L$ se e soltanto se esiste $y_x \in \{0,1\}^*$ tale che $|y_x| \leq |x|^k$ e $T(x, y_x)$ accetta.
Sia, allora, $L \subseteq \Sigma^*$ un linguaggio in **NP**: per definizione di **NP**, esistono una macchina di Turing non deterministica NT e un intero $h \in \mathbb{N}$ tali che NT accetta L e, per ogni $x \in L$, $ntime(NT, x) \leq |x|^h$. Questo significa che, per ogni $x \in L$, esiste una computazione deterministica di $NT(x)$ che termina nello stato di accettazione, ossia, esiste una sequenza di quintuple $p_1, p_2, \ldots, p_{|x|^k}$ che, eseguite a partire dallo stato globale di NT in cui l'input $x = x_1 x_2 \ldots x_n$ è scritto sul nastro, lo stato interno è lo stato iniziale q_0 di NT e la testina è posizionata sulla cella contenente x_1, porta ad uno stato globale di accettazione. Allora, se indichiamo con $p_i = \langle q_{i_1}, s_{i_1}, s_{i_2}, q_{i_2}, m_i \rangle$ la i-esima quintupla della sequenza, per ogni $i = 1, \ldots, n^k$, deve essere $q_{1_1} = q_0$, $q_{(n^k)_2} = q_A$ e, per ogni $2 \leq i \leq n^k$, $q_{i_1} = q_{(i-1)_2}$. Poniamo, allora,

$$y(x) = q_{1_1}, s_{1_1}, s_{1_2}, q_{1_2}, m_1 - q_{2_1}, s_{2_1}, s_{2_2}, q_{2_2}, m_2 - \ldots - q_{(n^k)_1}, s_{(n^k)_1}, s_{(n^k)_2}, q_{(n^k)_2}, m_{(n^k)} \tag{9.1}$$

ossia, $y(x)$ è la parola nell'alfabeto $\Sigma \cup Q \cup \{-, s, f, d\}$ ottenuta concatenando le parole che corrispondono alla sequenza accettante di quintuple.
Sulla base della precedente osservazione, definiamo, ora, una macchina di Turing deterministica a due nastri (a testine indipendenti) $\overline{T}$ che corrisponde alla macchina NT: essa possiede, codificata nelle sue quintuple, la descrizione dell'insieme P delle quintuple di NT. La computazione $\overline{T}(x, y)$, con input $x \in \Sigma^*$ scritto sul primo nastro e $y \in \left(\Sigma \cup Q \cup \{-, s, f, d\}\right)^*$ scritto sul secondo nastro, procede come di seguito descritto.

1) $\overline{T}$ verifica che y sia nella forma $q_{1_1}, s_{1_1}, s_{1_2}, q_{1_2}, m_1 - q_{2_1}, s_{2_1}, s_{2_2}, q_{2_2}, m_2 - \ldots - q_{(n^k)_1}, s_{(n^k)_1}, s_{(n^k)_2}, q_{(n^k)_2}, m_{(n^k)}$: se così non è, rigetta.

2) $\overline{T}$ verifica che , per ogni $1 \leq i \leq n^k$, $\langle q_{i_1}, s_{i_1}, s_{i_2}, q_{i_2}, m_i \rangle \in P$: se così non è, rigetta.

3) $\overline{T}$ verifica che $q_{1_1} = q_0$ e $q_{(n^k)_2} = q_A$: se così non è, rigetta.

4) $\overline{T}$ verifica che, per ogni $2 \leq i \leq n^k$, $q_{i_1} = q_{(i-1)_2}$: se così non è, rigetta.

5) $\overline{T}$ simula la computazione di $NT(x)$ descritta da y:

   5.1) con la testina posizionata sulla cella contenente x_1, verifica se la quintupla $\langle q_{1_1}, s_{1_1}, s_{1_2}, q_{1_2}, m_1 \rangle$ può essere eseguita, ossia, se $s_{1_1} = x_1$ e, se è così, eseguila modificando (eventualmente) il contenuto della cella del primo nastro su cui è posizionata la testina e spostando (eventualmente) la testina sul primo nastro, altrimenti rigetta;

   5.2) per ogni $2 \leq i \leq n^k$, verifica se la quintupla $\langle q_{i_1}, s_{i_1}, s_{i_2}, q_{i_2}, m_i \rangle$ può essere eseguita, ossia, se il simbolo letto dalla testina è s_{i_1} e, se è così, la esegue modificando (eventualmente) il contenuto della cella del primo nastro su cui è posizionata la testina e spostando (eventualmente) la testina sul primo nastro, altrimenti rigetta.

6) $\overline{T}$ accetta.

Mostriamo ora che $x \in L$ se e soltanto se esiste $y \in \left(\Sigma \cup Q \cup \{-, s, f, d\}\right)^*$ tale che $\overline{T}(x, y)$ accetta. Sia $x \in L$. Allora per quanto precedentemente osservato, esiste una parola $y(x) \in \left[\Sigma \cup Q \cup \{-, s, f, d\}\right]^*$ che corrisponde alla codifica di una computazione accettante di $NT(x)$ e quindi, per costruzione, $\overline{T}(x, y(x))$ accetta. Se, invece, $x \notin L$, non esiste alcuna computazione accettante di $NT(x)$. Allora, qualunque sia $y(x) \in \left[\Sigma \cup Q \cup \{-, s, f, d\}\right]^*$, non può accadere che $\overline{T}(x, y(x))$ accetti: infatti, o $y(x)$ non codifica una possibile computazione accettante di NT (e, quindi, $\overline{T}(x, y(x))$ rigetta al passo 1) o al passo 2) o al passo 3) o al passo 4)), oppure $y(x)$ codifica una possibile computazione accettante di NT che, però, non può essere eseguita sull'input x (e, quindi, $\overline{T}(x, y(x))$ rigetta al passo 5.1) oppure a qualche iterazione del passo 5.2)). Dunque, $x \in L$ se e solo se esiste $y(x) \in \left[\Sigma \cup Q \cup \{-, s, f, d\}\right]^*$ tale che $\overline{T}(x, y(x))$ accetta.
È immediato verificare che $\overline{T}$ opera in tempo polinomiale in $|x|$ e $|y(x)|$. Osserviamo, inoltre, che, se $x \in L$, allora $y(x)$ è la codifica di una computazione accettante di $NT(x)$ che è costituita di al più $|x|^k$ passi. Quindi, se $x \in L$, $|y(x)| \in \mathbf{O}(|x|^k)$ e, di conseguenza, $\overline{T}$ opera in tempo polinomiale in $|x|$.
Infine, è possibile trasformare $\overline{T}$ in una macchina T il cui alfabeto di lavoro sia $\{0, 1\}$, secondo quanto descritto nel Paragrafo 2.5 della Dispensa 5. Ricordando che le macchine di Turing deterministiche che operano sull'alfabeto $\{0, 1\}^*$ sono polinomialmente correlate alle macchine di Turing che operano su un generico alfabeto Σ con $|\Sigma| > 2$ (Teorema 6.3 della Dispensa 6), questo completa la prima parte della dimostrazione, ossia, che, se $L \in \mathbf{NP}$, allora esistono una macchina di Turing deterministica T che opera in tempo polinomiale e una costante $k \in \mathbb{N}$ tali che, per ogni $x \in \Sigma^*$, $x \in L$ se e soltanto se esiste $y_x \in \{0, 1\}^*$ tale che $|y_x| \leq |x|^k$ e $T(x, y_x)$ accetta, ove y_x è la codifica nell'alfabeto $\{0, 1\}$ utilizzata da T della parola $y(x)$.

Viceversa, sia $L \subseteq \Sigma^*$ un linguaggio per il quale esistono una macchina di Turing deterministica T che opera in tempo polinomiale e una costante $k \in \mathbb{N}$ tali che, per ogni $x \in \Sigma^*$,

$$x \in L \Leftrightarrow \exists y_x \in \{0, 1\}^* : \ |y_x| \leq |x|^k \ \wedge T(x, y_x) \text{ accetta.}$$

Senza perdita di generalità, assumiamo che T disponga di un solo nastro sul quale, inizialmente, sono scritte le due parole x e y separate da un carattere '□'. Definiamo la seguente macchina di Turing non deterministica NT che opera in due fasi: con input x

- durante la prima fase genera una parola $y \in \{0, 1\}^*$ di lunghezza al più $|x|^k$ scrivendola sul nastro, alla destra di x e separato da esso da un carattere '□',
- durante la seconda fase simula la computazione $T(x, y)$.

Si osservi, innanzi tutto, che NT opera in tempo polinomiale in $|x|$: infatti, la prima fase, in cui viene generata y, richiede al più $\mathbf{O}(|x|^k)$ passi e la seconda fase richiede tempo proporzionale a $dtime(T, (x, y))$ e quindi, poiché $dtime(T, (x, y))$ è un polinomio in $|x|$ e $|y|$ e $|y| \leq |x|^k$, polinomiale in $|x|$.

Sia $x \in \Sigma^*$. Se $x \in L$, allora, per ipotesi, esiste una parola $y_x \in \{0, 1\}^*$ di lunghezza $|y_x| \leq |x|^k$ tale che $T(x, y_x)$ accetta. Allora, durante la prima fase della computazione di $NT(x)$ esiste una sequenza di scelte che genera y_x che, nella seconda fase, induce NT ad accettare. Quindi, NT accetta x. Di contro, se NT accetta x allora esiste una parola $y_x \in \{0, 1\}^*$ di lunghezza $|y_x| \leq |x|^k$, generata durante la prima fase, che induce la seconda fase ad accettare: ma, poiché la seconda fase di NT non è altro che una simulazione di T, questo significa $T(x, y_x)$ accetta. Quindi, $x \in L$.
In conclusione, $x \in L$ se e soltanto se $NT(x)$ accetta, e, poiché NT è una macchina di Turing non deterministica che opera in tempo polinomiale, questo prova che $L \in \mathbf{NP}$. □

Cerchiamo, ora, di comprendere a fondo il significato del Teorema 9.1.
La parola y_x che compare nell'enunciato del Teorema 9.1 può essere considerata una sorta di *prova* che x sia davvero contenuto in L, una prova che deve essere successivamente verificata. Ad essa ci riferisce generalmente come ad un *certificato* in quanto certifica l'appartenenza di una parola ad un linguaggio. Ad esempio, se ci chiediamo se una data formula booleana f è soddisfacibile, un ovvio certificato è una assegnazione di verità alle variabili che compaiono in f: se la successiva verifica mostra che f assume il valore `vero` sotto quella assegnazione, siamo certi che f sia soddisfacibile. Osserviamo, ora, che ogni problema Γ in **NP** che abbiamo descritto nel Paragrafo 9.2 poteva essere formalizzato nel seguente modo:

- un insieme di istanze I_Γ,

- per ogni $x \in I_\Gamma$, un insieme di soluzioni possibili $S_\Gamma(x)$ generabili non deterministicamente in tempo polinomiale,
- per ogni istanza $x \in I_\Gamma$, un predicato $\pi_\Gamma(x, S_\Gamma(x))$ nella forma

$$\pi_\Gamma(x, S_\Gamma(x)) = \exists y \in S_\Gamma(x) : \ \eta_\Gamma(x, y)$$

tale che $\eta_\Gamma(x,y)$ è decidibile in tempo polinomiale in $|x|$.

In altri termini, ognuno dei problemi che abbiamo incontrato ammetteva l'esistenza di una soluzione possibile, ossia, di un certificato, che si potesse verificare essere una soluzione effettiva in tempo polinomiale nella dimensione dell'istanza. Il Teorema 9.1 mostra che, in effetti, un problema è in **NP** se e soltanto se esso può essere formalizzato in questo modo, ossia: *un problema è in **NP** se e soltanto se le parole che esso contiene ammettono certificati verificabili in tempo polinomiale nella loro dimensione*.

9.4 Problemi NP-completi: il teorema di Cook-Levin

Cominciamo, in questo paragrafo, a studiare la struttura della classe **NP** mostrando che contiene una sottoclasse non vuota, la sottoclasse **NPC** dei problemi (o, equivalentemente, dei linguaggi) **NP**-completi. Già conosciamo, dalla Dispensa 6, i linguaggi **NP**-completi e sappiamo che essi sono, in un certo senso, i linguaggi più "difficili" in **NP**. Infatti, ricordiamo che un problema Γ è **NP**-completo se

1) $\Gamma \in$ **NP** e
2) ogni altro linguaggio $\Gamma' \in$ **NP** è riducibile a Γ.

Quindi, se un linguaggio Γ è **NP**-completo, la soluzione di ogni altro linguaggio in **NP** è riconducibile alla soluzione di Γ. In effetti, come abbiamo dimostato nel Corollario 6.3, che richiamiamo,

*se un problema **NP**-completo è contenuto in **P** allora **P** = **NP**.*

Alla luce del Corollario 6.3, dunque, i problemi **NP**-completi sono i candidati ad essere problemi separatori fra **P** e **NP**, cioè, *se* esiste un problema **NP**-completo è poco probabile che esso sia contenuto in **P**. Ma esiste qualche problema **NP**-completo?
Il Teorema di Cook-Levin risponde positivamente alla precedente domanda. Esso si basa sull'osservazione che le computazioni di qualunque macchina di Turing possono essere descritte da formule booleane.

Sia NT una macchina di Turing non deterministica ad un nastro definita sull'alfabeto Σ e sull'insieme degli stati Q, e sia $P(q,\sigma)$ l'insieme delle quintuple di NT i cui primi due elementi siano $q \in Q$ e $\sigma \in \Sigma$. Il significato dell'insieme $P(\sigma,q)$ è : ad ogni istante t della computazione, *se* la macchina si trova nello stato q *e* la testina sta scandendo una cella che contiene il carattere σ *allora* la macchina scrive σ' nella cella che sta scandendo, entra nello stato q' e sposta la testina in accordo ad m per qualche $\sigma' \in \Sigma$, $q' \in Q$, $m \in \{-1,0,+1\}$ tali che $\langle q, \sigma, \sigma', q', m\rangle \in P(q,s)$.
Allo scopo di esprimere in modo più formale ciò che abbiamo appena descritto, introduciamo un insieme di variabili booleane che, globalmente, rappresentino ogni stato globale possibile di una computazione di NT, assumendo che $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_s\}$, $\sigma_s = \square$ e $Q = \{q_0, q_1, \ldots, q_m\}$, con stato iniziale q_0, stato di accettazione q_1 e stato di rigetto q_2. Per semplicità, supponiamo che NT sia dotata di un nastro semi-infinito, ossia, che il suo nastro non abbia celle di indice minore di 0: è lasciato come semplice esercizio mostrare l'equivalenza del modello a nastro infinito con il modello a nastro semi-infinito. Introduciamo, dunque, per ogni istante di tempo $t \geq 0$ in cui una data computazione di NT è attiva, il seguente insieme di variabili:

- per $0 \leq i \leq t$ e per $1 \leq j \leq s$, $N^t_{i,j}$ = al tempo t la cella i del nastro contiene il carattere σ_j;
- per $0 \leq i \leq t$, R^t_i = al tempo t la testina è posizionata sulla cella i del nastro;
- per $0 \leq j \leq m$, M^t_j = al tempo t la macchina si trova nello stato interno q_j.

Mostriamo, ora, come associare all'insieme $P(q,\sigma)$ un predicato in forma congiuntiva normale. Prima di formalizzare il concetto, procediamo con un paio di esempi e, al fine di semplificare la notazione, siano $q = q_3$ e $\sigma = \sigma_1$.
Se $P(q_3,\sigma_1) = \{\langle q_3,\sigma_1,\sigma_2,q_4,+1\rangle\}$, ossia, se esiste una sola quintupla i cui primi due elementi sono q_3 e σ_1, allora l'insieme $P(q_3,\sigma_1)$ viene rappresentato dal predicato:

$$\forall\, t \geq 0\ \ \forall\, 0 \leq i \leq t\ \left[\ M_3^t\ \wedge\ R_i^t\ \wedge\ N_{i,1}^t\ \rightarrow\ N_{i,2}^{t+1}\ \wedge\ M_4^{t+1}\ \wedge\ R_{i+1}^{t+1}\ \right]$$

che può essere anche scritto come

$$\forall\, t \geq 0\ \ \forall\, 0 \leq i \leq t\ \left[\ \neg\big(\ M_3^t\ \wedge\ R_i^t\ \wedge\ N_{i,1}^t\ \big)\ \vee\ \big(\ N_{i,2}^{t+1}\ \wedge\ M_4^{t+1}\ \wedge\ R_{i+1}^{t+1}\ \big)\right]$$

ovvero, per ogni $t \geq 0$ e per ogni $0 \leq i \leq t$,

$$\big(\neg M_3^t\ \vee\ \neg R_i^t\ \vee\ \neg N_{i,1}^t\ \vee\ N_{i,2}^{t+1}\ \big) \wedge \big(\neg M_3^t\ \vee\ \neg R_i^t\ \vee\ \neg N_{i,1}^t\ \vee\ M_4^{t+1}\ \big) \wedge \big(\neg M_8^t\ \vee\ \neg R_i^t\ \vee\ \neg N_{i,1}^t\ \vee\ R_{i+1}^{t+1}\ \big), \quad (9.2)$$

che è un predicato in forma congiuntiva normale.
Se, invece, $|P(q_3,\sigma_1)| > 1$, un ragionamento analogo al precedente permette di derivare da esso un predicato in forma congiuntiva normale. Ad esempio, se

$$P(q_3,\sigma_1) = \{\langle q_3,\sigma_1,\sigma_2,q_4,+1\rangle, \langle q_3,\sigma_1,\sigma_5,q_6,-1\rangle\},$$

allora ad esso corrisponderà il predicato

$$\forall\, t \geq 0\ \ \forall\, 0 \leq i \leq t\ \left[\ M_3^t\ \wedge\ R_i^t\ \wedge\ N_{i,1}^t\ \rightarrow\ \big(N_{i,2}^{t+1}\ \wedge\ M_4^{t+1}\ \wedge\ R_{i+1}^{t+1}\big) \vee \big(N_{i,5}^{t+1}\ \wedge\ M_6^{t+1}\ \wedge\ R_{i-1}^{t+1}\big)\ \right]$$

che può essere riscritta in forma congiuntiva normale.
Sia, dunque, $P = P_1 \cup P_2 \cup \ldots \cup P_k$ l'insieme delle quintuple di NT ove, per ogni $1 \leq \ell \leq k$,

$$P_\ell = \{\langle q_{\ell;0},s_{\ell;0},s_{\ell;1},q_{\ell;1},m_{\ell;1}\rangle, \langle q_{\ell;0},s_{\ell;0},s_{\ell;2},q_{\ell;2},m_{\ell;2}\rangle, \ldots \langle q_{\ell;0},s_{\ell;0},s_{\ell;k_\ell},q_{\ell;k_\ell},m_{\ell;k_\ell}\rangle\}.$$

Inichiamo con $G_{i,\ell}^t$ il predicato seguente scritto, però, in forma congiuntiva normale

$$M_{\ell;0}^t\ \wedge\ R_i^t\ \wedge\ N_{i,\ell;0}^t\ \rightarrow\ \big(N_{i,\ell;1}^{t+1}\ \wedge\ M_{\ell;1}^{t+1}\ \wedge\ R_{i+m_{\ell;1}}^{t+1}\big) \vee \ldots \vee \big(N_{i,\ell;k_\ell}^{t+1}\ \wedge\ M_{\ell;k_\ell}^{t+1}\ \wedge\ R_{i+m_{\ell;k_\ell}}^{t+1}\big). \quad (9.3)$$

Allora l'insieme di quintuple P_ℓ corrisponde al seguente predicato: $\forall\, t \geq 0\ \ \forall\, 0 \leq i \leq t\ [\ G_{i,\ell}^t\ ]$.
In conclusione, la macchina di Turing NT è completamente descritta dal predicato

$$\forall\, t \geq 0\, \forall\, 0 \leq i \leq t\ [\ G_{i,1}^t\ \wedge\ G_{i,2}^t \wedge \ldots \wedge\ G_{i,k}^t\ ], \quad (9.4)$$

Sia $L \subseteq \Sigma^*$ un linguaggio in **NP**, e la macchina NT descritta sino ad ora sia la macchina di Turing non deterministica che accetta L in tempo polinomiale. Allora, esiste $\alpha \in \mathbb{N}$ tale che, per ogni $x \in L$, $ntime(NT,x) \leq |x|^\alpha$, ossia, NT accetta x in $|x|^\alpha$ passi. Come abbiamo già osservato, questo significa che, se NT non accetta $x \in \Sigma^*$ entro $|x|^\alpha$ passi, allora non lo accetterà nemmeno successivamente.

