Appartiene per esempio a $\mathcal{P}$ il problema decisionale di stabilire, per tre interi assegnati x, y, z, se il prodotto xy sia maggiore di z. (E' sufficiente calcolare il prodotto xy con un algoritmo polinomiale deterministico, e confrontarlo con z.) Appartiene a $\mathcal{NP}$ il problema decisionale della soddisfattibilità, per cui è stato già fornito un algoritmo polinomiale non deterministico (algoritmo 7.8 nel § 7.2.2).

Ovviamente risulta

$$\mathcal{P} \subseteq \mathcal{NP},$$

poiché gli algoritmi deterministici possono essere visti come caso particolare dei non deterministici. Tuttavia non è mai stato dimostrato se valga propriamente l'inclusione, cioè se $\mathcal{P} \neq \mathcal{NP}$: questo è oggi il più importante quesito irrisolto della teoria della computabilità. Possiamo solo dire che l'ipotesi $\mathcal{P} \neq \mathcal{NP}$ è suffragata sia dalla considerazione che gli algoritmi non deterministici dovrebbero essere "sostanzialmente" più potenti di quelli deterministici, sia dai numerosissimi studi su tanti problemi in $\mathcal{NP}$ per cui non si è mai trovato un algoritmo deterministico polinomiale.

Considerando il comportamento degli algoritmi in termini delle scelte successive da essi compiute, si riconoscerà come nella classe $\mathcal{P}$ si riescano a sfruttare particolari caratteristiche dei problemi per dirigere direttamente (deterministicamente) la computazione sul risultato, ciò che sembra precluso nella classe $\mathcal{NP}$, a meno che non si dimostri $\mathcal{P} = \mathcal{NP}$. Se però si vuole *controllare* che una data successione di scelte $(s_1, s_2, \ldots)$ costituisce una soluzione, è sufficiente ripercorrere tali scelte una a una verificandone gli effetti sul problema, sia che questo appartenga a $\mathcal{P}$ sia che appartenga a $\mathcal{NP}$. Considerando per esempio l'albero di autobus di figura 27, mentre non pare esistere algoritmo polinomiale deterministico per individuare una successione di scelte che conduca al luogo voluto D, si verifica che la successione (5, 15) è una soluzione, semplicemente effettuando le scelte 5 e 15 e controllando il luogo di arrivo.

Ciò suggerisce una definizione alternativa della classe $\mathcal{NP}$, come classe di tutti i problemi per cui la legalità di una soluzione proposta può essere *controllata* in tempo polinomiale da un algoritmo deterministico. Tale proprietà è ovviamente verificata anche per i problemi in $\mathcal{P}$ (per i quali resta ferma la definizione precedente), e implica nuovamente $\mathcal{P} \subseteq \mathcal{NP}$.[1]

Tutti questi argomenti meritano il sostanziale approfondimento che segue.

[1] La proprietà non vale invece per i problemi intrinsecamente esponenziali, che emergono nuovamente come i più complessi.

8.2.1 Riducibilità polinomiale: il teorema di Cook. Il meccanismo fondamentale di indagine tra i problemi della classe $\mathcal{NP}$ è quello della riduzione di un problema P_1 a un problema P_2, mediante un algoritmo che trasformi ogni situazione possibile per P_1 in una equivalente situazione per P_2. Nota la riduzione, un algoritmo per risolvere P_2 può essere usato per risolvere P_1; questo permette di ridurre il problema della soluzione di P_1 a quello, in genere più semplice, della trasformazione di P_1 in un problema P_2 per cui sia noto un algoritmo di soluzione.

Conseguenza generale del meccanismo di riduzione è che potremo definire classi di problemi riducibili l'uno all'altro, e quindi computazionalmente equivalenti se la complessità della riduzione non supera la complessità della soluzione dei singoli problemi. In particolare ci occuperemo di riduzioni eseguibili in tempo polinomiale, ponendo la definizione:

Un problema P_1 si riduce (in tempo polinomiale) a un problema P_2 se ogni soluzione di P_1 può ottenersi deterministicamente in tempo polinomiale da una soluzione di P_2.

