Capitolo 8

Problemi intrattabili

Siamo giunti all'ultima fase del nostro lavoro, maturi per una discussione critica sulle famiglie di problemi computazionalmente trattabili.

Una prima importante considerazione riguarda la possibilità che un algoritmo non termini la sua computazione in tempo finito. Non abbiamo mai considerato questa eventualità, ammettendo implicitamente che tale fatto sia legato a un errore di progetto dell'algoritmo. Tale "errore" potrebbe però essere non banale, o potrebbe essere intrinsecamente inevitabile.

Consideriamo per esempio un algoritmo di esaurimento per individuare i cicli in un grafo ad *archi orientati* (la definizione è ovvia: vedi fig. 31), che ad ogni passo arricchisca il percorso corrente $(s_1, ..., s_{k-1})$ di un nuovo nodo s_k, se esiste l'arco da s_{k-1} a s_k, e controlli poi se esiste l'arco da s_k a s_1, generando la soluzione $(s_1, ..., s_k)$ se ciò si verifica. Applicato al grafo di figura 31 un tale algoritmo non termina mai la computazione quando, effettuata la prima scelta sul nodo 1, prosegue con la generazione di percorsi (1, 2, 3, 2, 3, ...) senza mai trovare un arco di

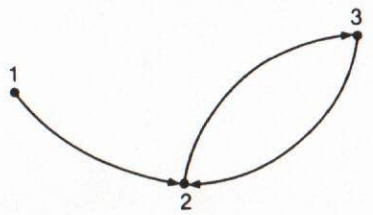

Figura 31
Un grafo ad archi orientati.

ritorno a 1. Lo stesso algoritmo terminerebbe, generando correttamente il ciclo (2, 3), se il grafo fosse privo del nodo 1.

Ovviamente l'algoritmo è mal progettato, poiché si potrebbe per esempio arrestare la computazione quando il numero di nodi del percorso corrente supera il numero di nodi del grafo. Tuttavia esistono procedimenti che possono terminare in tempo finito per alcuni dati di ingresso, e proseguire indefinitamente per altri, senza che ciò possa essere previsto né evitato. Questo accade per esempio nella risoluzione numerica di problemi algebrici per approssimazioni successive, come il calcolo delle radici di una equazione. *Accetteremo come algoritmi solo quelli che terminano sempre in tempo finito*, cercando garanzie specifiche che ciò avvenga (per esempio interromperemo gli algoritmi numerici su soluzioni opportunamente approssimate). La terminazione è però concetto assai più sottile e cercheremo di approfondirlo.

Seguendo una linea ormai consolidata tratteremo essenzialmente problemi decisionali (definiti nel § 7.2.2), mostrando poi come questa limitazione non sia severa.

8.1 Problemi indecidibili

Il primo passo nello studio della intrattabilità è anche il più drastico: ci chiediamo se esistono problemi *indecidibili*, tali cioè che non esista alcun algoritmo (deterministico o non deterministico) per risolverli in tempo finito. A una risposta genericamente affermativa, derivante da alcuni risultati di teoria dei numeri del secolo scorso, Turing fu in grado di far riscontro nel 1936 con la prima enunciazione di un problema indecidibile. Riportato in termini a noi familiari, il problema di Turing può esprimersi nella forma seguente:

Problema decisionale della terminazione

Dato un algoritmo B e i suoi dati D, stabilire se la computazione $B(D)$ termina.

Questo problema è indecidibile, ovvero non esiste alcun algoritmo decisionale A che, accettata una qualsiasi coppia B, D come dato di ingresso, stabilisca *sempre* in tempo finito se $B(D)$ termina o meno. (Si noti che A non può semplicemente consistere nel comandare l'esecuzione $B(D)$ e controllarne il comportamento, poiché, se tale esecuzione non terminasse, A non risponderebbe in tempo finito.)

In modo sommamente schematico l'argomentazione di Turing può essere così espressa.