Vediamo, ora, come rappresentare mediante un predicato il fatto che una computazione di NT accetti.
Sia $x \in \Sigma^*$ con $x = x_0x_1\ldots x_{n-1}$. Ricordiamo che una computazione che termina è una sequenza finita di stati globali il cui primo elemento è uno stato globale iniziale e il cui ultimo elemento è uno stato globale finale e tale che lo stato globale i-esimo della sequenza è ottenuto eseguendo una quintupla a partire dallo stato globale $(i-1)$-esimo. Dobbiamo, allora, rappresentare gli stati globali, e le transizioni da uno stato globale al successivo, mediante predicati. Prima di procedere osserviamo che, poiché la computazione $NT(x)$ consta di al più n^α passi, allora, per ogni $t > n^\alpha$, non ha senso considerare transizioni che avvengono dal tempo t al tempo $t+1$; quindi, le transizioni che costituiscono la computazione $NT(x)$ sono rappresentate dal sottoinsieme delle clausole che costituiscono l'Equazione 9.4 in cui $t \leq n^\alpha$. In conclusione, le transizioni che costituiscono la computazione $NT(x)$ sono rappresentate dal predicato

$$G\ =\ G^0\ \wedge\ G^1\ \wedge \ldots \wedge\ G^{n^\alpha},$$

dove, per ogni $t = 0, \ldots, n^\alpha$,

$$G^t \;=\; [\, G^t_{0,1} \,\wedge\, G^t_{0,2} \wedge \ldots \wedge\, G^t_{0,k} \,] \,\wedge\, [\, G^t_{1,1} \,\wedge\, G^t_{1,2} \wedge \ldots \wedge\, G^t_{1,k} \,] \,\wedge \ldots \wedge\, [\, G^t_{t,1} \,\wedge\, G^t_{t,2} \wedge \ldots \wedge\, G^t_{t,k} \,]$$

rappresenta tutte e sole le transizioni che possono avvenire al tempo t.
Si osservi che G è la riscrittura in forma più esplicita (ossia, priva dei quantificatori) del Predicato 9.4. Si osservi, inoltre, che G è un predicato in forma congiuntiva normale (CNF) ossia, è la congiunzione delle clausole G^t_{ij}, con $1 \le t \le n^\alpha \;\le i \le t$ e $1 \le j \le k$, in cui ogni clausola è la congiunzione di variabili (eventualmente negate).

Occupiamoci, ora, di rappresentare ciascuno stato globale della computazione $NT(x)$. Per farlo, è necessario descrivere vincoli che devono essere rispettati da qualunque computazione di NT che accetti in al più n^α passi (punti 1)-5) che seguono) e vincoli specifici della computazione $NT(x)$ (punto 6) che segue).

1) Dobbiamo esprimere il fatto che, ad ogni istante di tempo, la macchina si trovi in uno ed un solo stato, ossia, che (a) esista un indice h tale che $M^t_h = \texttt{vero}$ e che (b) non esiste alcuna coppia di indici i, j tali che $M^t_i = \texttt{vero}$ e $M^t_h = \texttt{vero}$. Quindi, per ogni $t = 0, 1, \ldots, n^\alpha$,

$$M^t \;=\; \left[\, \left(M^t_1 \vee \ldots \vee M^t_m\right) \,\wedge\, \forall 1 \le i, j \le m : i \ne j \,\left(\neg M^t_i \vee \neg M^t_j\right)\right],$$

   che si può riscrivere in forma esplicita, privandola dei quantificatori. Infine,

$$M \;=\; M^0 \,\wedge\, M^1 \,\wedge \ldots \wedge\, M^{n^\alpha}.$$

   Si osservi che M è in forma congiuntiva normale.

2) Dobbiamo esprimere il fatto che, ad ogni istante di tempo, la testina sia posizionata sopra una ed una sola cella del nastro: per ogni $t = 0, 1, \ldots, n^\alpha$,

$$R^t \;=\; \left[\, \left(R^t_1 \vee \ldots \vee R^t_t\right) \,\wedge\, \forall 0 \le i, j \le t : i \ne j \,\left(\neg R^t_i \vee \neg R^t_j\right)\right],$$

   che si può riscrivere in forma esplicita, privandola dei quantificatori. Infine,

$$R \;=\; R^0 \,\wedge\, R^1 \,\wedge \ldots \wedge\, R^{n^\alpha}.$$

   Si osservi che R è in forma congiuntiva normale.

3) Dobbiamo esprimere il fatto che, ad ogni istante di tempo, ogni cella del nastro contenga uno ed un solo elemento di Σ^*: per ogni $t = 0, 1, \ldots, n^\alpha$,

$$N^t \;=\; \forall 0 \le i \le n^\alpha \;\left[\, \left(N^t_{i,1} \vee \ldots \vee N^t_{i,s}\right) \,\wedge\, \forall 1 \le j, h \le s : j \ne h \,\left(\neg N^t_{i,j} \vee \neg N^t_{i,h}\right)\right],$$

   che si può riscrivere in forma esplicita, privandola dei quantificatori. Infine,

$$N \;=\; N^0 \,\wedge\, N^1 \,\wedge \ldots \wedge\, N^{n^\alpha}.$$

   Si osservi che N è in forma congiuntiva normale.

4) Dobbiamo esprimere il fatto che, all'istante iniziale, la macchina si trovi nello stato interno iniziale q_0: questo è espresso dalla singola variabile M^0_0, che è in forma congiuntiva normale essendo una singola clausola costituita da una singola variabile.

5) Dobbiamo esprimere il fatto che la $NT(x)$ raggiunga lo stato di accettazione q_1 entro n^α passi:

$$\overline{M} \;=\; M^0_1 \vee\, M^1_1 \vee \ldots \vee M^{n^\alpha}_1.$$

   Si osservi che $\overline{M}$ è in forma congiuntiva normale essendo una singola clausola costituita da n^α variabili.

6) Dobbiamo esprimere il fatto che, all'istante iniziale, le prime n celle del nastro contengano l'input $x = x_0 x_1 \ldots x_{n-1}$ e le rimanenti celle contengano $\Box$. Allora, per $j = 0, \ldots, n-1$, indicato con i_j l'indice di x_j in Σ^* (cioè, $x_j = s\sigma_{i_j}$) e ricordando che $\Box = \sigma_s$:

$$\overline{N} \;=\; N^0_{0,i_0} \wedge\, N^0_{1,i_1} \wedge \ldots \wedge N^0_{(n-1),i_{n-1}} \,\wedge\, N^0_{n,s} \wedge \ldots \wedge\, N^0_{n^\alpha,s}.$$

   Si osservi che $\overline{N}$ è in forma congiuntiva normale con $n^\alpha + 1$ clausole, ciascuna contenente una singola variabile.

In conclusione, per ogni $x \in \Sigma^*$, abbiamo costruito il predicato

$$f(x) = M \wedge R \wedge N \wedge M_0^0 \wedge \overline{M} \wedge \overline{N} \wedge G.$$

associato alla computazione $NT(x)$.

Osserviamo che, per costruzione, per ogni $t > 0$, esiste una assegnazione di verità alle variabili che soddisfa il predicato $M^t \wedge R^t \wedge N^t$ se e soltanto se esiste uno stato globale SG_t di NT che corrisponde a tale assegnazione. Inoltre, per ogni $t > 0$, esiste una assegnazione di verità alle variabili che soddisfa $M^t \wedge R^t \wedge N^t$, $M^{t-1} \wedge R^{t-1} \wedge N^{t-1}$ e G^{t-1} se e soltanto se esiste una transizione dallo stato globale SG_{t-1} allo stato globale SG_t corrispondenti a tale assegnazione oppure lo stato interno in cui si trova NT nello stato globale SG_{t-1} è uno stato finale (e, non essendo possibile eseguire alcuna quintupla, tutti gli antecedenti delle implicazioni in G^{t-1} hanno valore `falso`).

Mostriamo ora che $f(x)$ è soddisfacibile se e soltanto se $NT(x)$ accetta, ossia, se e soltanto se $x \in L$.

Supponiamo che $x \in L$; allora, esiste una computazione (deterministica) di NT che accetta L. In altre parole, esiste una sequenza finita di stati globali $SG_0, SG_1, \ldots, SG_\tau$ di NT tale che $\tau \leq |x|^\alpha$, SG_0 è lo stato globale iniziale in cui, per $i = 0, \ldots, n-1$, nella cella i è scritto il carattere x_i (e le celle rimanenti contengono $\square$), SG_τ è uno stato globale di accettazione, e, per $t = 1, \ldots, \tau$, lo stato globale SG_t è ottenuto eseguendo una quintupla a partire dallo stato globale SG_{t-1}. In tal caso, per ogni $t = 0, \ldots, \tau$, assegniamo valore `vero` alle variabili M_i^t, R_j^t, $N_{\ell,h}^t$ se, nello stato globale SG_t, la macchina si trova nello stato interno q_i, la testina è posizionata sulla cella di indirizzo j, la cella di indirizzo ℓ del nastro contiene il carattere σ_h. È semplice verificare che tale assegnazione soddisfa i predicati M, R, N, $\overline{N}$, M_0^0. Inoltre, poiché NT accetta x in $\tau \leq n^\alpha$ passi, anche il predicato $\overline{M}$ è soddisfatto da tale assegnazione. Infine, in virtù della ossevazione fatta in precedenza, poiché per ogni $t = 0, \ldots, \tau - 1$ esiste una transizione dallo stato globale SG_{t-1} allo stato globale SG_t, anche il predicato G è soddisfatto. Questo dimostra che f è soddisfacibile.

Viceversa, supponiamo che f sia soddisfacibile. Allora, esiste una assegnazione di verità che, per ogni $t = 0, \ldots, |x|^\alpha$, soddisfa M^t, R^t, N^t e G^t; per ogni $t = 0, \ldots, |x|^\alpha$, indichiamo con SG_t lo stato globale corrispondente a questa assegnazione di verità: nello stato globale SG_t, la macchina si trova nello stato interno q_i, la testina è posizionata sulla cella di indirizzo j, la cella di indirizzo ℓ del nastro contiene il carattere σ_h se alle variabili M_i^t, R_j^t, $N_{\ell,h}^t$ è assegnato il valore `vero`. Inoltre, poiché G_t è soddisfatto da tale assegnazione, esiste una transizione dallo stato globale SG^t allo stato globale SG^{t+1}, oppure SG^t è uno stato finale. Inoltre, poiché deve essere $M_0^0 =$ `vero`, lo stato interno in cui si trova la macchina nello stato globale SG_0 è q_0 e, poiché deve essere soddisfatto il predicato $\overline{N}$, nello stato globale SG_0 sul nastro è scritta la parola x a partire dalla cella di indirizzo 0.
Dunque, a partire dall'assegnazione di verità che soddisfa f, abbiamo costruito una delle possibili computazioni (deterministiche) di $NT(x)$. Ma, affinché il predicato $\overline{M}$ sia soddisfatto, esiste τ, con $0 \leq \tau \leq |x|^\alpha$, tale che $M_1^\tau =$ `vero`: allora, in virtù della definizione degli stati globali corrispondenti all'assegnazione di verità, lo stato interno in cui si trova NT al passo τ della computazione di $NT(x)$ è $q_1 = q_A$. Quindi,

$$SG_0, SG_1, \ldots, SG_\tau$$

è una computazione accettante di $NT(x)$. Questo prova che $x \in L$.

Osserviamo, infine, che $f(x)$ è calcolabile in tempo (deterministico) polinomiale in $|x|$. Infatti, il numero di variabili è proporzionale a $|x|^\alpha$ (dove la costante di proporzionalità e α sono caratteristiche di NT e non dipendono da x), e la costruzione di ciascuno dei predicati

$$G^1, \ldots G^{|x|^\alpha},\ M^1, \ldots, M^t,\ R^1, \ldots, R^t,\ N^1, \ldots, N^t, \overline{N},\ \overline{M}$$

richiede tempo proporzionale a $|x|^\alpha$.

In conclusione, abbiamo mostrato come trasformare in tempo polinomiale una parola $x \in \Sigma^*$ in un predicato $f(x)$ in modo tale che $x \in L$ se e soltanto se $f(x)$ è soddisfacibile, ovvero, ricordando la definizione del problema SAT dell'Esempio 7.5 della Dispensa 7 ed osservando che $f(x)$ è un predicato in forma congiuntiva normale, $x \in L$ se e soltanto se $f(x) \in$ SAT. In altre parole, abbiamo mostrato una riduzione polinomiale da L a SAT.
Ma, poiché L è un qualunque linguaggio in **NP**, abbiamo mostrato come ridurre polinomialmente *qualunque* linguaggio in **NP** a SAT. Poiché, come abbiamo mostrato nell'Esempio 9.1, SAT $\in$ **NP**, questo dimostra il seguente **Teorema di Cook-Levin**.

Teorema 9.2: SAT *è un problema* ***NP****-completo.*

9.5 Prove di NP-completezza

Nel Paragrafo 9.4 abbiamo individuato un problema **NP**-completo, il problema SAT. Ma esistono altri problemi **NP**-completi? E come dimostrare la completezza di altri problemi in **NP**? In effetti, il Teorema di Cook-Levin ci fornisce un primo problema **NP**-completo, ma la tecnica utilizzata per prrovare la completezza di SAT non sembra essere riutilizzabile per dimostrare la completezza di altri problemi.
In effetti, dimostreremo, in questo paragrafo, la completezza di numerosi problemi, e lo strumento che utilizzeremo per farlo è il seguente teorema.

Teorema 9.3: *Sia Γ_0 un problema in **NP**. Se esiste un problema **NP**-completo riducibile a Γ_0, allora Γ_0 è **NP**-completo.*

Dimostrazione: Sia Γ_1 un problema **NP**-completo tale che $\Gamma_1 \preceq \Gamma_0$.
Poiché $\Gamma_1 \preceq \Gamma_0$, esiste una funzione $f_{10} : I_{\Gamma_1} \to I_{\Gamma_0}$ tale che $f_{10} \in \mathbf{FP}$ e, per ogni $x \in I_{\Gamma_1}$, $x \in \Gamma_1$ se e soltanto se $f(x) \in \Gamma_0$.
Poiché Γ_1 è **NP**-completo, per ogni problema $\Gamma_2 \in \mathbf{NP}$, si ha che $\Gamma_2 \preceq \Gamma_1$, e dunque esiste una funzione $f_{21} : I_{\Gamma_2} \to I_{\Gamma_1}$ tale che $f_{21} \in \mathbf{FP}$ e, per ogni $x \in I_{\Gamma_2}$, $x \in \Gamma_2$ se e soltanto se $f(x) \in \Gamma_1$.
Mostriamo ora che la composizione delle due funzioni f_{21} e f_{10} è una riduzione polinomiale da Γ_2 a Γ_0.

Sia $x \in I_{\Gamma_2}$: allora, $x \in \Gamma_2$ se e soltanto se $f_{21}(x) \in \Gamma_1$ e, inoltre, $f_{21}(x) \in \Gamma_1$ se e soltanto se $f_{10}\big(f_{21}(x)\big) \in \Gamma_0$. Se indichiamo con f_{20} la composizione delle funzioni f_{21} e f_{10}, questo dimostra che f_{20} è una riduzione da Γ_2 a Γ_0.

Poiché $f_{21} \in \mathbf{FP}$, esistono una macchina di Turing di tipo trasduttore T_{21} e un intero $k \in \mathbb{N}$ tali che, per ogni $x \in I_{\Gamma_2}$, $T_{21}(x)$ calcola $f_{21}(x)$ e $dtime(T_{21},x) \leq |x|^k$. Osserviamo che, poiché la computazione $T_{21}(x)$ deve anche scrivere il risultato $f_{21}(x)$ sul nastro di output, questo implica che $|f_{21}(x)| \leq dtime(T_{21},x)$, ossia, $|f_{21}(x)| \leq |x|^k$.
Analogamente, poiché $f_{10} \in \mathbf{FP}$, esistono una macchina di Turing di tipo trasduttore T_{10} e un intero $h \in \mathbb{N}$ tali che, per ogni $x \in I_{\Gamma_1}$, $T_{10}(x)$ calcola $f_{10}(x)$ e $dtime(T_{10},x) \leq |x|^h$. Allora, possiamo definire la seguente macchina di Turing di tipo trasduttore T_{20} che calcola f_{20}: quando la computazione $T_{20}(x)$ ha inizio, l'input $x \in I_{\Gamma_2}$ è scritto sul nastro di lavoro e, a questo punto,

1) viene eseguita la computazione $T_{21}(x)$ scrivendo il suo output $f_{21}(x)$ sul nastro di lavoro;
2) utilizzando il risultato della computazione $T_{21}(x)$, viene eseguita la computazione $T_{10}(f_{21}(x)$ ed il suo output viene scritto sul nastro di output.

Infine, in virtù del fatto che $|f_{21}(x)| \leq |x|^k$, per ogni $x \in \Gamma_2$,

$$dtime(T_{20},x) \leq |x|^k \ + \ |f_{21}(x)|^h \leq |x|^k \ + \ |x|^{kh} \leq 2|x|^{kh} \leq |x|^{kh+1}.$$

Essendo h e k due valori costanti (indipendenti dall'input x), questo dimostra che $f_{20} \in \mathbf{FP}$.

Quindi, abbiamo dimostrato che $\Gamma_2 \preceq \Gamma_0$, e, poiché Γ_2 è un qualunque problema in **NP**, questo prova che ogni problema in **NP** è riducibile polinomialmente a Γ_0. Dall'appartenenza di Γ_0 a **NP** segue che Γ_0 è **NP**-completo. □

Il teorema appena dimostrato è lo strumento che ci permetterà di dimostrare la completezza di numerosi problemi in **NP**: per dimostrare che un problema $\Gamma \in \mathbf{NP}$ è completo, dunque, sarà sufficiente individuare un problema$\Gamma_1 \in \mathbf{NP}$ di cui sia già nota la completezza ed una riduzione polinomiale $f : I_{\Gamma_1} \to I_{\Gamma}$.