In sostanza, la definizione implica l'esistenza di un algoritmo deterministico polinomiale A che trasformi ogni possibile insieme di dati di ingresso D_1 per il problema P_1 in un corrispondente insieme D_2 per il problema P_2, in modo che $P_1(D_1)$ ha risposta affermativa se e solo se $P_2(D_2)$ ha risposta affermativa (ricordiamo che tutti i problemi qui considerati sono decisionali). Così un algoritmo A_2 per risolvere P_2 può essere usato anche per risolvere P_1, se A_2 si applica alla trasformazione dei dati di P_1 ottenuta mediante l'algoritmo A: si esegue cioè $A_2(A(D_1))$. Poiché l'algoritmo A è deterministico polinomiale, l'algoritmo composto da A_2 e A seguirà la complessità di A_2: ne consegue che, se $P_2 \in \mathcal{P}$, allora $P_1 \in \mathcal{P}$; se $P_2 \in \mathcal{NP}$, allora $P_1 \in \mathcal{NP}$, anche se non è escluso che possa trovarsi, per P_1, un algoritmo migliore della composizione di A_2 e A, che confini P_1 in $\mathcal{P}$. Computazionalmente P_2 è "difficile almeno quanto P_1".

Per illustrare il meccanismo di riduzione, ricordiamo anzitutto che un *sottografo* G' di un grafo G è costituito da un sottoinsieme di nodi di G, e da tutti gli archi di G che uniscono nodi di G'; e che un grafo (o sottografo) è *completo* se per ogni coppia di nodi esiste un arco che unisce tali nodi. Per esempio nel grafo (non completo) di figura 30 (§ 7.1.2) esiste il sottografo completo di nodi 3, 4, 6. Enunciamo allora il seguente:

Problema decisionale del sottografo completo (clique)

Dato un grafo G e un intero positivo k, stabilire se G ha un sottografo completo di k nodi.

Si può ora dimostrare che il problema decisionale della soddisfattibilità, enunciato nel paragrafo 7.2.2, si riduce al problema decisionale del sottografo completo. Data una forma normale congiuntiva $F(x_1, ..., x_n)$ si tratta di costruire in tempo polinomiale un grafo G che, per un opportuno valore di k dipendente da F, abbia un sottografo completo di k nodi se e solo se esiste un assegnamento di valori di $x_1, ..., x_n$ che soddisfa la F. Questa dimostrazione sarà ora esposta in dettaglio: chi fosse interessato solo ai risultati può saltare direttamente al teorema successivo.

Poniamo che la F sia composta di k clausole: $F = C_1 \wedge C_2 \wedge ... \wedge C_k$. Ogni apparizione di una variabile x_i nella F, sia essa in forma diretta o negata, è detta un *letterale* di F. Il grafo G avrà un nodo per ogni letterale di F: la coppia (z, j) indicherà il nodo corrispondente alla presenza del letterale z nella clausola C_j. Gli archi di G uniranno le coppie di nodi (z, j), (w, h), con $z \neq \overline{w}$ e $j \neq h$: gli archi indicheranno cioè coppie di letterali appartenenti a clausole diverse ($j \neq h$), che possono avere contemporaneamente valore *vero* ($z \neq \overline{w}$). Seguiamo la costruzione di G sull'esempio di figura 32. La F è costruita su $n = 3$ variabili a, b, c, ed è composta di $k = 3$ clausole. Il corrispondente grafo ha sei nodi, quanti sono i letterali nella F; vi compare, ad esempio, l'arco tra $(a, 1)$ e $(\overline{b}, 2)$; ma non vi compare l'arco tra $(a, 1)$ e $(b, 1)$, perché i nodi sono relativi alla stessa clausola, o l'arco tra $(a, 1)$ e $(\overline{a}, 2)$, perché i nodi corrispondono alla stessa variabile in forma diretta e negata.