1. L'algoritmo B è formulato per mezzo di un programma di lunghezza finita: è dunque possibile rappresentare B con un numero intero (che si ottiene per esempio considerando la sequenza di caratteri che costituisce il programma come una sequenza di cifre in una opportuna base di numerazione), cosicché ad algoritmi diversi corrispondono interi diversi. E' anzi possibile, con una semplice trasformazione di codice, porre gli algoritmi in corrispondenza biunivoca con gli interi: diremo che a B corrisponde l'intero $\overline{B}$.

2. Similmente i dati D possono essere messi in corrispondenza biunivoca con gli interi:a D faremo corrispondere l'intero $\overline{D}$. In tal modo l'algoritmo B sarà applicato al parametro intero $\overline{D}$.

3. E' ora lecito chiedere che si esegua la computazione $B(\overline{B})$, interpretando astrattamente la rappresentazione intera di B come parametro di ingresso per B (questo meccanismo di autoriferimento è perfettamente accettato nella logica, anche se può destare qualche sospetto in chi vi si accosti per la prima volta). Indichiamo con $\overline{\overline{B}}$ la questione:

$B(\overline{B})$ termina?

e poniamo $\overline{\overline{B}} = vero$ se la risposta a $\overline{\overline{B}}$ è affermativa, $\overline{\overline{B}} = falso$ altrimenti.

4. Ammettiamo che esista un algoritmo decisionale A per risolvere il problema della terminazione, e applichiamolo alla questione $\overline{\overline{B}}$. Avremo

$$A(\overline{\overline{B}}) \quad \text{termina su} \begin{cases} \textbf{success}, & \text{se} \quad \overline{\overline{B}} = vero, \\ \textbf{failure}, & \text{se} \quad \overline{\overline{B}} = falso. \end{cases}$$

(Si ricordi che per definizione $A(\overline{\overline{B}})$ deve sempre terminare.)

5. Consideriamo il seguente algoritmo C, che accetta come dato di ingresso un altro algoritmo X specificato dall'intero corrispondente $\overline{X}$:

```
procedure C(X̄):
    begin
ALFA: if * A(X̿) termina su success * then goto ALFA
                                     else success
    end;
```

$C(\overline{X})$ termina se e solo se $A(\overline{\overline{X}})$ termina su **failure**. Ovviamente C esiste se e solo se A esiste; in questo caso C sarà un membro della lista di tutti gli algoritmi, ove corrisponderà all'intero $\overline{C}$.

6. Cosa succede ora se poniamo la questione $\bar{\bar{C}}$:

$C(\bar{C})$ termina?

Sostituendo $\bar{C}$ a $\bar{X}$ nella definizione di C, si costata che $\bar{\bar{C}}$=*vero* se e solo se $A(\bar{\bar{C}})$ termina su **failure**. Ma per definizione di A (punto 4) si ha che $A(\bar{\bar{C}})$ termina su **failure** se e solo se $\bar{C}$=*falso*. Questa contraddizione ci obbliga a concludere che A non esiste.

Dai tempi di Turing sono stati scoperti altri problemi indecidibili, costruiti in genere come trasformazione del problema della terminazione. Non si deve però credere che i problemi indecidibili siano pochi, poiché anzi si dimostra che essi costituiscono l'assoluta maggioranza fra tutti i problemi possibili: la nostra scarsa abilità nell'individuarli deve dipendere dal fatto che i problemi che spontaneamente si pongono hanno sempre una soluzione, ed esiste un modo per trovarla.

I problemi indecidibili sono intrattabili senza possibilità di appello, poiché non esiste algoritmo per risolverli. La presente brevissima escursione nel loro mondo ha quindi motivazioni essenzialmente culturali, e non nasce da esigenze di pratica costruzione di algoritmi: la teoria della indecidibilità è una branca della logica e deve essere approfondita sui testi appropriati (vedi per esempio Crossley e altri, 1976).

Vi sono però anche problemi decidibili che devono essere considerati intrattabili, poiché il tempo richiesto a risolverli cresce in modo esponenziale con le dimensioni del problema. Tale costatazione, più volte ripetuta in questo testo, sarà ora sottoposta a un esame accurato.