9.5.1 Il problema 3-SODDISFACIBILITÀ (3SAT)

Abbiamo già ampiamente conosciuto i problemi SAT e 3SAT. Per comodità, comunque, riportiamo qui le loro formalizzazioni.
Iniziamo con la formalizzazione di SAT.

$I_{SAT} = \{f : \{\texttt{vero},\texttt{falso}\}^n \to \{\texttt{vero},\texttt{falso}\}$ tale che f è in forma congiuntiva normale $\}$;

$S_{SAT}(f) = \{(b_1,\ldots,b_n) \in \{\texttt{vero},\texttt{falso}\}^n \}$;

$\pi_{SAT}(f, S_{SAT}(f)) = \exists (b_1, \ldots, b_n) \in S_{SAT}(f) : f(b_1, \ldots, b_n) = \texttt{vero}$, ossia, sostituendo in f ogni occorrenza della variabile x_i con il valore b_i (ed ogni occorrenza di $\neg x_i$ con $\neg b_i$), per ogni $i = 1, \ldots, n$, la funzione f assume il valore `vero`.

E, di seguito, la formalizzazione di 3SAT.

$I_{3SAT} = \{f : \{\texttt{vero}, \texttt{falso}\}^n \to \{\texttt{vero}, \texttt{falso}\}$ tale che f è in forma 3-congiuntiva normale $\}$;

$S_{S3AT}(f) = \{(b_1, \ldots, b_n) \in \{\texttt{vero}, \texttt{falso}\}^n \}$;

$\pi_{3SAT}(f, S_{SAT}) = \exists (b_1, \ldots, b_n) \in \{\texttt{vero}, \texttt{falso}\}^n : f(b_1, \ldots, b_n) = \texttt{vero}$, ossia, sostituendo in f ogni occorrenza della variabile x_i con il valore b_i (ed ogni occorrenza di $\neg x_i$ con $\neg b_i$), per ogni $i = 1, \ldots, n$, la funzione f assume il valore `vero`.

Osserviamo immediatamente che l'unica differenza fra i due problemi è nella definizione dell'insieme delle istanze. In effetti, essi sono legati dalla relazione di inclusione: $I_{3SAT} \subseteq I_{SAT}$.

Nel seguito, parleremo di *letterale* per riferirci all'occorrenza di una variabile, diretta o negata, in una clausola: ad esempio, la clausola $x_1 \vee \neg x_2$ contiene i due letterali x_1 e $\neg x_2$. Inoltre, indicheremo sempre con X l'insieme delle variabili che compaiono in un predicato e rappresenteremo una assegnazione di verità come una funzione $a : X \to \{\texttt{vero}, \texttt{falso}\}$.

Sappiamo dall'Esempio 9.1 del Paragrafo 9.2 che SAT $\in$ **NP** e, poiché, come abbiamo appena osservato, le istanze di 3SAT costituiscono un sottoinsieme dell'insieme delle istanze di SAT, anche 3SAT $\in$ **NP**.

Dimostriamo, ora, la completezza di `3SAT` mediante una riduzione da `SAT` (che è l'unico problema che noi al momento sappiamo essere completo).

Sia $f \in I_{SAT}$ una istanza di SAT; indichiamo con $X = \{x_1, \ldots, x_n\}$ l'insieme delle variabili booleane che compaiono in f. Poiché f è in forma congiuntiva normale, possiamo considerare f come un insieme di clausole che *devono essere tutte soddisfatte* affinché f sia soddisfacibile, ossia, $f = \{c_1, \ldots, c_m\}$.

Mostreremo, in quanto segue, come trasformare ciascuna clausola $c_j \in f$ in un insieme C_j di clausole a 3 variabili, in cui compaiono le stesse variabili che compaiono in c_j e, in alcuni casi, variabili che non compaiono in f, in modo tale che un'assegnazione di verità soddisfa c_j se e soltanto se la stessa assegnazione alle variabili che compaiono anche in C_j soddisfa anche tutte le clausole in C_j, per ogni $j = 1, \ldots, m$.
Una volta definiti gli insiemi C_j, poniamo $\overline{f} = C_1 \cup C_2 \cup \ldots \cup C_m$: $\overline{f}$ è un predicato in forma 3-congiuntiva normale e, dunque, è una istanza di 3SAT. Inoltre, per costruzione degli insiemi C_j, una assegnazione di verità soddisfa tutte le clausole in f se e soltanto la stessa assegnazione alle variabili che compaiono anche in $\overline{f}$ soddisfa tutte le clausole in $\overline{f}$, ossia, $f \in$ `SAT` se e soltanto se $\overline{f} \in$ `3SAT`.

Vediamo, ora, per ogni $j = 1, \ldots, m$, come costruire l'insieme C_j dipendentemente dal numero di letterali che compaiono in c_j.

- Se c_j contiene un solo letterale ℓ (che, ricordiamo, è una variabile in X oppure la sua negazione): sia $Y_j = \{y_{j1}, y_{j_2}\}$ un nuovo insieme di variabili booleane; allora,

  $$C_j = \{(\ell \vee y_{j1} \vee y_{j2}), (\ell \vee \neg y_{j1} \vee y_{j2}), (\ell \vee y_{j1} \vee \neg y_{j2}), (\ell \vee \neg y_{j1} \vee \neg y_{j2})\}.$$

  La clausola $c_j = \ell$ è soddisfatta solo dall'assegnazione $\ell = \texttt{vero}$ (cioè, $x_i = \texttt{vero}$ se $\ell = x_i$, oppure $x_i = \texttt{falso}$ se $\ell = \neg x_i$) ed è immediato verificare che tutte le clausole in C_j sono soddisfatte se e soltanto se $\ell = \texttt{vero}$.

- Se c_j contiene due letterali, ossia, $c_j = \ell_1 \vee \ell_2$: sia $Y_j = \{y_j\}$ una nuova variabile booleana; allora,

  $$C_j = \{(\ell_1 \vee \ell_2 \vee y_j), (\ell_1 \vee \ell_2 \vee \neg y_j)\}.$$

  La clausola c_j è soddisfatta dalle assegnazioni tali che $\ell_1 = \texttt{vero}$ e dalle assegnazioni tali che $\ell_2 = \texttt{vero}$ (e solo da esse) ed è immediato verificare che le due clausole in C_j sono soddisfatte se e soltanto se $\ell_1 = \texttt{vero}$ oppure $\ell_2 = \texttt{vero}$.

- Se c_j contiene tre letterali, ossia, $c_j = \ell_1 \vee \ell_2 \vee \ell_3$: essa già contiene 3 letterali; allora,$C_j = \{c_j\}$.
- Se c_j contiene $k \geq 4$ letterali, ossia, $c_j = \ell_1 \vee \ell_2 \ldots \ell_k$: sia $Y_j = \{y_{j1}, \ldots, y_{jk-3}\}$ un nuovo insieme di variabili booleane; allora,

$$C_j = \{(\ell_1 \vee \ell_2 \vee y_{j1}), (\neg y_{j1} \vee \ell_3 \vee y_{j2}), \ldots, (\neg y_{ji} \vee \ell_{i+2} \vee y_{ji+1}), \ldots, (\neg y_{jk-3} \vee \ell_{k-1} \vee \ell_k)\}.$$

  La clausola c_j è soddisfatta dalle assegnazioni tali che, per almeno un indice $i = 1, \ldots, k$, $\ell_i = \texttt{vero}$ (e solo da esse) ed è immediato verificare che le $k-3$ clausole in C_j sono soddisfatte se e soltanto se $\ell_i = \texttt{vero}$ per almeno un indice $i = 1, \ldots, k$.

Costruire l'insieme C_j a partire dalla clausola c_j richiede, i tutti i 4 casi elencati, tempo lineare nel numero di letterali in c_j: poichéale numero non può essere maggiore di $2n$, è possibile costruire C_j in tempo lineare in n.
Allora costruiamo $\overline{f}$ a partire da f in tempo proporzionale a nm, e dunque in tempo in $\mathbf{O}(|f|^2)$.

In concusione, abbiamo mostrato una riduzione polinomiale da SAT a 3SAT dimostrando, in tal modo, che 3SAT è **NP**-completo.

9.5.2 Il problema VERTEX COVER (VC)

Sia $G = (V,E)$ un grafo non orientato e sia $V' \subseteq V$; V' è un *vertex cover* per G se ogni arco in E ha un estremo in V'.
Il problema VERTEX COVER consiste nel decidere, dati un grafo $G = (V,E)$ e un intero k, se G contiene un vertex cover di cardinalità $\leq k$.

Il problema VERTEX COVER (VC) può essere formalizzato nella maniera seguente:

- I_{VC} = $\{\langle G = (V,E), k\rangle :$ G è un grafo non orientato e k un intero positivo $\}$.
- $S_{VC}(G,k) = \{V' \subseteq V\}$.
- $\pi_{VC}(G,k,S_{VC}(G,k)) = \exists V' \in S_{VC}(G,k) :\ |V'| \leq k \ \wedge\ \forall (u,v) \in E\ [\ v \in V' \vee v \in V'\ ]$.

Un certificato per una istanza $x = \langle G = (V,E), k\rangle$ di VERTEX COVER è un sottoinsieme V' dei nodi di G (ossia, una soluzione possibile per l'istanza x). Poiché è possibile verificare se un certificato V' soddisfa il predicato $\rho_{VC} = |V'| \leq k \wedge \forall (u,v) \in E\ [\ v \in V' \vee v \in V'\ ]$ in tempo $\mathbf{O}(|E||V|)$, questo prova che VC $\in$ **NP**.

Per dimostrarne la completezza, mostriamo una riduzione polinomiale da 3SAT, ossia, una funzione $f : I_{3\mathrm{SAT}} \to I_{\mathrm{VC}}$, contenuta in **FP**, tale che, per ogni $\phi \in I_{\mathrm{SAT}}$, $\phi \in$ SAT se e solo se $f(\phi) \in$ VC.

Per individuare f utilizzeremo, in quanto segue, una tecnica cui faremo spesso riferimento anche in seguito quando vogliamo ridurre un problema A ad un problema B: ad ogni struttura (vedremo a breve per mezzo di esempi cosa intendiamo con questo termine) dell'istanza di A faremo corrispondere una struttura dell'istanza di B, in modo che strutture dello stesso genere presenti nell'istanza di A corrispondano a strutture dello stesso genere dell'istanza di B. A tali strutture daremo il nome di *gadget*.

Sia ϕ un predicato in forma 3-congiuntiva normale, definito sull'insieme di variabili $X = \{x_1, \ldots, x_n\}$; dunque, $\phi = \{c_1, \ldots, c_m\}$ con $c_j = \ell_{j1} \vee \ell_{j2} \vee \ell_{j3}$ e $\ell_{ji} \in X$ oppure $\neg \ell_{ji} \in X$, per $j = 1, \ldots, m$ e $i = 1,2,3$. Dobbiamo trasformare ϕ in una istanza di VC, cioè, $f(\phi) = \langle G = (V,E), k\rangle$ dove G è un grafo non orientato e $k \in \mathbb{N}$.
Per ogni variabile booleana $x_i \in X$, costruiamo un *gadget-variabile* X_i: il gadget X_i è un grafo non orientato costituito dai due nodi u_i e $\neg u_i$ e dall'arco $(u_i, \neg u_i)$. Per ogni clausola $c_j \in \phi$, costruiamo un *gadget-clausola* C_j: il gadget C_j è un grafo non orientato costituito dai tre nodi v_{j1}, v_{j2}, v_{j3} e dagli archi (v_{j1}, v_{j2}), (v_{j2}, v_{j3}), $(v_{j3}, v_{j1}$. Colleghiamo, poi, i due tipi di gadget mediante archi:

se $\ell_{ji} = x_h$, ossia, l'i-esimo letterale della clausola c_j è x_h, allora esiste l'arco (v_{ji}, u_h),

se $\ell_{ji} = \neg x_h$, ossia, l'i-esimo letterale della clausola c_j è $\neg x_h$, allora esiste l'arco $(v_{ji}, \neg u_h)$.

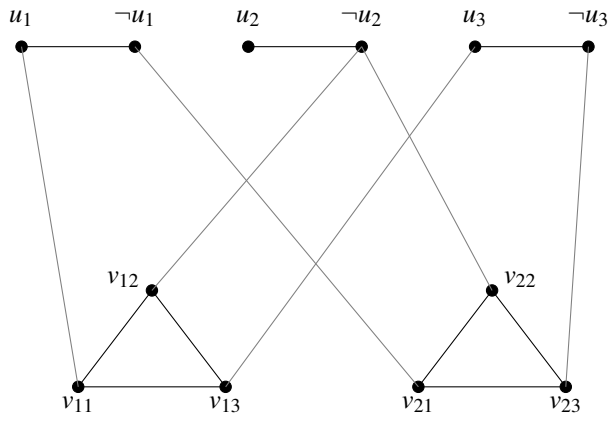

Figura 9.1: Il grafo G, istanza di VERTEX COVER, corrispondente alla coppia di clausole $c_1 = \ell_{11} \vee \ell_{12} \vee \ell_{13} = x_1 \vee \neg x_2 \vee x_3$ e $c_2 = \ell_{21} \vee \ell_{22} \vee \ell_{23} = \neg x_1 \vee \neg x_2 \vee \neg x_3$.

Il grafo G corrispondente a ϕ è, quindi, costituito da tutti i gadget-clausola, da tutti i gadget-variabile e dagli archi appena definiti. Un esempio di grafo ottenuto mediante questa trasformazione è mostrato in Figura 9.1.
Infine, poniamo $k = n + 2m$.

Banalmente, $f(\phi)$ è calcolabile deterministicamente in tempo $\mathbf{O}(nm)$ e, quindi, $f \in FP$.

Resta da dimostrare che ϕ è soddisfacibile se e soltanto se esiste un vertex cover per G di cardinalità al più $n + 2m$.
Prima di procedere con la dimostrazione, osserviamo che la cardinalità di *qualunque* vertex cover per G è almeno $n + 2m$: infatti, per ogni variabile $x_h \in X$ è necessario un nodo per coprire l'arco $(u_h, \neg u_h)$ (e, quindi, almeno uno dei due nodi deve essere nel ricoprimento) e per ogni clausola $c_j \in f$ sono necessari due nodi per coprire i tre archi (v_{j1}, v_{j2}), (v_{j2}, v_{j3}) e (v_{j3}, v_{j1}) (e, quindi, almeno due dei tre nodi devono essere nel ricoprimento) . Quindi, ciò che dovremo effettivamente dimostrare è che ϕ è soddisfacibile se e soltanto se esiste un vertex cover per G di cardinalità esattamente $n + 2m$: esso contiene esattamente un nodo di ogni gadget-variabile ed esattamente due nodi di ogni gadget-clausola.

Supponiamo che esista un vertex cover V' per G di cardinalità esattamente $n + 2m$. Allora, V' deve contenere esattamente un nodo per ogni gadget-variabile X_h: quindi, possiamo associare a V' l'assegnazione di verità a per X in cui, per ogni $h = 1, \ldots, n$, $a(x_h) = \texttt{vero}$ se e soltanto se $u_h \in V'$.
Mostriamo, ora, che a soddisfa ϕ. Ricordiamo che, per ogni gadget-clausola C_j, esattamente due nodi del gadget sono contenuti in V' e che ogni nodo del gadget è adiacente ad esattamente un gadget-variabile (più precisamente, ad uno dei suoi due nodi): quindi, uno degli archi che connettono un nodo di C_j a nodi in $\cup_{h=1}^{n} X_h$ non è coperto dai nodi in C_j che appartengono a V' e, poiché V' è un vertex cover, tale arco deve essere coperto da qualche nodo in $\cup_{i=h}^{n} X_h$. Allora, per ogni $j = 1, \ldots, m$, esiste $i \in \{1, 2, 3\}$ tale che il nodo in $\cup_{i=h}^{n} X_h$ adiacente a v_{ji} è contenuto in V': chiamiamo y_j tale nodo. Per costruzione della assegnazione a, poiché $y_j \in V'$, se $y_j = x_h$ (per qualche $h = 1, \ldots, n$) allora $a(x_h) = \texttt{vero}$ altrimenti, se $y_j = \neg x_h$, allora $a(x_h) = \texttt{falso}$: questo implica che, per ogni $j = 1, \ldots, m$, l'assegnazione a soddisfa la clausola c_j. Dunque, a soddisfa ϕ.

Viceversa, supponiamo che ϕ sia soddisfacibile e sia a una assegnazione di verità che soddisfa ϕ. Poniamo

$$V'_X = \bigcup_{i=1}^{n} \left[\{u_i \in X_i : a(x_i) = \texttt{vero}\} \cup \{\neg u_i \in X_i : a(x_i) = \texttt{falso}\} \right].$$

Scegliamo, inoltre, per ogni $j = 1, \ldots, m$, un nodo y_j in C_j corrispondente ad un letterale soddisfatto da a (ossia, se y_j è una variabile x allora $a(x) = \texttt{vero}$, se y_j è la negazione di una variabile x allora $a(x) = \texttt{falso}$): poiché a soddisfa ϕ,

il nodo y_j esiste per ogni $j = 1,\ldots,m$. Poniamo, dunque, per ogni $j = 1,\ldots,m$, $V'_j = \{v_{j1}, v_{j2}, v_{j3}\} - \{y_j\}$ e

$$V' = V'_X \bigcup_{j=1}^{m} V'_j.$$

Grazie a considerazioni analoghe a quelle che ci hanno permesso di dimostrare la proposizione inversa, è facile mostrare che V' è un Vertex Cover per G.

Questo completa la dimostrazione che f è una riduzione polinomiale da 3SAT a VC e che, dunque, VC è **NP**-completo.

9.5.3 I problemi INDEPENDENT SET (IS) e CLIQUE (CL)

Utilizzeremo il problema VERTEX COVER per dimostrare la completezza di numerosi problemi. In questo paragrafo iniziamo con un primo problema, INDEPENDENT SET, la cui completezza verrà, a sua volta, utilizzata per provare la completezza del secondo, CLIQUE. Essi vengono presentati insieme perché le due riduzioni che li riguardano sono entrambe molto semplici, essendo poco più che letture inchiave diversa di uno stesso problema.

Sia $G = (V,E)$ un grafo non orientato e sia $I \subseteq V$; I è un Insieme Indipendente per G se, per ogni coppia di nodi $u, v \in I$ si ha che $(u,v) \notin E$. Il problema INDEPENDENT SET (IS) consiste nel decidere, dati un grafo $G = (V,E)$ e un intero k, se G contiene un insieme indipendente di cardinalità $\geq k$.

Il problema INDEPENDENT SET può essere formalizzato nella maniera seguente:

- $I_{IS} = \{\langle G = (V,E), k\rangle : \ G$ è un grafo non orientato e $k \in \mathbb{N}\}$.
- $S_{IS}(G,k) = \{I \subseteq V\}$.
- $\pi_{IS}(G,k,S_{IS}(G,k)) = \exists I \in S_{IS}(G,k) : |I| \geq k \ \wedge \ \forall u,v \in I[\ (u,v) \notin E\ ]$.