Il sottografo completo G' da ricercare in G avrà k nodi, ciascuno relativo a un letterale in una diversa clausola di F (si noti che G' non può avere due nodi relativi alla stessa clausola, poiché non esisterebbe l'arco corrispondente, contro l'ipotesi che G' sia completo). Se G' esiste, l'assegnazione del valore *vero* ai k letterali relativi ai suoi nodi renderà *vero* il valore di tutte le clausole, e quindi della F. Il grafo di figura 32 contiene due sottografi completi relativi ai letterali $a, \overline{b}, \overline{c}$ e $\overline{a}, b, \overline{c}$: le assegnazioni del valore *vero* ai letterali di ciascuna terna (cioè le assegnazioni: $a = vero$, $b = falso$, $c = falso$; e $a = falso$, $b = vero$, $c = falso$; alle variabili delle terne) costituiscono due scelte che soddisfano la F. Dunque la F è soddisfattibile se G ha un sottografo completo di k nodi. Viceversa, se la F è soddisfattibile, deve essere possibile assegnare contemporaneamente il valore *vero* ad almeno un letterale per ogni clausola: G conterrà allora un sottografo completo costruito sui nodi corrispondenti a tali letterali. In conclusione la F è soddisfattibile se e solo se G ha un sottografo completo di k nodi.

Notiamo infine che G può essere costruito in tempo polinomiale da F. Infatti vi sono al massimo $k \cdot n$ letterali in F, e quindi $k \cdot n$ nodi in G. L'esistenza dell'arco tra i nodi generici (z, j) e (w, h) può essere stabilita in tempo costante confrontando z con w, e j con h: poiché vi sono $O((kn)^2)$ possibili archi, l'intera costruzione di G avverrà in tempo di pari ordine.

Abbiamo così compiutamente provato che il problema decisionale della soddisfattibilità si riduce a quello del sottografo completo.

$F = C_1 \wedge C_2 \wedge C_3$
$= (a \vee b) \wedge (\overline{a} \vee \overline{b} \vee c) \wedge (\overline{c})$

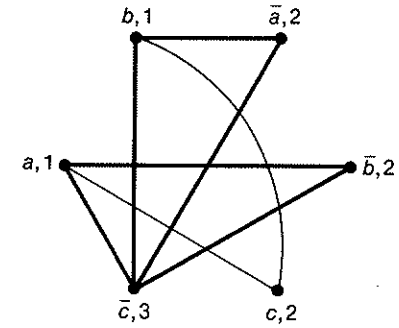

Figura 32
Una forma normale congiuntiva con tre clausole e il grafo corrispondente; i due sottografi completi di tre nodi indicano le scelte: $a = \overline{b} = \overline{c} = vero$, $b = \overline{a} = \overline{c} = vero$, che soddisfano la F.

Possiamo ora affrontare il risultato di maggior importanza oggi noto per i problemi della classe $\mathcal{NP}$. Questo risultato fu presentato da Stephen Cook in un famoso articolo (Cook, 1971), che pose le basi teoriche per tutti i successivi studi.

Teorema *Qualunque problema nella classe $\mathcal{NP}$ si riduce al problema decisionale della soddisfattibilità.*

La dimostrazione del teorema di Cook è piuttosto complessa, e non può essere riportata qui. In sostanza essa mostra come, per qualsiasi algoritmo polinomiale non deterministico A e per qualsiasi insieme di dati di ingresso D, si possa costruire in tempo polinomiale una forma normale congiuntiva F che assume valore *vero* se e solo se la computazione $A(D)$ termina su **success**.

Il problema della soddisfattibilità P_S è dunque, in certo senso, il "più difficile" problema in $\mathcal{NP}$,[2] poiché un algoritmo di soluzione per P_S può essere impiegato per risolvere qualsiasi altro problema $P \in \mathcal{NP}$. Se si

[2] Abbiamo già mostrato che $P_S \in \mathcal{NP}$, poiché esiste l'algoritmo polinomiale non deterministico 7.8 per risolvere P_S.