Un certificato per una istanza $x = \langle G = (V,E), k\rangle$ di INDEPENDENT SET è un sottoinsieme I dei nodi di G (ossia, una soluzione possibile per l'istanza x). Poiché è possibile verificare se un certificato I soddisfa il predicato $\rho_{IS} = |I| \geq k \ \wedge \forall (u,v) \in E\ [\ v \in V' \vee v \in V'\ ]$ in tempo $\mathbf{O}(|E||V|^2)$, questo prova che IS $\in$ **NP**.

Dimostriamo, ora, la completezza di IS presentando una riduzione da VC.
A questo scopo, osserviamo che, dato un qualunque grafo non orientato $G = (V,E)$ e un sottoinsieme $V' \subseteq V$ dell'insieme dei suoi nodi, V' è un Vertex Cover per G se e soltanto se $V - V'$ è un insieme indipendente. Infatti, se V' è un Vertex Cover allora, per ogni arco $(u,v) \in E$ non può accadere che $u \in V - V'$ e $v \in V - V'$: in altre parole, per ogni coppia di nodi $u, v \in V - V'$, $(u,v) \notin E$, ossia, $V - V'$ è un insieme indipendente. Viceversa, se $V - V'$ è un insieme indipendente allora, per ogni coppia di nodi $u, v \in V - V'$, $(u,v) \notin E$, ossia , per ogni arco $(u,v) \in E$ deve accadere che $u \in V'$ oppure $v \in V'$: dunque, V' è un Vertex Cover per G.
Sulla base della precedente affermazione, la funzione f che riduce VERTEX COVER a INDEPENDENT SET è definita come segue: per ogni istanza $\langle G = (V,E), k\rangle$ di VC, $f(G,k) = \langle G = (V,E), |V| - k\rangle$. Per quanto osservato, G ammette un Vertex Cover V' di $\leq k$ nodi se e soltanto se G contiene un insieme indipendente $I = V' - V$ di $\geq |V| - k$ nodi.
Banalmente, f è calcolabile in tempo lineare nella lunghezza dell'istanza di VC, e questo prova che IS è **NP**-completo.

Sia $G = (V,E)$ un grafo non orientato e sia $V' \subseteq V$; C è una clique se, ogni nodo in C è adiacente ad ogni altro nodo in C. Il problema CLIQUE (CL) consiste nel decidere, dati un grafo $G = (V,E)$ e un intero k, se G contiene una clique di cardinalità $\geq k$.

Il problema CLIQUE può essere formalizzato nella maniera seguente:

- $I_{CL} = \{\langle G = (V,E), k\rangle : \ G$ è un grafo non orientato e $k \in \mathbb{N}\}$.
- $S_{CL}(G,k) = \{C \subseteq V\}$.
- $\pi_{CL}(G,k,S_{CL}(G,k)) = \exists C \in S_{CL}(G,k) : |C| \geq k \ \wedge \ \forall u,v \in C[\ (u,v) \in E\ ]$.

Un certificato per una istanza $x = \langle G = (V,E), k\rangle$ di CLIQUE è un sottoinsieme C dei nodi di G (ossia, una soluzione possibile per l'istanza x). Poiché è possibile verificare se un certificato C soddisfa il predicato $\rho_{CL} = |C| \geq k \wedge \forall u, v \in C\ [\ (u,v) \in E\ ]$ in tempo $\mathbf{O}(|E||V|^2)$, questo prova che CL $\in$ **NP**.

Dimostriamo, ora, la completezza di CL presentando una riduzione da IS.
A questo scopo, dato un grafo non orientato $G = (V,E)$, definiamo il suo *grafo complemento* $G^c = (V,E^C)$ come il grafo definito sullo stesso insieme V di nodi di G ed il cui insieme degli archi E^c è tale che, per ogni coppia di nodi $u, v \in V$,

$$(u,v) \in E^c \Leftrightarrow (u,v) \notin E,$$

ossia, una coppia di nodi è adiacente in G^c se e soltanto se non è adiacente in G.
Segue dalla definizione di grafo complemento che un sottoinsieme $U \subseteq V$ di nodi è un insieme indipendente in G se e soltanto se U è una clique in G^c.
Allora, la funzione f che riduce INDEPENDENT SET a CLIQUE è definita come segue: per ogni istanza $\langle G = (V,E), k\rangle$ di CL, $f(G,k) = \langle G^c = (V,E^c), k\rangle$. Per quanto osservato, G contiene un insieme indipendente I di $\geq k$ nodi se e soltanto se I è una clique (di $\geq k$ nodi) in G^c.
Banalmente, f è calcolabile in tempo lineare nella lunghezza dell'istanza di VC, e questo prova che CL è **NP**-completo.

9.5.4 Il problema DOMINATING SET (DS)

Sia $G = (V,E)$ un grafo non orientato e sia $D \subseteq V$; D è un insieme dominante per G se ogni nodo in $V - D$ ha un vicino in D. Il problema DOMINATING SET consiste nel decidere, dati un grafo $G = (V,E)$ e un intero k, se G contiene un insieme dominante di cardinalità $\leq k$.

Il problema DOMINATING SET può essere formalizzato nella maniera seguente:

- I_{DS} = $\{\langle G = (V,E), k\rangle :$ G è un grafo non orientato e k un intero positivo $\}$.
- $S_{DS}(G,k) = \{D \subseteq V\}$.
- $\pi_{DS}(G,k,S_{DS}(G,k)) = \exists D \in S_{DS}(G,k) : |D| \leq k\ \wedge\ \forall u \in V - D[\exists v \in D : (u,v) \in E]$.

Il problema è in **NP**: infatti, un certificato per una istanza $x = \langle G = (V,E), k\rangle$ di DOMINATINGT SET è un sottoinsieme D dei nodi di G (ossia, una soluzione possibile per l'istanza x). Poiché è possibile verificare se un certificato D soddisfa il predicato $\rho_{DS} = |D| \leq k\ \wedge \forall u \in V - D\ [\exists v \in D : (u,v) \in E\ ]$ in tempo $\mathbf{O}(|E||V|^2)$, questo prova che DS $\in$ **NP**.

Descriviamo, ora, una funzione f corrispondente ad una riduzione polinomiale da VERTEX COVER a DOMINATING SET. Sia $\langle G = (V,E), k\rangle$ una istanza di VERTEX COVER; allora, $f(G,k) = \langle H = (W,F), k\rangle$ dove H è il grafo di seguito descritto.

- W è ottenuto eliminando da V i nodi che in G sono isolati ed aggiungendo all'insieme risultante un nuovo nodo per ogni arco presente in E, ossia,

  $$W = \big(V - \{u \in V : \forall v \in V[(u,v) \notin E]\}\big) \cup \{x_e : e \in E\};$$

  denotiamo con X l'insieme dei nodi di H corrispondenti agli archi di G: $X = \{x_e : e \in E\}$.
- Sia $Y = \{(u,x_e),(v,x_e) : e = (u,v) \in E\}$; allora, $F = E \cup Y$.

Un esempio della riduzione è mostrato in Figura 9.2

Mostriamo ora che, dato un grafo non orientato G ed un intero positivo k, $f(G,k) = \langle H,k\rangle$ è calcolabile in tempo polinomiale in $|G|$ e k, come mostrato nelle linee di codice che seguono:

1) $W \leftarrow \emptyset$;
2) **for** $(u \in V)$ **do**
   if $(\exists e \in E : e = (u,v))$ **then** $W \rightarrow W \cup \{u\}$;

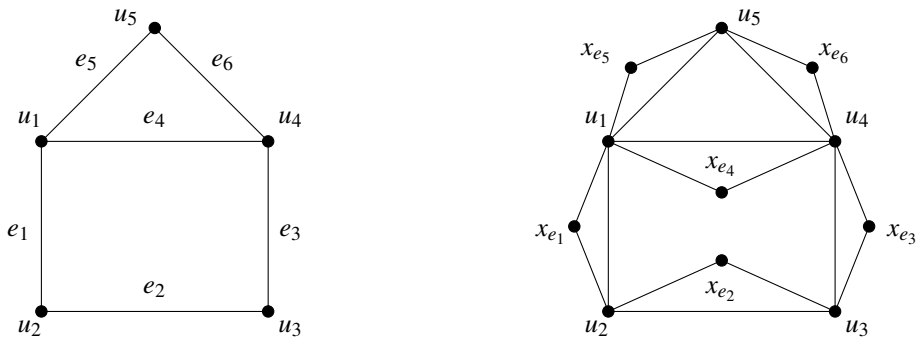

Figura 9.2: Il grafo G, istanza di VERTEX COVER ed il suo corrispondente H tramite la funzione f. In G sono stati anche evidenziati i nomi degli archi: all'arco e_i di G corrisponde il nodo x_{e_i} di H, per $i = 1, \ldots, 6$.

3) $X \leftarrow \emptyset$;

4) **for** $(e \in E)$ **do** $X \leftarrow X \cup \{x_e\}$;

5) $W \leftarrow V \cup X$;

6) $Y \leftarrow \emptyset$;

7) **for** $(e = (u,v) \in E)$ **do** $Y \leftarrow Y \cup \{(u,x_e),(v,x_e)\}$;

8) $F \leftarrow E \cup Y$;

9) **Output**: $H = (W,F)$.

Si verifica facilmente che questo richiede tempo in $O(|E||V|)$.

Resta da far vedere che se $\langle G = (V,E),k\rangle$ è una istanza sì di VERTEX COVER allora $f(G,k) = \langle H = (W,F),k\rangle$ è una istanza sì di DOMINATING SET. A questo scopo, dato un insieme dominante C per un grafo, diciamo che un nodo b non contenuto in C è *dominato* da un nodo $c \in C$ se (b,c) è un arco del grafo.
Osserviamo preliminarmente che, se $u \in V$ è un nodo isolato in G, allora non è necessario che u appartenga ad alcun Vertex Cover per G, ovvero, G ammette un Vertex Cover di cardinalità k se e soltanto se $G - \{u\}$ ammette un Vertex Cover di cardinalità k. Allora, senza perdita di generalità, possiamo assumere che G non contenga nodi isolati e che, quindi, sia $W = V \cup X$ e $F = E \cup Y$.
Supponiamo, allora, che $G = (V,E)$ ammetta un Vertex Cover di k nodi: in tal caso, esiste $V' \subseteq V$ tale che $|V'| \leq k$ e, per ogni arco $(u,v) \in E$, $u \in V'$ oppure $v \in V'$. Proviamo, ora, che $D = V'$ è un Dominating Set per H. Infatti, per ogni $u \in W - D$, sono possibili i due casi seguenti: $u \in V$ oppure $u \in X$. Di seguito, analizziamo separatamente i due casi.

- Se $u \in V$ allora, poiché G non contiene nodi isolati, u è adiacente a qualche nodo $v \in V$ (ossia, $(u,v) \in E$). In questo caso, poiché V' è un vertex cover per G e poiché $u \notin V'$, allora $v \in V' = D$ e, quindi, u è dominato da v.
- Se $u \in X$ allora $u = x_e$, per qualche $e = (v_1,v_2) \in E$. Poiché $v_1 \in V' \vee v_2 \in V'$ e $(x_e,v_1) \in F \wedge (x_e,v_2) \in F$, allora $u = x_e$ è dominato da v_1 o da v_2.

Dunque, D è un insieme dominante per G.

Viceversa, supponiamo che H ammetta un insieme dominante D di k nodi. Osserviamo che, se D contiene qualche elemento $x_e \in X$, allora, detto $e = (u,v)$, anche l'insieme $D' = (D - \{x_e\}) \cup \{u\}$ (così come $D'' = (D - \{x_e\}) \cup \{v\}$) è un insieme dominante per H. Di conseguenza, se esiste un insieme dominante per H di k nodi allora esiste anche un insieme dominante per H di k nodi che non contiene elementi di X. Pertanto, senza perdita di generalità, possiamo

assumere che D contenga solo elementi di V, ossia, $D \subseteq V$.
Mostriamo, ora, che D è un Vertex Cover per G. Sia $e = (u,v) \in E$ un qualunque arco di G: allora, per costruzione, $x_e \in X$, ossia, x_e è un nodo di H. Poiché D è un insieme dominante per H e poiché $D \cap X = \emptyset$, x_e deve essere dominato da qualche nodo in D. Ma x_e è adiacente in H soltanto a u e a v: allora $u \in D$ oppure $v \in D$. Dunque, D è un Vertex Cover per G.

9.5.5 Problemi di colorabilità

Dato un grafo non orientato $G = (V,E)$, una *colorazione* di G con k colori è una funzione $c : V \to \{1,\ldots,k\}$ che assegna valori (ossia, colori) diversi a nodi adiacenti. Il problema COLORABILITÀ consiste nel decidere, dati un grafo non orientato $G = (V,E)$ e un intero k, se G può essere colorato con k colori.

Il problema COLORABILITÀ (da ora in poi, in breve, COL) può essere formalizzato nella maniera seguente:

- $I_{\mathrm{COL}} = \{\langle G = (V,E), k\rangle : G \text{ è un grafo non orientato e } k \in \mathbb{N}\}$;
- $S_{\mathrm{COL}}(G,k) = \{\chi : V \to \{1,\ldots,k\}\}$;
- $\pi_{\mathrm{COL}}(G,k,S_{\mathrm{COL}}(G,k)) = \exists \chi \in S_{\mathrm{COL}}(G,k) :\ \forall (u,v) \in E : \chi(u) \neq \chi(v)$.

Un certicato per una istanza $x = \langle G = (V,E), k\rangle$ di COLORABILITÀ è una assegnazione χ di valori in $\{1,\ldots,k\}$ ai nodi di G (ossia, una soluzione possibile per listanza x). Poiché è possibile vericare se un certicato χ soddisfa il predicato $\rho_{\mathrm{COL}} = \forall (u,v) \in E : \chi(u) \neq \chi(v)$ in tempo $\mathbf{O}(|E||V|^2)$, questo prova che COL $\in$ **NP**.

Del problema COLORABILITÀ sono stati studiati casi particolari in cui *il numero k di colori non è dato in input, ma è costante*. Per sottolineare l'indipendenza del numero di colori dalle istanze, questi casi particolari del problema COLORABILITÀ vengono chiamati problemi di k-COLORABILITÀ (in breve, k-COL).
Abbiamo già incontrato il problema 2-COL nella Dispensa 8 e, come abbiamo dimostrato, esso è contenuto nella classe **P**. Ci occuperemo, ora, del problema 3-COL dimostrandone la completezza per la classe **NP** (in cui è contenuto, essendo un caso particolare di COLORABILITÀ). Al termine della sezione, mostreremo come da ciò si deduca la completezza di k-COL, per ogni $k > 3$, e del più generale COL.

Per completezza, presentiamo anche la formalizzazione del problema 3-COLORABILITÀ:

- $I_{3-\mathrm{COL}} = \{G = (V,E) : G \text{ è un grafo non orientato}\}$;
- $S_{3-\mathrm{COL}}(G) = \{\chi : V \to \{1,2,3\}\}$;
- $\pi_{3-\mathrm{COL}}(G,S_{3-\mathrm{COL}}(G)) = \exists \chi \in S_{3-\mathrm{COL}}(G) : \forall (u,v) \in E : \chi(u) \neq \chi(v)$.

Per dimostrare la completezza di 3-COL, descriviamo ora una riduzione polinomiale dal problema 3-SAT.
Sia dunque $f(x_1,x_2,\ldots,x_n) = c_1 \wedge c_2 \wedge \ldots \wedge c_m$ in cui ogni clausola c_j è una disgiunzione di tre letterali in X, ovvero, per ogni $j = 1,\ldots,m$, $c_j = l_{j_1} \vee l_{j_2} \vee l_{j_3}$ con $l_{j_1}, l_{j_2}, l_{j_3} \in \{x_1,x_2,\ldots,x_n\} \cup \{\neg x_1, \neg x_2, \ldots, \neg x_n\}$.
Vogliamo associare ad f un grafo $G_f = (V_f, E_f)$ tale che G_f è 3-colorabile se e soltanto se f è soddisfacibile.

Ricordiamo che con il termine *letterale* ci riferiamo ad una variabile booleana o alla sua negazione. Innanzi tutto, associamo ad ogni variabile booleana x_i, $i = 1,\ldots,n$, una coppia di nodi u_i e v_i del grafo G_f che, intutitivamente, corrispondono, rispettivamente, ai due letterali x_i e $\neg x_i$. Indichiamo l'insieme di tali nodi con $W = \cup_{i=1}^{n} \{u_i, v_i\}$. Ci proponiamo di fare in modo che

1) per ogni $i = 1,\ldots,n$, u_i e v_i debbano essere necessariamente colorati con colori diversi, ed inoltre
2) esista un colore $h \in \{1,2,3\}$ tale che nessun nodo u_i o v_i, $i = 1,\ldots,n$, sia colorato con il colore h, e infine
3) per ogni assegnazione di verità a che soddisfa f, esista una colorazione effettiva di G_f tale che tutti i nodi associati a letterali cui a assegna valore `vero` siano colorati con lo stesso colore; come conseguenza di questo vincolo e di quelli ai due punti precedenti, richiediamo che, nella colorazione effettiva di G_f corrispondente ad a, tutti i nodi associati a letterali cui a assegna valore `falso` siano colorati con lo stesso colore e che tale colore sia diverso da quello utilizzato per (tutti) i nodi associati a letterali che hanno ricevuto il valore `vero`.

Il punto 1) viene risolto semplicemente inserendo in G_f l'arco (u_i, v_i), per ogni $i = 1, \ldots, n$.
Anche il punto 2) ha una soluzione immediata: è sufficiente inserire un nodo R in G_f e collegarlo a tutti i nodi u_i e a tutti i nodi v_i, per $i = 1, \ldots, n$. In questo modo si generano in G_f n cicli di lunghezza 3 (ad esempio, il ciclo fra i nodi u_1, v_1 ed R) i cui nodi devono essere necessariamente colorati con colori differenti. Come conseguenza, nessun nodo u_i e nessun nodo v_i, $i = 1, \ldots, n$, può essere colorato con lo stesso colore di R in alcuna colorazione effettiva di G_f.
Resta da gestire il punto 3). Osserviamo, intanto, che il punto 3) fa esplicito riferimento alla soddisfacibilità della funzione f e che fino ad ora sono state rappresentate le sole variabili booleane che servono a comporre la funzione f, ma la struttura di f non è stata considerata in alcun modo. Iniziamo, dunque, ad inserire in G_f una serie di *componenti* (o, come abbiamo visto, *gadget*) che corrispondano alle clausole in f. Intuitivamente, collegheremo la componente corrispondente alla clausola c_j di f con i letterali che in essa compaiono: quello che ci proponiamo di ottenere è che ciascuna clausola sia collegata con (almeno) un nodo associato ad una variabile booleana (ossia, un nodo u_i oppure un nodo v_i) colorato con il colore associato al valore `vero` se e soltanto se f è soddisfacibile. Osserviamo che abbiamo già garantito che i nodi corrispondenti a letterali non possono essere colorati con il colore di R (diciamo, il colore 3); poiché in una formula booleana il valore `vero` ha un significato sostanzialmente diverso dal valore `falso` mentre i colori 1 e 2 sono sino ad ora indistinguibili, abbiamo bisogno di un modo per distinguere fra essi: introduciamo, dunque, due nuovi nodi, T ed F (True e False) collegati fra loro e con R (così da imporre che i loro colori siano differenti) e cerchiamo di imporre che una clausola sia soddisfacibile se e soltanto se la componente corrispondente è collegata ad (almeno) un nodo-letterale che ha lo stesso colore di T.

Ciascuna clausola $c_j = l_{j_1} \vee l_{j_2} \vee l_{j_3}$, $j = 1, 2, \ldots, m$, corrisponde ad un sottografo di G_f costituito dai 5 nodi λ_{j_1}, λ_{j_2}, λ_{j_3}, α_j e β_j:

- ciascuno dei nodi λ_{j_1}, λ_{j_2} e λ_{j_3} è collegato con il nodo associato alla variabile booleana (eventualmente negata) ad esso corrispondente: ad esempio, se $l_{j_1} = x_i$ allora $(\lambda_{j_1}, u_i) \in E_f$, altrimenti, se $l_{j_1} = \neg x_i$ allora $(\lambda_{j_1}, v_i) \in E_f$;
- i nodi λ_{j_1}, α_j e T formano un triangolo;
- i nodi λ_{j_2}, λ_{j_3} e β_j formano un triangolo;
- i nodi α_j e β_j sono collegati da un arco.

In figura 9.3 è mostrata la parte di grafo corrispondente alla clausola $c_j = x_1 \vee \neg x_2 \vee \neg x_3$.

È lasciato come semplice esercizio dimostrare che G_f può essere calcolato in tempo polinomiale in $|f|$.

Rimane da dimostrare che f è soddisfacibile se e soltanto se $G_f = (V_f, E_f)$ è 3-colorabile.

Supponiamo inizialmente che f sia soddisfacibile: allora esiste una assegnazione di verità a per le variabili $x_1, x_2, \ldots, x_n$ tale che, per ogni $j = 1, 2, \ldots, m$, la clausola c_j di f contiene un letterale l_{j_i} tale che $a(l_{j_i}) = \texttt{vero}$. Definiamo allora, a partire da a, la funzione $\chi_a : V_f \to \{1, 2, 3\}$ seguente:

- $\chi_a(R) = 3$, $\chi_a(T) = 1$, $\chi_a(F) = 2$;
- per ogni $i = 1, 2, \ldots, n$, se $a(x_i) = \texttt{vero}$ allora $\chi_a(u_i) = 1$ e $\chi_a(v_i) = 2$, altrimenti $\chi_a(u_i) = 2$ e $\chi_a(v_i) = 1$;
- per ogni $j = 1, 2, \ldots, m$ e per $h = 1, 2, 3$, deriviamo la colorazione dei nodi corrispondenti a c_j considerando le 7 possibili assegnazioni di verità ad l_{j_1}, l_{j_2} e l_{j_3} che soddisfano c_j
  1. se $a(l_{j_1}) = a(l_{j_2}) = a(l_{j_3}) = \texttt{vero}$, allora $\chi_a(\lambda_{j_1}) = 3$, $\chi_a(\lambda_{j_2}) = 2$, $\chi_a(\lambda_{j_3}) = 3$, $\chi_a(\alpha_j) = 2$ e $\chi_a(\beta_j) = 1$;
  2. se $a(l_{j_1}) = a(l_{j_2}) = \texttt{vero}$ e $a(l_{j_3}) = \texttt{falso}$, allora $\chi_a(\lambda_{j_1}) = 3$, $\chi_a(\lambda_{j_2}) = 2$, $\chi_a(\lambda_{j_3}) = 3$, $\chi_a(\alpha_j) = 2$ e $\chi_a(\beta_j) = 1$;
  3. se $a(l_{j_1}) = a(l_{j_3}) = \texttt{vero}$ e $a(l_{j_2}) = \texttt{falso}$, allora $\chi_a(\lambda_{j_1}) = 3$, $\chi_a(\lambda_{j_2}) = 3$, $\chi_a(\lambda_{j_3}) = 2$, $\chi_a(\alpha_j) = 2$ e $\chi_a(\beta_j) = 1$;
  4. se $a(l_{j_1}) = \texttt{vero}$ e $a(l_{j_2}) = a(l_{j_3}) = \texttt{falso}$, allora $\chi_a(\lambda_{j_1}) = 2$, $\chi_a(\lambda_{j_2}) = 1$, $\chi_a(\lambda_{j_3}) = 3$, $\chi_a(\alpha_j) = 3$ e $\chi_a(\beta_j) = 2$;
  5. se $a(l_{j_1}) = \texttt{falso}$ e $a(l_{j_2}) = a(l_{j_3}) = \texttt{vero}$, allora $\chi_a(\lambda_{j_1}) = 3$, $\chi_a(\lambda_{j_2}) = 2$, $\chi_a(\lambda_{j_3}) = 3$, $\chi_a(\alpha_j) = 2$ e $\chi_a(\beta_j) = 1$;

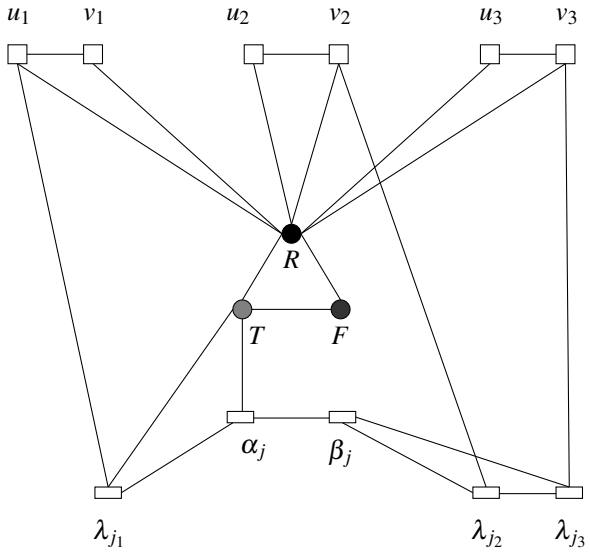

Figura 9.3: Grafo corrispondente alla clausola $c_j = x_1 \vee \neg x_2 \vee \neg x_3$. I tre nodi R, T, ed F sono stati colorati con toni differenti di grigio per evidenziare che devono essere colorati con colori differenti. Inoltre, per chiarezza, i nodi corrispondenti a variabili booleane, a clausole, e i tre nodi R, T, F sono stati disegnati, rispettivamente, come quadrati, rettangoli e cerchi.

6. se $a(l_{j_1}) = a(l_{j_3}) =$ `falso` e $a(l_{j_3}) =$ `vero`, allora $\chi_a(\lambda_{j_1}) = 3$, $\chi_a(\lambda_{j_2}) = 2$, $\chi_a(\lambda_{j_3}) = 3$, $\chi_a(\alpha_j) = 2$ e $\chi_a(\beta_j) = 1$;
7. se $a(l_{j_1}) = a(l_{j_2}) =$ `falso` e $a(l_{j_3}) =$ `vero`, allora $\chi_a(\lambda_{j_1}) = 3$, $\chi_a(\lambda_{j_2}) = 3$, $\chi_a(\lambda_{j_3}) = 2$, $\chi_a(\alpha_j) = 2$ e $\chi_a(\beta_j) = 1$.

Lasciamo come (semplice) esercizio la verifica che χ_a sia una colorazione effettiva di G_f, ossia, che $\pi_{3COL}(G_f, \chi_a) =$ `vero`.

Mostriamo, infine, che se G_f è 3-colorabile allora f è soddisfacibile. Osserviamo, innanzi tutto, che, poiché al fine di soddisfare il predicato π_{3COL} i nodi che corrispondono alle variabili booleane possono essere colorati solo con due colori (senza perdita di generalità, con i colori 1 e 2) e, inoltre, i nodi di ciascuna coppia del tipo $\langle u_i, v_i \rangle$ devono avere sempre colore differente, allora ogni colorazione di tali nodi corrisponde ad una assegnazione di verità per le variabili booleane $x_1, x_2, \ldots, x_n$.
Dimostriamo, innanzi tutto, che ogni colorazione χ di G_f tale che $\pi_{3COL}(G_f, \chi) =$ `vero` non può colorare tutti i letterali di una clausola con il colore $\chi(F)$ (senza perdita di generalità, sia esso il colore 2). Se infatti, per assurdo, esistesse una clausola $c_j = l_{j_1} \vee l_{j_2} \vee l_{j_3}$ tale che i nodi in W adiacenti a λ_{j_1}, λ_{j_2} e λ_{j_3} (nell'esempio in figura, i nodi u_1, v_2 e v_3) fossero tutti colorati con il colore $\chi(F) = 2$, allora i due nodi λ_{j_2} e λ_{j_3} dovrebbero essere colorati uno con il colore $1 = \chi(T)$ e l'altro con il colore $3 = \chi(R)$. Ma allora il nodo β_j dovrebbe essere colorato con il colore 2 e, di conseguenza, il nodo α_j (adiacente anche al nodo T) con il colore 3. Ora, il nodo λ_{j_1} è adiacente, al nodo T (colorato 1), al nodo α_j (colorato 3) e al nodo in W corrispondente ad l_{j_1} (che, per ipotesi, ha colore 2): dunque nessun colore potrebbe essere assegnato a α_j in modo tale da soddisfare il predicato π_{3COL}.
Questo dimostra che ogni 3-colorazione χ di G_f tale che $\pi_{3COL}(G_f, \chi) =$ `vero` corrisponde ad una assegnazione di verità alle variabili booleane di f tale che ogni clausola di f contiene un letterale cui è stato assegnato il valore `vero`. In altri termini, ogni 3-colorazione χ di G_f tale che $\pi_{3COL}(G_f, \chi) =$ `vero` corrisponde ad una assegnazione di verità

alle variabili booleane di f che soddisfa f. Dunque, se G_f è 3-colorabile allora f è soddisfacibile.

Questo completa la dimostrazione che 3-COL è **NP**-completo.

La dimostrazione di **NP**-completezza di 3-COL può ora essere utilizzata per mostrare che anche la versione generale COLORABILITÀ e tutte le versioni di k-COLORABILITÀ sono **NP**-complete.

Per quanto riguarda il problema COLORABILITÀ, la trasformazione di un'istanza $\langle G=(V,E)\rangle \in I_{3-\mathrm{COL}}$ di 3-COL ad una istanza $\langle G'=(V',E'),k\rangle \in I_{3-\mathrm{COL}}$ di COL è immediata: è sufficiante lasciare invariato il grafo G, ossia, porre $G'=G$, e scegliere $k=3$. Banalmente, $\langle G=(V,E)\rangle \in 3-\mathrm{COL}$ se e soltanto se $\langle G=(V,E),3\rangle \in \mathrm{COL}$, ossia, quella che abbiamo descritto è una riduzione da 3-COL a COL, provando, cos`, che COL è **NP**-completo.

La dimostrazione che k-COL è **NP**-completo per ogni $k>3$ è leggeremente più complessa e si basa su un ragionamento induttivo.
Cominciamo a mostrare una riduzione da k-COL a $(k+1)$-COL. Sia $\langle G=(V,E)\rangle \in I_{k-\mathrm{COL}}$ una istanza di k-COL; la corrispondente istanza di $(k+1)$-COL è $\langle G'=(V',E')\rangle \in I_{(k+1)-\mathrm{COL}}$ con $V'=V\cup\{x\}$, ove x è un nuovo nodo (non contenuto in V), e $E'=E\cup\{(u,x):u\in V\}$. Cioè, G' è ottenuto aggiungendo a G il nodo x e gli archi fra x e tutti i nodi di G. Poiché è adiacente ad ogni elemento di V, x deve essere colorato con colore diverso da quello con cui è colorato qualunque nodo in V: dunque, G è colorabile con k colori se e soltanto se G' è colorabile con $k+1$ colori, e questo prova che si tratta effettivamente di una riduzione (polinomiale).
La riduzione appena descritta, nel caso $k=3$ dimostra che 4-COL è **NP**-completo. Poi, che 5-COL è **NP**-completo, e così via, induttivamente, che k-COL è **NP**-completo per ogni $k>3$.

9.5.6 Il problema CICLO HAMILTONIANO (HC)

Dato un grafo non orientato $G=(V,E)$, un *ciclo hamiltoniano* in G è un ciclo che passa attraverso ogni nodo una ed una sola volta. Naturalmente, non è detto che un grafo contenga un ciclo hamiltoniano: ad esempio, gli alberi, essendo aciclici, non contengono cicli hamiltoniani. Il problema CICLO HAMILTONIANO (in breve, HC) consiste nel decidere se un dato un grafo non orientato $G=(V,E)$ contiene un ciclo hamiltoniano.

Il problema CICLO HAMILTONIANO può essere formalizzato nella maniera seguente:

- $I_{\mathrm{HC}}=\{\langle G=(V,E)\rangle : G \text{ è un grafo non orientato }\}$;
- $S_{\mathrm{HC}}(G)=\{\psi : V\to\{1,\ldots,|V|\}\}$, ossia, $S_{\mathrm{HC}}(G)$ è l'insieme degli ordinamenti degli elementi di V;
- $\pi_{\mathrm{HC}}(G,S_{\mathrm{HC}}(G))=\exists\psi\in S_{\mathrm{HC}}(G):\ \forall 1\le i<n\ [(\psi^{-1}(i),\psi^{-1}(i+1))\in E\ ]\ \wedge\ (\psi^{-1}(n),\psi^{-1}(1))\in E$, ossia, esiste un ordinamento tale che il primo nodo dell'ordinamento è adiacente al secondo, il secondo nodo dell'ordinamento è adiacente al terzo, e così via fino all'ultimo nodo dell'ordinamento che è adiacente al primo: in altre parole, $\pi_{\mathrm{HC}}(G,S_{\mathrm{HC}}(G))=$ `vero` se e soltanto se esiste un ordinamento dei nodi che corrisponde ad un ciclo hamiltoniano in G.

Un certificato per una istanza $x=\langle G=(V,E),\rangle$ di HC è un ordinamento ψ dei nodi di G, ossia, una sequenza $\langle v_1,v_2,\ldots,v_n\rangle$ di $n=|V|$ nodi. Poiché è possibile vericare se un certicato ψ soddisfa il predicato $\rho_{\mathrm{HC}}=\forall 1\le i<n\ [(v_i,v_{i+1})\in E\ ]\ \wedge\ (v_n,v_1)\in E$ in tempo $\mathbf{O}(|E||V|)$, questo prova che $\mathrm{HC}\in\mathbf{NP}$.

Per dimostrare la completezza di HC riduciamo ad esso il problema VC. Mostriamo, dunque, come associare ad una istanza $\langle G=(V,E),k\rangle$ di VERTEX COVER un grafo $f(G,k)=G'=(V',E')$, istanza di CIRCUITO HAMILTONIANO tale che G ammette un vertex cover di k nodi se e soltanto se G' contiene un circuito hamiltoniano. Ricordiamo che, senza perdita di generalità possiamo assumere che G non contenga nodi isolati.

Sia $u\in V$ un nodo di G e sia E_u l'insieme degli archi di G incidenti su u e sia $\langle e_1,e_2,\ldots,e_{|E_u|}\rangle$ un ordinamento dell'insieme E_u. Al nodo u associamo un grafo C_u che consiste di una catena di $6|E_u|$ nodi, ossia, degli archi:

$$(u_{11},u_{12}),(u_{12},u_{13}),\ldots,(u_{15},u_{16}),(u_{16},u_{21}),\ldots,(u_{26},u_{31}),\ldots(u_{|E_u|5},u_{|E_u|6}).$$

Se l'arco $(u,v)\in E$ è l'i-esimo arco incidente su u e il j-esimo arco incidente su v, associamo ad (u,v) un gadget che utilizza i nodi $u_{i1},\ldots,u_{i6}$ e $v_{j1},\ldots,v_{j6}$: tale gadget contiene, oltre alle sottocatene associate al nodo u e al nodo v, i quattro archi (u_{i1},v_{j3}), $(u_{i_3},v_{j1}$, (u_{i4},v_{j6}) e (u_{i6},v_{j4}). Tale gadget è illustrato in Figura 9.4.

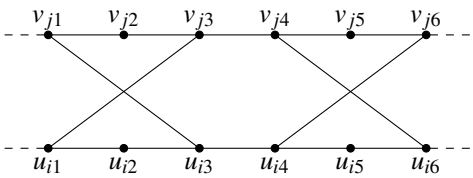

Figura 9.4: Gadget corrispondente all'arco (u,v) nel caso in cui sia stato scelto un ordinamento in cui (u,v) è l'i-esimo arco incidente su u e il j-esimo arco incidente su v.

Infine, all'intero k che compare nell'istanza di VC associamo k nodi $x_1, x_2, \ldots, x_k$ in G': ciascuno di tali nodi è adiacente in G' ai nodi iniziale e finale di ogni catena associata ad un nodo di G, ossia, per ogni $u \in V$ e per ogni $i = 1, \ldots, k$, $(u_{11}, x_1) \in E'$ e $(u_{|E_u|6}, x_i) \in E'$.

Il grafo G è l'unione di tutte le componenti descritte sino ad ora. Nelle Figure 9.5 e 9.6 sono mostrati i grafi G' corrispondenti, rispettivamente, alle istanze $\langle \overline{G} = (\{u,v,z\}, \{(u,v),(v,z)\}), 1\rangle$ e $\langle \hat{G} = (\{u,v,z\}, \{(u,v),(v,z),(z,u)\}), 1\rangle$. Si osservi che $\overline{G}$ ammette un vertex cover costituito da un solo nodo e $\hat{G}$ non ammette un vertex cover costituito da un solo nodo.

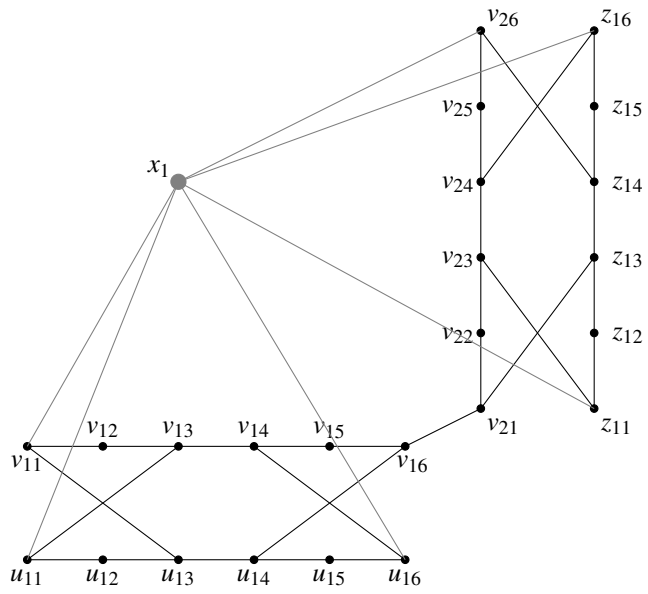

Figura 9.5: Grafo H corrispondente all'istanza $\langle G = (V,E), 1\rangle$ di VC in cui grafo $\overline{G}$ è costituito dalla coppia di archi $(u,v),(v,z)$.

Prima di procedere con la dimostrazione che G ammette un vertex cover di k nodi se e soltanto se esiste un ciclo hamiltoniano in G', osserviamo che, per ogni arco (u,v) in G che sia l'i-esimo arco per u e il j-esimo arco per v, non esiste alcun ciclo hamiltoniano in G' che "entra" nel gadget dal nodo u_{i1} o dal nodo u_{i6} ed "esce" dal gadget attraverso il nodo v_{j1} o attraverso il nodo v_{j6}. Supponiamo, per contraddizione, che un arco esterno al gadget incidente su u_{i1} (ossia, l'arco $(u_{(i-1)6}, u_{i1})$ se $i > 1$, oppure un arco (x_h, u_{i1}) se $i = 1$) e che un arco esterno al gadget incidente su v_{j6} (ossia, l'arco $(v_{j6}, v_{(j+1)1})$ se $j < |E_v|1$, oppure un arco (v_{j6}, x_h) se $j = |E_v|$) facciano entrambi parte di un cammino hamiltoniano in G': se così fosse, chiamiamo y_u il nodo del cammino hamiltoniano esterno al gadget e adiacente a u_{i1}

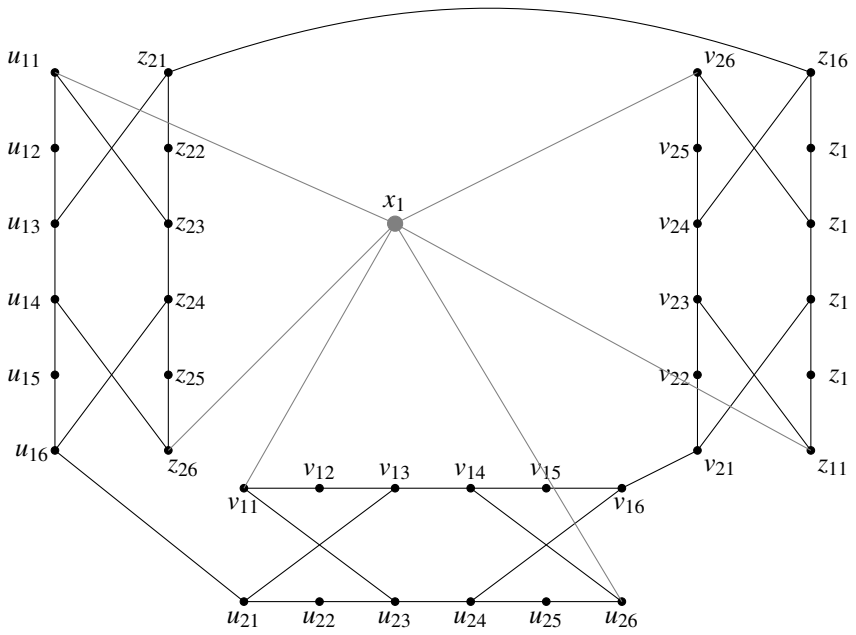

Figura 9.6: Grafo H corrispondente all'istanza $\langle G = (V,E), 1\rangle$ di VC in cui grafo $\hat{G}$ è costituito dal ciclo $(u,v),(v,z),(z,u)$.

e y_v il nodo del cammino hamiltoniano esterno al gadget e adiacente a v_{j6}; dovrebbe, allora, verificarsi uno dei casi seguenti.

- Una porzione del cammino hamiltoniano è costituita dalla catena $y_u, u_{i1}, u_{i2}, u_{i3}, u_{i4}, v_{j6}, y_v$: in questo caso, le due catene $v_{j1}, v_{j2}, v_{j3}, v_{j4}, v_{j5}$ e u_{i5}, u_{i6} sarebbero due "vicoli ciechi", ossia, non potrebbero far parte del cammino hamiltoniano.

- Una porzione del cammino hamiltoniano è costituita dalla catena $y_u, u_{i1}, u_{i2}, u_{i3}, v_{j1}, v_{j2}, v_{j3}, v_{j4}, v_{j5} v_{j6}, y_v$: in questo caso, la catena u_{i4}, u_{i5}, u_{i6} sarebbe un "vicolo cieco', ossia, non potrebbe far parte del cammino hamiltoniano.

- Una porzione del cammino hamiltoniano è costituita dalla catena $y_u, u_{i1}, u_{i2}, u_{i3}, v_{j1}$ (uscendo dal gadget attaverso v_{j1}) e un'altra porzione del cammino hamiltoniano è costituita dalla catena $u_{i6}, u_{i5}, u_{i4} v_{j6}, y_v$ (entrando nel gadget dal nodo u_{i6}): in questo caso, i nodi $v_{j2}, v_{j3}, v_{j4}, v_{j5}$ non sarebbero raggiungibili, ossia, non potrebbero far parte del cammino hamiltoniano.

- Una porzione del cammino hamiltoniano è costituita dalla catena $y_u, u_{i1}, u_{i2}, u_{i3}, v_{j1}$ (uscendo dal gadget attaverso v_{j1}) e un'altra porzione del cammino hamiltoniano è costituita dalla catena $u_{i6}, v_{j4}, v_{j5}, v_{j6}, y_v$ (entrando nel gadget dal nodo u_{i6}): in questo caso, i nodi $v_{j2}, v_{j3}, u_{i4}, u_{i5}$ non sarebbero raggiungibili, ossia, non potrebbero far parte del cammino hamiltoniano.

- Una porzione del cammino hamiltoniano è costituita dalla catena $y_u, u_{i1}, v_{j3}, v_{j2}, v_{j1}$ (uscendo dal gadget attaverso v_{j1}) e un'altra porzione del cammino hamiltoniano è costituita dalla catena $u_{i6}, u_{i5}, u_{i4}, v_{j6}, y_v$ (entrando nel gadget dal nodo u_{i6}): in questo caso, i nodi $u_{i2}, u_{i3}, v_{j4}, v_{j5}$ non sarebbero raggiungibili, ossia, non potrebbero far parte del cammino hamiltoniano.

- Una porzione del cammino hamiltoniano è costituita dalla catena $y_u, u_{i1}, v_{13}, v_{j2}, v_{j1}$ (uscendo dal gadget attaverso v_{j1}) e un'altra porzione del cammino hamiltoniano è costituita dalla catena $u_{i6}, v_{j4}, v_{j5}, v_{j6}, y_v$ (entrando

nel gadget dal nodo u_{i6}): in questo caso, i nodi $u_{i2}, u_{i3}, u_{i4}, u_{i5}$ non sarebbero raggiungibili, ossia, non potrebbero far parte del cammino hamiltoniano.

Ossia, non potrebbe esistere un cammino hamiltoniano, contraddicendo il fatto che le porzioni di gadget indicate fossero parte di un cammino hamiltoniano.
L'impossibilità dei casi rimanenti (entrata nel gadget dal nodo u_{i1} e uscita dal gadget attraverso il nodo v_{j1}, entrata nel gadget dal nodo u_{i6} e uscita dal gadget attraverso il nodo v_{j1}, entrata nel gadget dal nodo u_{i6} e uscita dal gadget attraverso il nodo v_{j6}) può essere provata in maniera analoga.

Come conseguenza della precedente osservazione, possiamo concludere che, per ogni arco (u,v) in G che sia l'i-esimo arco per u e il j-esimo arco per v, e per ogni ciclo hamiltoniano ψ in G',

- se ψ "entra" nel gadget dal nodo u_{i1} allora ψ "esce" dal gadget attraverso il nodo u_{i6},
- se ψ "entra" nel gadget dal nodo v_{j1} allora ψ "esce" dal gadget attraverso il nodo v_{j6}.

Se G ammette un vertex cover di al più k nodi, allora esiste $V' \subseteq V$ tale che $|V'| \leq k$ e, per ogni $(u,v) \in E$, $u \in V' \vee v \in V'$. Osserviamo subito che questo implica che $k \leq |V|$; inoltre, se aggiungiamo a V' $k - |V'|$ elementi di $V - V'$, l'insieme così ottenuto è ancora un vertex cover per G. Allora, senza perdita di generalità, assumiamo che $|V'| = k$.
Indichiamo i nodi di V' con $z^1, \ldots, z^k$ e, per ogni $z^h \in V'$ costruiamo in G' il percorso $p(z^h)$ costituito da $|E_{z^h}|$ tratte, come di seguito spiegato. La tratta i di $p(z^h)$ attraversa il gadget dell'i-esimo arco incidente su z^h: se indichiamo con (z^h, v) tale arco e se esso è il j-esimo arco incidente su v, allora la tratta i di $p(z^h)$ attraversa il gadget di (z^h, v) come illustrato in Figura 9.7 (a) se $v \notin V'$ o in Figura 9.7 (c) se $v \in V'$. Il percorso $p(z_h)$ è poi completato dagli archi $(z^h_{\ell 6}, z^h_{(\ell+1)1})$, per $\ell = 1, \ldots, |E_{z^h}| - 1$, e dall'arco $(z^h_{|E_{z^h}|}, x_h)$.
Consideriamo, ora, l'insieme di tutti i percorsi $p(z^1), \ldots, p(z^k)$ ed aggiungiamo ad esso gli archi

$$(x_1, z^2 11),\ (x_2, z^3 11),\ \ldots,\ (x_{k-1}, z^k 11),\ (x_k, z^1_{11}).$$

In questo modo, abbiamo ottenuto un ciclo hamiltoniano in G'. Infatti, innanzi tutto, il ciclo che abbiamo costruito rispetta i vincoli di cui all'osservazione sopra e, quindi, non lascia vicoli ciechi o nodi irraggiungibili nei gadget che attraversa. Poi, poiché V' è un vertex cover per G, per ogni $(u,v) \in E$, almeno uno dei due estremi di (u,v) è contenuto in V' e, quindi, il gadget di (u,v) è attraversato da almeno uno dei due percorsi $p(u)$ e $p(v)$ in accordo a quanto illustrato in Figura 9.7: in tutti e tre i casi possibili, il ciclo passa attraverso ciascun nodo del gadget una e una sola volta. Infine, per costruzione, anche i nodi $x_1, \ldots, x_k$ sono attraversati dal ciclo che abbiamo costruito una ed una sola volta.

Supponiamo, invece, che G' contenga un ciclo hamiltoniano ψ. In virtù di quanto osservato, tale ciclo deve attraversare il gadget di ciascun arco in accordo a quanto illustrato in Figura 9.7 e, quindi, per ogni arco (u,v) deve contenere un percorso che attraversa tutti i nodi $u_{11}, u_{21}, \ldots u_{|E_u|6}$, oppure un percorso che attraversa tutti i nodi $v_{11}, v_{21}, \ldots v_{|E_v|6}$, o un percorso che attraversa entrambi (dipendentemente da quale caso della Figura 9.7 si applichi). Essendo ψ un ciclo, ciascuno di tali percorsi deve essere collegato, ad entrambi gli estremi, ad un nodo x_i; essendo ψ un ciclo *hamiltoniano*, se uno dei percorsi è collegato ad un suo estremo con il nodo x_i allora dovrà essere collegato, all'altro estremo, a qualche $x_j \neq x_i$. Tutto ciò implica che ψ

- è suddiviso in k sottopercorsi, $\psi_1, \ldots, \psi_k$, tali che ψ_1 inizia con il nodo x_k e termina con il nodo x_1 e, per $1 = 2, \ldots k$, ψ_i inizia con il nodo x_{i-1} e termina con il nodo x_i;
- per ogni $i = 1, \ldots, k$, esiste un nodo $z^i \in V$ tale che il sottopercorso ψ_i attraversa tutti i nodi $z^i_{11}, z_{21}, \ldots z_{|E^i_z|6}$;
- per ogni arco $(u,v) \in E$ esiste $i \in \{1, \ldots, k\}$ tale che $u = z^i$ o $v = z^i$.

Dunque, $V' = \{z^1, \ldots, z^k\}$ è un vertex cover per G.

Resta da dimostrare che la riduzione da $\langle G = (V,E), k \rangle$ a G' richiede tempo polinomiale in $|\langle G = (V,E), k \rangle|$. Costruire il gadget di un arco richiede tempo costante; quindi costruire i gadget di tutti gli archi richiede tempo $\mathbf{O}(|E|)$. Collegare nei gadget degli archi gli elementi che rappresentano lo stesso nodo in V (ossia, un nodo u_{i6} con un nodo $u_{(i+1)6}$, per

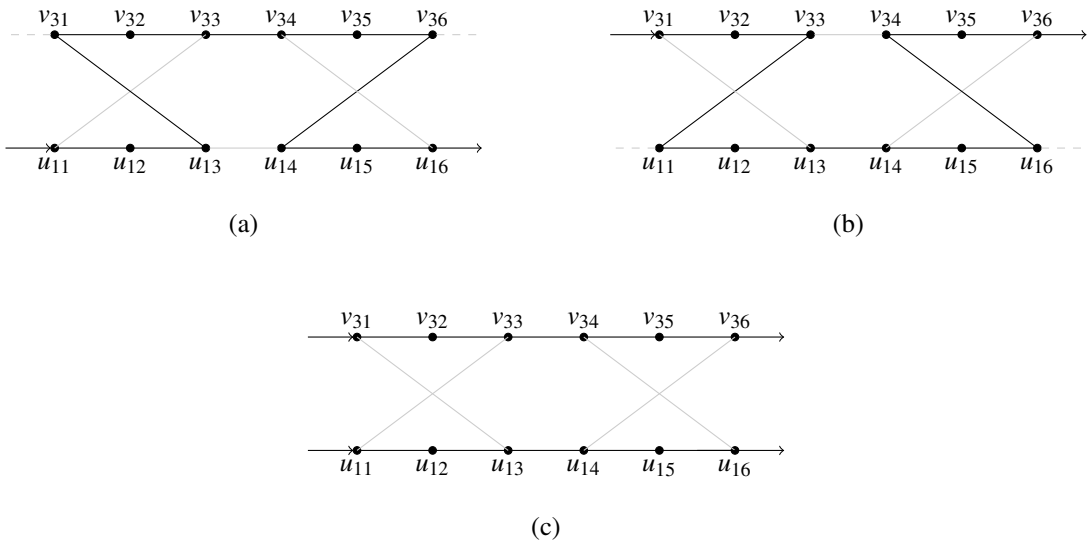

Figura 9.7: I tre modi in cui il gadget corrispondente all'arco (u,v) può essere attraversato da un ciclo hamiltoniano (indicato dagli archi neri): essi cosrrispondono (a) al caso in cui $u \in V'$ e $v \in V - V'$, (b) al caso in cui $u \in V - V'$ e $v \in V'$, (c) al caso in cui $u \in V'$ e $v \in V'$.

ogni $1 \leq i < |E_u|$) richiede tempo lineare nel grado del nodo ($|E_u|$): dunque, complessivamente, tempo in $\mathbf{O}(|V||E|)$. Infine, poiché, come abbiamo osservato, $k \leq |V|$, costruire i nodi $\mathbf{x}_1, \ldots x_k$ e collegarli a ciascuna coppia di nodi $u_{11}, u_{|E_u|6}$, per ogni $u \in V$, richiede tempo $\mathbf{O}(|V|^2$.

Questo completa la dimostrazione che HC è **NP**-completo.

9.5.7 I problemi PERCORSO HAMILTONIANO (HP), LONG PATH (LP) e COMMESSO VIAGGIATORE (TSP)

Dati un grafo non orientato $G = (V,E)$ ed una coppia di nodi $u, v \in V$, un *percorso hamiltoniano* in G fra u e v è un percorso che connette u a v e che passa attraverso ogni altro nodo una ed una sola volta. Il problema PERCORSO HAMILTONIANO (in breve, HP) consiste nel decidere se un dato un grafo non orientato $G = (V,E)$ contiene un percorso hamiltoniano fra una data coppia u e v di suoi nodi.

Il problema PERCORSO HAMILTONIANO può essere formalizzato nella maniera seguente:

- $I_{\mathrm{HP}} = \{\langle G = (V,E), u, v\rangle : G$ è un grafo non orientato $\wedge\ u, v \in V\}$;
- $S_{\mathrm{HP}}(G,u,v) = \{\psi : V \to \{1,\ldots,|V|\} : \psi(1) = u\ \wedge\ \psi(n) = v\}$, ossia, $S_{\mathrm{HC}}(G)$ è l'insieme degli ordinamenti degli elementi di V che hanno u come primo elemento e v come ultimo elemento;
- $\pi_{\mathrm{HP}}(G,u,v,S_{\mathrm{HP}}(G,u,v)) = \exists \psi \in S_{\mathrm{HP}}(G,u,v) :\ \forall 1 \leq i < n\ [(\psi^{-1}(i), \psi^{-1}(i+1)) \in E\ ]$, ossia, esiste un ordinamento tale che il primo nodo dell'ordinamento è adiacente al secondo, il secondo nodo dell'ordinamento è adiacente al terzo, e così via fino all'ultimo nodo dell'ordinamento: in altre parole, $\pi_{\mathrm{HP}}(Gu,v,S_{\mathrm{HP}}(G,u,v)) = \texttt{vero}$ se e soltanto se esiste un ordinamento dei nodi che corrisponde ad un percorso hamiltoniano in G fra u e v.

Un certificato per una istanza $x = \langle G = (V,E), u, v\rangle$ di HP è un ordinamento ψ dei nodi di G, ossia, una sequenza $\langle v_1, v_2, \ldots, v_n\rangle$ di $n = |V|$ nodi, tale che $u = v_1$ e $v = v_n$. Poiché è possibile vericare se un certicato ψ soddisfa il predicato $\rho_{\mathrm{HP}} = \forall 1 \leq i < n\ [(v_i, v_{i+1}) \in E]$ in tempo $\mathbf{O}(|E||V|)$, questo prova che HP $\in$ **NP**.

Osserviamo che il problema HP è molto simile al problema HC. Infatti, per dimostrare la completezza di HP, descriveremo una riduzione da HC.
Poiché un grafo non connesso, banalmente, non contiene cicli hamiltoniani, senza perdita di generalità considereremo nel seguito solo istanze di HC consistenti in grafi connessi. Sia, dunque, $\langle G = (V,e)\rangle$ una istanza di HC tale che G è un grafo connesso e siano $u \in V$ un nodo in G e $x,y \notin V$; la corrispondente istanza di HP è $\langle G' = (V',E'),x,y\langle$, con:

- $V' = V \cup \{x,y\}$,
- $E' = E \cup E_x \; cup E_y$, dove $E_x = \{(x,u)\}$ e $E_y = \{(y,v) : v \in V \ \wedge \ (u,v) \in E\}$.

Dunque, il grafo G' è ottenuto aggiungendo a G due nuovi nodi x ed y, congiungendo x con un (qualsiasi) nodo u di G e congiungendo y a tutti i nodi adiacenti a u in G. Banalmente, costruire G' richiede tempo lineare in $|E|$.

Supponiamo che G contenga un ciclo hamiltoniano $\psi = \{v_1,\ldots,v_n\}$, con $n = |V|$; senza perdita di generalità possiamo supporre che $u = v_1$: infatti, se fosse $u = v_i$, con $i > 1$ potremmo considerare l'ordinamento $\psi' = \{v_i,\ldots v_n,v_1,\ldots,v_{i-1}\}$ che induce anch'esso un circuito hamiltoniano. Allora, G' contiene il seguente percorso hamiltoniano: $x,v_1 = u,\ldots,v_n,y$. Infatti, $(x,v_1) = (x,u) \in E$ per costruzione e, essendo ψ un ciclo hamiltoniano in G, $(v_1,v_n) = (u,v_n) \in E$ e quindi, per definizione dell'insieme E_y, $(v_n,y) \in E_y$.

Viceversa, supponiamo che G' contenga un percorso hamiltoniano $\psi = \{x,v_1,\ldots,v_n,y\}$, con $n = |V|$; allora, poiché è u l'unico nodo adiacente a x in G', deve essere $u = v_1$. Inoltre, poiché (v_n,y) è un arco del percorso hamiltoniano, deve essere $(v_n,y) \in E_y$ e quindi, per definizione dell'insieme E_y, $(u,v_n) = (v_1,v_n) \in E$. Pertanto, $\{v_1,\ldots,v_n\}$ è un ciclo hamiltoniano in G.

Ciò completa la prova che, quella descritta, è una riduzione polinomiale da HC a HP, dimostrando, così, la completezza di HP.

Il problema LONG PATH (in breve, LP) consiste nel decidere, dati un grafo non orientato $G = (V,E)$, una coppia di nodi $u,v \in V$ ed un intero $k \in \mathbb{N}$, se G contiene un percorso da u a v di lunghezza almeno k. Esso può essere formalizzato nella maniera seguente:

- $I_{\mathrm{LP}} = \{\langle G = (V,E),u,v,k\rangle : G \text{ è un grafo non orientato} \wedge u,v \in V \ \wedge \ k \in \mathbb{N}\}$;
- $S_{\mathrm{LP}}(G,u,v,k) = \{p = \langle v_0,v_1,\ldots,v_h\rangle : \forall i = 0,\ldots,h \; [v_i \in V] \ \wedge u = v_0 \ \wedge \ v = v_h\}$, ossia, $S_{\mathrm{LP}}(G)$ è l'insieme delle sequenze di nodi il cui primo elemento è u e il cui ultimo elemento è v;
- $\pi_{\mathrm{LP}}(G,u,v,k,S_{\mathrm{LP}}(G,u,v,k)) = \exists p = \langle v_0,v_1,\ldots,v_h\rangle \in S_{\mathrm{LP}}(G,u,v,k) : \ \forall i = 0,\ldots,h-1 \; [(v_i,v_{i+1}) \in E] \ \wedge \ h \geq k$.

Un certificato per una istanza $x = \langle G = (V,E),u,v,k\rangle$ di LP è una sequenza di nodi $p\langle v_0 = u,v_1,\ldots,v_h = v\rangle$ in G. Poiché è possibile vericare se un certicato $p = \langle v_0 = u,v_1,\ldots,v_h = v\rangle$ soddisfa il predicato $\rho_{\mathrm{LP}} = \forall i = 0,\ldots,h-1 \; [(v_i,v_{i+1}) \in E] \ \wedge \ h \geq k$ in tempo $\mathbf{O}(|E||V|)$, questo prova che LP $\in$ **NP**.

Osserviamo, ora, che le istanze del problema HP sono anche istanze del problema LP in cui $k = n-1$, e questa osservazione è sufficiente a provare la completezza di LP: infatti, la trasformazione di una istanza $\langle G = (V,E),u,v\rangle$ di HP nell'istanza $\langle G = (V,E),u,v,n-1\rangle$ è, banalmente, una riduzione da HP a LP.

La situazione in cui le istanze di un problema Γ_1 sono anche istanze di un problema Γ_2 si esprime formalmente affermando che il problema Γ_1 è una *restrizione* del problema Γ_2. In generale, se la restrizione Γ_1 di un problema Γ_2 è **NP**-completa, allora anche il problema Γ_2 è **NP**-completo.

Sia $G = (V,E)$ un grafo completo e $w : E \to \mathbb{R}^+$ una funzione peso per l'insieme degli archi. Dati un grafo completo $G = (V,E)$, una funzione peso per l'insieme degli archi $w : E \to \mathbb{R}^+$, e $k \in \mathbb{R}^+$, il problema COMMESSO VIAGGIATORE (in breve, TSP, da TRAVELLING SALESMAN PROBLEM) consiste nel verificare se G contiene un ciclo hamiltoniano di peso non maggiore di k, dove il peso del ciclo è definito essere la somma dei pesi degli archi che lo compongono.

Osserviamo esplicitamente che, essendo G un grafo completo, esso contiene $|V|!$ cicli hamiltoniani: uno per ogni possibile permutazione dell'insieme degli archi. La difficoltà, nel problema TSP, consiste nell'individuare un ciclo hamiltoniano *di costo limitato*.
Osserviamo, infine, che un ciclo hamiltoniano è individuato univocamente dall'insieme degli archi che lo compongono; inoltre, è semplice mostrare che un insieme E' di archi è un ciclo hamiltoniano se e soltanto se $|E'| = |V|$ e, per ogni nodo $v \in V$, v è estremo di due archi in E'.

Il problema TSP può essere formalizzato nella maniera seguente:

- $I_{\mathrm{TSP}} = \{\langle G = (V,E), w, k\rangle : G \text{ è un grafo completo non orientato} \wedge w : E \to \mathbb{R}^+ \wedge k \in \mathbb{R}^+\}$;
- $S_{\mathrm{TSP}}(G,w,k) = \{E' \subseteq E\}$;
- $\pi_{\mathrm{TSP}}(G,w,k,S_{\mathrm{TSP}}(G,w,k)) = \exists E' \in S_{\mathrm{TSP}}(G,w,k) : |E'| = |V| \ \wedge \ \forall v \in V \ [\ \exists u,z \in V : (u,v) \in E' \ \wedge \ (v,z) \in E'\ ] \wedge \sum_{e \in E'} w(e) \leq k$.

Un certificato per una istanza $x = \langle G = (V,E), w, k\rangle$ di TSP è un sottoinsieme E' di E. Poiché è possibile vericare se un certicato E' soddisfa il predicato $|E'| = |V| \wedge \forall v \in V\ [\ \exists u,z \in V : (u,v) \in E' \wedge (v,z) \in E'\ ]$ in tempo $\mathbf{O}(|E||V|)$, e poiché è possibile vericare se $\sum_{e \in E'} w(e) \leq k$ in tempo $\mathbf{O}(|E| \log \max\{w(u,v) : (u,v) \in E\})$, questo prova che TSP $\in$ **NP**.

Per dimostrare la completezza di TSP utilizziamo una riduzione da HC.
Sia $\langle G = (V,E)\rangle$ una istanza di HC; ad essa associamo l'istanza $\langle \hat{G} = (V,\hat{E}), w, n\rangle$ di TSP, dove $\hat{E} = \{(u,v) : u,v \in V \wedge u \neq v\}$, $n = |V|$ e, per ogni $(u,v) \in \hat{E}$,

$$w(u,v) = \begin{cases} 1 & \text{se } (u,v) \in E \\ 2n & \text{se } (u,v) \in E \end{cases} . \tag{9.5}$$

Banalmente, calcolare $\langle \hat{G} = (V,\hat{E}), w, n\rangle$ da $\langle G = (V,E)\rangle$ richiede tempo polinomiale in $|G|$.

Se G contiene un ciclo hamiltoniano, tale ciclo in $\hat{G}$ è costituito da soli archi in E il cui peso, in virtù dell'Equazione 9.5, è 1: quindi esso ha peso totale n e, quindi, $\langle \hat{G} = (V,\hat{E}), w, n\rangle$ è una istanza sì di TSP.

Viceversa, se non G contiene alcun ciclo hamiltoniano, allora ogni ciclo hamiltoniano in $\hat{G}$ contiene almeno un arco in $\hat{E} - E$ il cui peso, in virtù dell'Equazione 9.5, è $2n$: quindi esso ha peso totale $\geq 2n + n - 1 > n$ e, quindi, $\langle \hat{G} = (V,\hat{E}), w, n\rangle$ è una istanza no di TSP.

Qusto completa prova che $f(G) = \langle \hat{G}, w, n\rangle$ è una riduzione da HC a TSP e,quindi, che TSP è **NP**-completo.

9.6 Problemi NP-intermedi: il teorema di Ladner

Poiché la classe **P** è chiusa rispetto alla riducibilità polinomiale, allora, nell'ipotesi **P**$\neq$ **NP**, ogni problema **NP**-completo è contenuto in **NP**- **P**. In questa dispensa ci occupiamo di investigare l'esistenza di problemi **NP**-intermedi, ossia, di problemi *non completi* per la classe **NP** e tuttavia appartenenti a **NP**- **P** (sempre nell'ipotesi **P**$\neq$ **NP**).

Teorema 9.4: *Se* ***P*** $\neq$ ***NP*** *allora esiste un linguaggio* $A \in$ ***NP*** *-* ***P*** *che non è* ***NP****-completo.*

Dimostrazione: L'idea della dimostrazione (da una lezione di M. Sudan trascritta da S. Mc Veety) è quella di costruire iterativamente il linguaggio A per mezzo di una riduzione polinomiale ad un linguaggio L **NP**-completo. Tale riduzione dovrà garantire, da un lato, $A \notin$ **P** e, dall'altro, che L non sia riducibile polinomialmente ad A.

A questo scopo, iniziamo con il denotare con $f_1, f_2, \ldots$ una enumerazione delle funzioni in **FP** e con $T_1, T_2, \ldots$ una enumerazione delle macchine di Turing deterministiche che operano in tempo polinomiale.

Definiamo il linguaggio A mediante la descrizione di un algoritmo che decide se una data parola x appartiene o meno ad A. Tale algoritmo opera iterativamente alternando fra due passi: quello che ci proponiamo di ottenere durante l'iterazione j è una *prova* del fatto che $A \neq f_j(L)$, ossia, che f_j non riduce L ad A, e che $A \neq L(T_j)$, ossia, che A non è il linguaggio deciso dalla macchina T_j. La prova dei due fatti avviene individuando due parole, y e z, tali che

- y appartiene ad uno ed un solo linguaggio fra A e $f_j(L)$ (e, quindi, $A \neq f_j(L)$) e

Input:	x.	
Output:	accetta o rigetta.	
Fase 1.	Inizializza il cronometro a 0 e, incrementandolo ad *ogni* istruzione, esegui il seguente ciclo **for** fino a quando esso non avrà raggiunto il valore $\|x\|/2$:	
	$cron \leftarrow 0$;	
	for ($j \leftarrow 1$; $cron \leq \|x\|/2$; $j \leftarrow j+1$) **do begin**	
	passo j.1:	
	$i \leftarrow 1$;	
	incrementando *cron* ad ogni istruzione, esamina una alla volta tutte le parole $y = \emptyset, 0, 1, 00, 01, \ldots$ finché trovi y tale che $(y \in L \wedge f_j(y) \notin A) \vee (y \notin L \wedge f_j(y) \in A)$ oppure hai eseguito $\|x\|/2$ istruzioni;	
	passo j.2:	
	$i \leftarrow 2$;	
	incrementando *cron* ad ogni istruzione, esamina una alla volta tutte le parole $z = \emptyset, 0, 1, 00, 01, \ldots$ finché trovi z tale che $(z \in A \wedge T_j(z)$ rigetta$) \vee (z \notin A \wedge T_j(y)$ accetta$)$ oppure hai eseguito $\|x\|/2$ istruzioni;	
	end	
Fase 2.	Decidi se $x \in A$ o meno:	
	if ($i = j.1$) **then Output**: rigetta;	cioè, $x \notin A$
	else begin	cioè, $x \in A \Leftrightarrow x \in L$
	if ($x \in L$) **then Output**: accetta;	
	else Output: rigetta;	
	end	

Tabella 9.5: Algoritmo non deterministico `A:NPI` che decide $x \in A$.

- z appartiene ad uno ed un solo linguaggio fra A e $L(T_j)$ (e, quindi, $A \neq L(T_j)$).

In conclusione, ogni iterazione j dell'algoritmo sarà suddivisa in due **passi**: durante il **passo j.1** cercheremo una parola y che appartenga ad $L - f_j(A)$ o a $f_j(A) - L$, mentre durante il **passo j.2** cercheremo una parola z che appartenga a $A - L(T_j)$ o a $L(T_j) - A$. Sottolineiamo, ancora, che, in questo modo, nel **passo j.1** escludiamo la possibilità che f_j riduca L ad A, e nel **passo j.2** escludiamo la possibilità che $A = L(T_j)$.

Comunque, individuare le due parole y e z per poter escludere da ogni successiva considerazione f_j (per la riducibilità di L ad A) e T_j (per l'appartenenza a **P** di A) potrebbe risultare troppo costosa in termini di tempo di calcolo. Per questa ragione, utilizzeremo una sorta di time-out in cui cercheremo le due parole y e z solo per una quantità di tempo limitata, scaduta la quale, dipendentemente se la scadenza sarà avvenuta mentre eravamo alla ricerca di y o di z, decideremo se accettare o meno x (ossia, se $x \in A$ o $x \notin A$).

L'algoritmo che, dato x, decide se $x \in A$ è descritto in Tabella 9.5.

Analisi di `A:NPI`.

Osserviamo, innanzi tutto, che $A \in$ **NP**: infatti, per costruzione, la **Fase 1** dell'algoritmo termina (calcolando il valore i) in $|x|/2+1$ passi, mentre la **Fase 2** richiede la decisione circa l'appartenenza di una stringa ad un linguaggio in **NP** ($x \in L$).

Resta da dimostrare che L non è polinomialmente riducibile ad A (e, dunque, A non è **NP**-completo) e che A non è decidibile mediante un algoritmo polinomiale (e, dunque, $A \notin$ **P**). Per provare la verità di queste due affermazioni, dimostriamo che, per ogni $j \in \mathbf{N}$, f_j non è una riduzione polinomiale da L ad A e T_j non decide A ricorrendo ad una doppia diagonalizzazione. Supponiamo allora, per assurdo, che ciò non sia vero, ossia, supponiamo che esista $j \in N$ tale che una delle due seguenti affermazioni è vera:

1) per ogni parola y, $y \in L \Leftrightarrow f_j(y) \in A$

2) per ogni parola z, $z \in A \Leftrightarrow T_j(z)$ accetta.

Sia r il più piccolo indice per cui questo accade e consideriamo separatamente i due casi in cui, per la prima volta con l'indice r, si verifica 1) o 2).

1) Per ogni parola y, $y \in L \Leftrightarrow f_r(y) \in A$ e, per ogni $k < r$, esistono due parole y_k e z_k tali che

   1.1) $(y_k \in L \wedge f_k(y_k) \notin A) \vee (y_k \notin L \wedge f_k(y_k) \in A)$ e

   1.2) $(z_k \in A \wedge T_k(z_k)$ rigetta$) \vee (z_k \notin A \wedge T_k(z_k)$ accetta$)$.

   Indichiamo con $m = \max\{|y_k|, |z_k| : k < r\}$. Osserviamo ora che, per ogni parola x sufficientemente lunga (diciamo, $|x| > 2^{2^m}$), $|x|/2$ istruzioni sono sufficienti per decidere se $w \in L$, se $f_k(w) \in A$, se $w \in A$ e se $T_k(w)$ accetta per tutti i $k < r$ e per tutte le parole w lunghe al più m. Perciò, su input x, l'algoritmo `A:NPI` troverà tutte le parole $y_1, y_2, \ldots, y_{r-1}$ e $z_1, z_2, \ldots, z_{r-1}$ ed inizierà l'iterazione r. In particolare, inizierà l'esecuzione del **passo r.1**: poiché $y \in L \Leftrightarrow f_r(y) \in A$ per ogni parola y, allora il tempo a disposizione ($|x|/2$) terminerà mentre è ancora in esecuzione il **passo r.1** e, quindi, la variabile i assumerà il valore $r.1$ e la parola x sarà rigettata. Ossia, $x \notin A$.
   Questo significa che A contiene solo parole di lunghezza minore della lunghezza di x, ossia, A contiene solo un numero finito di parole: dunque, $A \in \mathbf{P}$. D'altra parte, abbiamo supposto che per ogni parola y, $y \in L \Leftrightarrow f_r(y) \in A$, ossia che $L \leq A$: questo dimostra che anche $L \in \mathbf{P}$. Ma poiché L è un linguaggio **NP**-completo e $\mathbf{P} \neq \mathbf{NP}$ (per ipotesi), la supposizione è assurda.

2) Per ogni parola z, $z \in A \Leftrightarrow T_r(z)$ accetta e inoltre

   2.1) per ogni $k \leq r$ esiste una parola y_k tale che $(y_k \in L \wedge f_k(y_k) \notin A) \vee (y_k \notin L \wedge f_k(y_k) \in A)$ e

   2.2) per ogni $k < r$ esiste una parola z_k tale che $(z_k \in A \wedge T_k(z_k)$ rigetta$) \vee (z_k \notin A \wedge T_k(z_k)$ accetta$)$.

   Indichiamo con $m = \max\{|y_k|, |z_k|, |y_r| : k < r\}$. Osserviamo ora che, per ogni parola x sufficientemente lunga (diciamo, $|x| > 2^{2^m}$), $|x|/2$ istruzioni sono sufficienti per decidere se $w \in L$, se $f_k(w) \in A$ per tutti i $k \leq r$, e se $w \in A$ e se $T_k(w)$ accetta per tutti i $k < r$ e per tutte le parole w lunghe al più m. Perciò, su input x, l'algoritmo `A:NPI` troverà tutte le parole $y_1, y_2, \ldots, y_{r-1}$ e $z_1, z_2, \ldots, z_{r-1}$ ed inizierà l'iterazione r in cui, nel **passo r.1** troverà y_r. Inizierà dunque l'esecuzione del **passo r.2**: poiché $z \in L \Leftrightarrow T_r(z)$ accetta per ogni parola z, allora il tempo a disposizione ($|x|/2$) terminerà mentre è ancora in esecuzione il **passo r.2** e, quindi, la variabile i assumerà il valore $r.2$ e la parola x sarà rigettata se e soltanto se $x \in L$. Ossia, $x \in A \Leftrightarrow x \in L$.
   Questo significa che L ed A differiscono solo per parole di lunghezza minore della lunghezza di x, ossia, A è uguale ad L a meno di un numero finito di parole e, dunque, A ed L appartengono alla stessa classe di complessità. D'altra parte, abbiamo supposto che, per ogni parola z, $z \in A \Leftrightarrow T_r(z)$ accetta, ossia che $A \in \mathbf{P}$: questo dimostra che anche $L \in \mathbf{P}$. Ma poiché L è un linguaggio **NP**-completo e $\mathbf{P} \neq \mathbf{NP}$ (per ipotesi), la supposizione è assurda.

In conclusione, il linguaggio A è contenuto in **NP**- **P** ma il linguaggio **NP**-completo L non è ad esso polinomialmente riducibile, ossia, A non è completo per **NP**. □

9.7 Una visione riassuntiva delle classi P, NP e coNP

È opportuno, a questo punto, risassumere brevemente quello che sappiamo sulle tre classi di complessità **P**, **NP** e co**NP**.

Sappiamo, dal Teorema 6.1 della Dispensa 6, che $\mathbf{P} \subseteq \mathbf{NP}$ e, secondo la congettura fondamentale della Teoria della complessità, è $\mathbf{P} \neq \mathbf{NP}$.
La seconda congettura della Teoria della Complessità Computazionale afferma che co$\mathbf{NP} \neq \mathbf{NP}$. Allora, poiché co$\mathbf{P} = \mathbf{P}$, $\mathbf{P} \subseteq$ co**NP**.

Il Teorema di Cook-Levin ci assicura che, nell'ipotesi $\mathbf{P} \neq \mathbf{NP}$, la classe **NPC** dei problemi **NP**-completi è non vuota e il Teorema di Ladner dimostra che, nella stessa ipotesi, $\mathbf{NP} - (\mathbf{P} \cup \mathbf{NPC}) \neq \emptyset$.
Ancora, assumendo valida la congettura, sappiamo anche (Corollario 6.3 della Dispensa 6) che $\mathbf{NPC} \cap \mathbf{P} = \emptyset$.

Poi, in conseguenza del Teorema 6.21 della Dispensa 6 e del Teorema di Cook-Levin, sappiamo anche che co$\mathbf{NPC} \neq \emptyset$.
Ancora dal Teorema 6.21 e in virtù del Teorema di Ladner sappiamo anche che co$\mathbf{NP} - (\mathbf{P} \cup$ co$\mathbf{NPC}) \neq \emptyset$: infatti, se

Input:	$A = \{a_1, \ldots, a_n\} \subseteq \mathbb{N}$, con $\sum_{i=i}^{n} a_i = 2b$.
Output:	accetta o rigetta.
1	**for** $(i = 1, \ldots, n)$ **do begin**
2	$T[i,0] \leftarrow$ `vero`;
3	**for** $(j = 1, \ldots, b)$ **do**
4	$T[i,j] \leftarrow$ `falso`;
5	**end**
6	$T[1,a_1] \leftarrow$ `vero`;
7	**for** $(i = 2, \ldots, n)$ **do**
8	**for** $(j = 0, \ldots, b)$ **do**
9	**if** $(T[i-1,j] \vee (j \geq a_i \wedge T[i-1, j-a_i]))$ **then**
10	$T[i,j] \leftarrow$ `vero`;
11	**if** $(T[n,b])$ **then Output**: accetta;
12	**else Output**: rigetta.

Tabella 9.6: Algoritmo che decide se $A \in$ PARTIZIONE.

$L \in \mathbf{NP} - (\mathbf{P} \cup \mathbf{NPC})$ allora L^c non può essere co**NP**-completo (in virtù del Teorema 6.21) e non può appartenere a **P** (altrimenti, poiché co$\mathbf{P} = \mathbf{P}$, L dovrebbe appartenere a **P**).

Poiché co$\mathbf{NP} \neq \mathbf{NP}$ e in virtù del Teorema 6.22 della Dispensa 6, co$\mathbf{NPC} \cap \mathbf{NPC} = \emptyset$.

In conclusione, la struttura appena descritta è illustrata in Figura 9.8.

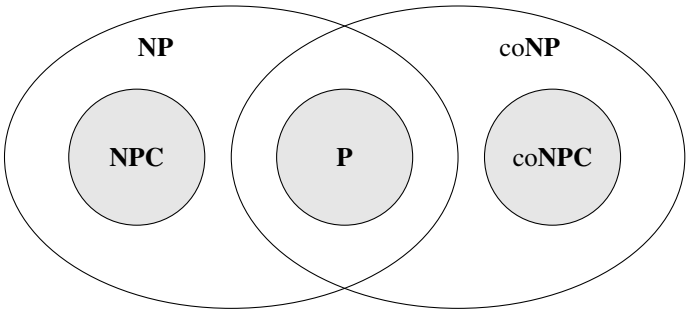

Figura 9.8: Struttura delle classi di complessità **P**, **NP** e co**NP**.

9.8 Problemi numerici e NP-completezza in senso forte

Il problema PARTIZIONE è definito come segue: dato un insieme $A = \{a_1, \ldots, a_n\} \subseteq \mathbb{N}^+$ la somma dei cui elementi sia un numero pari che indicheremo con $2b$, si vuole decidere se esiste un sottoinsieme A' di A tale che

$$\sum_{a \in A'} a = b, \qquad \text{o, equivalentemente,} \qquad \sum_{a \in A'} a = \sum_{a \in A - A'} a.$$

La dimostrazione del seguente teorema è basata su una serie di riduzioni che richiedono tecnicismi piuttosto articolati ed è, pertanto, omessa.

Teorema 9.5: *Il problema* PARTIZIONE *è **NP**-completo.*

Consideriamo, ora, l'agoritmo di programmazione dinamica illustrato in Tabella 9.6 che decide, data una istanza $A = \{a_1, \ldots, a_n\} \subseteq \mathbb{N}$ di PARTIZIONE con $\sum_{i=i}^{n} a_i = 2b$, se essa è una istanza sì.
Esso è basato sulla seguente osservazione: l'insieme A contiene un sottoinsieme A' la somma dei cui elementi è b se e soltanto se si verifica una delle due eventualità seguenti:

- A' non contiene a_n e, dunque, esso è contenuto in $\{a_1,\ldots,a_{n-1}\}$, oppure
- A' contiene a_n e, dunque, $b \geq a_n$ e inoltre $\{a_1,\ldots,a_{n-1}\}$ contiene un sottoinsieme la somma dei cui elementi è pari a $b-a_n$.

Traduciamo, ora, la precedente osservazione in un predicato: definiamo il predicato $T[i,j]$ che è `vero` se e soltanto se l'insieme $\{a_1,\ldots,a_i\}$ contiene un sottoinsieme la somma dei cui elementi è pari a j. Allora, per quanto abbiamo appena osservato,

$$T[i,j] = T[i-1,j] \ \vee \big(j \geq a_i \ \wedge \ T[i-1,j-a_i]\big). \tag{9.6}$$

L'algoritmo in Tabella 9.6 calcola gli elementi della matrice T riga per riga, per valori crescenti di i, utilizzando l'Equazione 9.6. Pertanto, al termine della sua computazione, verrà calcolato il valore $T[n,b]$ che, per definizione, è uguale a `vero` se e soltanto se A è una istanza sì di PARTIZIONE.

Valutiamo, ora, la complessità dell'algoritmo in Tabella 9.6. Poiché gli elementi della riga i-esima della matrice T vengono calcolati solo dopo che sono stati calcolati tutti gli elementi della riga $(i-1)$-esima, per $i = 2,\ldots,n$, la condizione dell'istruzione **if** alla linea 8 viene valutata in tempo costante, così come tempo costante richiedono anche le assegnazioni alle linee 3, 4, 5, e 9. Poiché le istruzioni di assegnazione alle linee 3 e 9 vengono eseguite nb volte, questo dimostra che l'algoritmo ha complessità $\mathbf{O}(nb)$.

Dunque, abbiamo individuato un algoritmo avente complessità $\mathbf{O}(nb)$ per un'istanza di un problema **NP**-completo che consiste di un insieme $A \subseteq \mathbb{N}^2$ di n elementi tali che $\sum_{a\in A} a = 2b$. Questo, ad una occhiata superficiale, sembrerebbe contraddire la congettura $\mathbf{P} \neq \mathbf{NP}$. Cerchiamo, dunque, di analizzre più approfonditamente la questione.
La chiave per comprendere l'assenza di contraddizione è nel concetto di *codifica ragionevole* che abbiamo discusso nella Dispensa 7. Infatti, se codifichiamo ciascuno degli elementi dell'insieme A in unario (ossia, ciascun $a \in A$ come una sequenza di a '1'), allora, detta χ_1 tale codifica, effettivamente l'algoritmo in Tabella 9.6 ha complessità lineare nella dimensione dell'istanza, in quanto $n \leq 2b$ e $|\chi_1(A)| = \Omega(b)$ (ossia, $b = \mathbf{O}(\chi_1(A))$). Il punto è che *la codifica χ_1 non è una codifica ragionevole* in quanto, detta χ_2 una codifica binaria degli elementi di A che codifica ciascun elemento di A con $\log_2(2b)$ caratteri, si ha che $|\chi_2(A)| \in \mathbf{O}(n\log_2 b)$ e, dunque, per tutti gli insiemi A contenenti almeno un elemento a tale che $a \geq 2^n$,

$$|\chi_2(A)| = n^2 \qquad \text{e} \qquad |\chi_1(A)| \geq 2^n \in \Omega(2^{\sqrt{|\chi_2(A)|}}).$$

Questo significa che l'algoritmo in Tabella 9.6 non è un algoritmo (deterministico) polinomiale che decide PARTIZIONE in quanto la dimensione di una istanza A di partizione ha dimensione $|A| \in \mathbf{O}(n\log_2 b)$ (in ogni codifica ragionevole).

D'altra parte, l'esistenza dell'algoritmo in Tabella 9.6 mostra una caratteristica interessante del problema PARTIZIONE. Sia $k \in \mathbb{N}$ un valore costante; indichiamo con k-PARTIZIONE, il problema PARTIZIONE *ristretto* al sottoinsieme delle istanze A tali che, per ogni $a \in A$, si abbia $a \leq |A|^k$. Formalmente, il problema k-PARTIZIONE è descritto dalla tripla seguente:

- $I_{k-\text{PARTIZIONE}} = \{\langle A = \{a_1,\ldots,a_n\}\rangle : \forall\, i = 1,\ldots,n\ [\ a_i \in \mathbb{N} \ \wedge\ a_i \leq n^k\ ] \ \wedge\ \sum_{a_i\in A} a_i = 2b\}$;
- $S_{k-\text{PARTIZIONE}}(A) = \{A' \subset A\}$;
- $\pi_{k-\text{PARTIZIONE}}(A, S_{k-\text{PARTIZIONE}}(A)) = \exists A' \in S_{k-\text{PARTIZIONE}}(A) :\ \sum_{a_i\in A'} a_i = b$.

Allora, per ogni $k \in \mathbb{N}$, se A è una istanza di k-PARTIZIONE, l'algoritmo decide A in tempo (deterministico) polinomiale. Quindi, per ogni *costante* $k \in \mathbb{N}$, $k-$ PARTIZIONE $\in \mathbf{P}$.

Quanto discusso fino ad ora circa il problema PARTIZIONE può essere esteso a tutti i *problemi numerici*.
Un problema si dice *numerico* se

- le sue istanze contengono valori numerici e

- il valore numerico massimo in esse contenuto non è, in generale, limitato superiormente da qualche polinomio nella dimensione della parte non numerica dell'istanza (nel caso di PARTIZIONE, da un polinomio nel numero di elementi di A).

Il problema PARTIZIONE è un esempio di problema numerico.
Ma esistono numerosi esempi di problemi le cui istanze contengono parti numeriche e che, tuttavia, non sono problemi numerici. Ad esempio, le istanze dei problemi VERTEX COVER, INDEPENDENT SET, CLIQUE e DOMINATING SET contengono tutte un valore $k \in \mathbb{N}$, ma tale valore è significativo solo se è non maggiore del numero dei nodi del grafo che costituisce la parte non numerica di ciascuna istanza: ovviamente, non ha senso chiedersi se un grafo di n nodi contiene un vertex cover, o una clique, o un insieme indipendente o dominante di $n+1$ nodi! Quindi, tutti i problemi appena elencati non sono problemi numerici.

Un algoritmo che decide un problema numerico è detto *algoritmo pseudo-polinomiale* se esso richiede tempo polinomiale quando ristretto a sottoinsiemi dell'insieme delle istanze tali che il valore numerico massimo in esse contenuto è limitato da qualche polinomio nella dimensione della parte non numerica dell'istanza (nel caso di PARTIZIONE, da un polinomio nel numero di elementi di A).
L'algoritmo in Tabella 9.6 è un esempio di algoritmo pseudo-polinomiale.

Un problema numerico che è deciso da un algoritmo pseudo-polinomiale viene detto **NP**-completo *in senso debole* (*weak* **NP**-complete).
Viceversa, se un problema numerico non è deciso da alcun algoritmo pseudo-polinomiale, esso viene detto **NP**-completo *in senso forte* (*strong* **NP**-complete).

Il problema TSP (Paragrafo 9.5.7) è un esempio di problema **NP**-completo in senso forte. Per provare questa affermazione, ricordiamo la riduzione da HC a TSP che abbiamo descritto nel Paragrafo 9.5.7 per dimostrarne la completezza: gli archi del grafo completo istanza di TSP risultante dalla riduzione hanno peso 1 oppure $2n$, dove n è il numero di nodi del grafo. Questo significa che la restrizione del problema TSP all'insieme delle istanze tali che il valore numerico massimo in esse contenuto è limitato da qualche polinomio nella dimensione della parte non numerica dell'istanza è un problema **NP**-completo. Dunque, se $\mathbf{P} \neq \mathbf{NP}$, non esiste alcun algoritmo pseudo-polinomiale per TSP.

9.9 Restrizioni degli insiemi delle istanze e il ruolo delle costanti

Nel paragrafo precedente abbiamo visto che il problema PARTIZIONE ristretto all'insieme delle istanze tali che il massimo valore numerico in esse contenuto è limitato superiormente da qualche polinomio nel numero di elementi dell'insieme A è decidibile in tempo polinomiale. Il grado di tale polinomio, però, deve essere *costante*, ossia, *non dipendente dall'istanza*.
Per essere più precisi, questo significa che, ad esempio, 1-PARTIZIONE, o 2-PARTIZIONE, o anche 2158-PARTIZIONE sono contenuti in **P**. Invece, vediamo cosa succede riguardo al problema ALTRA PARTIZIONE (in breve, AP) di seguito formalmente descritto:

- $I_{\mathrm{AP}} = \{\langle A = \{a_1, \ldots, a_n\}, k\rangle : k \in \mathbb{N} \wedge \forall\, i = 1, \ldots, n\ [\ a_i \in \mathbb{N} \wedge a_i \leq n^k\ ] \wedge \sum_{a_i \in A} a_i = 2b\}$;
- $S_{\mathrm{AP}}(A) = \{A' \subset A\}$;
- $\pi_{\mathrm{AP}}(A, S_{\mathrm{AP}}(A)) = \exists A' \in S_{\mathrm{AP}}(A) :\ \sum_{a_i \in A'} a_i = b$.

A prima vista, potrebbe sembrare che i due problemi k-PARTIZIONE e ALTRA PARTIZIONE coincidano. Proviamo, però, a considerare la complessità $\mathbf{O}(nb)$ dell'algoritmo in Tabella 9.6 quando il suo input è una istanza di AP: in questo caso, potrebbe esistere $a \in A$ tale che $a = n^k$ e, in tal caso, sarebbe $b \geq n^k$. Poiché k è un elemento dell'istanza, allora n^k non è un polinomio nella dimensione dell'istanza. Dunque, in generale, l'algoritmo in Tabella 9.6 non è un algoritmo polinomiale per il problema ALTRA PARTIZIONE.

In effetti, il comportamento appena descritto è tipico di molti problemi: anche se, in generale, il problema è **NP**-completo, restringendo l'insieme delle istanze ad un sottoinsieme in cui la parte numerica ha valore costante, potrebbe accadere che la complessità, si riduca. Ad esempio, se un dato grafo $G = (V, E)$ ammette un vertex cover di al più k nodi è decidibile in tempo $\mathbf{O}(|V|^k)$: infatti, è sufficiente generare deterministicamente tutti i sottoinsiemi di cardinalità

k di V e, per ciascuno di essi, verificare se si è ottenuto un vertex cover (e, ricordiamo, tale verifica richiede tempo $\mathbf{O}(|V||E|)$). Pertanto, fissata una costante $k \in \mathbb{N}$, se indichiamo con k-VERTEX COVER il problema di decidere se un dato grafo $G = (V,E)$ ammette un vertex cover di al più k nodi, allora il problema k-VERTEX COVER è contenuto in **P** (mentre, ricordiamo, il problema VERTEX COVER, in cui k è parte dell'input, è **NP**-completo).

Più in generale, può accadere che, non solo limitandosi a considerare istanze in cui i valorici numerici sono costanti, ma anche considerare particolari sottoinsiemi di istanze riduca la complessità del problema. Questo, ad esempio, è ciò che accade con il problema SODDISFACIBILITÀ: quando ci si limita a considerare istanze in cui le clausole sono costituite da due soli letterali il problema diventa 2SAT, che è contenuto in **P**.
Ancora, se ci si restringe a considerare il sottoinsieme delle istanze di VC il cui grafo è completo, allora otteniamo una *restrizione polinomiale* del problema VC, ossia, un problema contenuto in **P**: ogni vertex cover di un grafo completo di n nodi ha cardinalità $\geq n-1$ e, quindi, per decidere se un grafo completo di n nodi ammette un vertex cover di cardinalità k è sufficiente verificare se $k \geq n-1$.

Comunque, non tutti i problemi hanno questo tipo di comportamento e non per tutte le loro restrizioni. Ricordiamo ad, esempio, il problema COLORABILITÀ: anche la restrizione al sottoinsieme delle istanze in cui il numero di colori disponibili è una costante $k \geq 3$, il problema k-COLORABILITÀ rimane **NP**-completo. O, ancora il problema SODDISFACIBILITÀ rimane **NP**-completo quando il sottoinsieme delle istanze considerato è quello in cui le clausole contengono 3 letterali.

In conclusione, quando si considera una restrizione di un problema **NP**-completo, la complessità va studiata *ex novo*: potrebbe trattarsi sia di una restrizione polinomiale che di una restrizione che lascia invariata la completezza del problema.

Concludiamo questa dispensa con una nota su una locuzione che abbiamo utilizzato più volte in questa dispensa. Nella dimostrazione di **NP**-completezza di PERCORSO HAMILTONIANO, abbiamo detto che *senza perdita di generalità* potevamo assumere che il grafo istanza di HC (il problema che abbiamo ridotto a HP) era connesso. Altre volte, parlando del problema VERTEX COVER abbiamo detto che *senza perdita di generalità* potevamo assumere che il vaore k contenuto nell'istanza fosse minore del numero dei nodi del grafo. Ma cosa significa, in questi casi, *senza perdita di generalità*?
Riferiamoci ai due esempi cui abbiamo accennato. Se cerchiamo un ciclo hamiltoniano in un grafo non connesso, sappiamo che non riusciremo a trovarlo. Analogamente, certamente esiste un vertex cover di cardinalità pari al numero dei nodi di un grafo. Questo significa che decidere se una istanza $\langle G = (V,E)\rangle$ di CICLO HAMILTONIANO in cui G è non connesso è una istanza sì richiede tempo polinomiale: si tratta sempre di una istanza no. Analogamente, decidere se una istanza $\langle G = (V,E), k\rangle$ di VERTEX COVER in cui $k \geq |V|$ è una istanza sì richiede tempo polinomiale: si tratta sempre di una istanza sì. In altri termini, il sottoinsieme delle istanze $\langle G = (V,E)\rangle$ di CICLO HAMILTONIANO in cui G è non connesso e il sottoinsieme delle istanze $\langle G = (V,E), k\rangle$ di VERTEX COVER in cui $k \geq |V|$ sono restrizioni polinomiali dei rispettivi problemi **NP**-completi.
Quando utilizziamo un problema **NP**-completo in una riduzione, al fine di provare la completezza di un altro problema, è sufficiente considerare il sottoinsieme delle istanze che non contiene restrizioni polinomiali: infatti, ridurre una restrizione polinomiale di un problema **NP**-completo ad un altro problema è equivalente a ridurre a quest'ultimo un problema in **P**, ossia, non ci fornisce alcuna informazione circa la sua complessità. Per questa ragione, diciamo *senza perdita di generalità*: il nostro risultato ha significato limitandosi ad operare nelle ipotesi indicate *senza perdita di generalità*.