8.2 Le classi $\mathcal{P}$ e $\mathcal{NP}$

I problemi considerati nei primi sette capitoli sono tutti decidibili, poiché esiste sempre un algoritmo che ne individua la soluzione in tempo finito per ogni possibile valore dei dati di ingresso. Unicamente di problemi decidibili ci occuperemo ancora, tentando di tracciare un confine definitivo tra i problemi che possono essere risolti in tempo polinomiale nella dimensione, e quelli, praticamente intrattabili, che richiedono tempo esponenziale: la conclusione sarà sconcertante, poiché vedremo che, alla luce delle odierne conoscenze, tale confine sostanzialmente non è tracciabile.

Per stabilire la complessità degli algoritmi di soluzione dobbiamo anzitutto precisare il modello di calcolo. Abbiamo utilizzato per questo scopo programmi scritti in un linguaggio di riferimento, presumendo che esistesse un calcolatore pronto ad accettarli e a eseguirne ogni comando in tempo

costante. Ciò è perfettamente chiaro finché non si giunge al modello non deterministico, ove il comando **choice** pone un serio problema sul tempo di esecuzione del programma: se si potesse accettare che la computazione si moltiplichi con grado illimitato di parallelismo, o proceda comunque secondo il modello non deterministico, l'algoritmo avrebbe complessità proporzionale alla profondità h dell'albero delle scelte; poiché però l'unico modo noto di eseguire l'algoritmo è in pratica quello di effettuarne una simulazione deterministica (§ 7.2.2), si dovrà visitare l'intero albero con complessità esponenziale in h. Lo stato attuale delle conoscenze sui problemi intrattabili è in gran parte legato a questo dilemma: per molti problemi si conoscono algoritmi polinomiali solo per il modello non deterministico.

Chiediamoci allora se esistono problemi *intrinsecamente esponenziali*, tali cioè da richiedere tempo sicuramente esponenziale anche se affrontati col modello non deterministico. Si vede immediatamente che appartengono a questa famiglia tutti i problemi che richiedono risultati di lunghezza esponenziale nella dimensione del problema, con il divieto che il risultato sia diviso in pezzi prodotti in parallelo: si può per esempio richiedere che l'algoritmo reciti in sequenza tutte le permutazioni di un insieme. Ma naturalmente problemi così specificati sono sostanzialmente non interessanti, poiché gli si chiede di produrre un risultato che non può essere poi realisticamente esaminato.

Il primo esempio non banale di problema intrinsecamente esponenziale fu scoperto nel 1972, e da allora altri ne sono seguiti, afferenti in genere alla teoria dei linguaggi formali e alla logica. (Per gli opportuni riferimenti vedi il testo di Garey e Johnson, 1979.)

Concludendo, solo i problemi indecidibili e quelli intrinsecamente esponenziali sono dimostratamente intrattabili. Fortunatamente però non molti di questi problemi sono importanti in pratica, mentre si incontrano comunemente problemi decidibili che richiedono tempo esponenziale solo nel modello deterministico: dirigiamo finalmente su questi la nostra attenzione.

Con riferimento ai problemi decisionali, poniamo così due fondamentali definizioni:

> *$\mathcal{P}$ è l'insieme di tutti i problemi decisionali risolubili in tempo polinomiale da un algoritmo deterministico.*

> *$\mathcal{NP}$ è l'insieme di tutti i problemi decisionali risolubili in tempo polinomiale da un algoritmo non deterministico.*

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.

trovasse un algoritmo polinomiale deterministico per risolvere P_S, qualunque problema $P \in \mathcal{NP}$ potrebbe risolversi con un algoritmo polinomiale deterministico: l'effetto (colossale) sarebbe $\mathcal{P} = \mathcal{NP}$! D'altra parte, se si potesse dimostrare che un qualsiasi problema $P \in \mathcal{NP}$ *non può* essere risolto in tempo polinomiale deterministico, allora neanche P_S sarebbe risolubile in pari tempo.

Una volta dimostrata possibile, la riduzione a P_S dei problemi di $\mathcal{NP}$ non deve necessariamente passare attraverso la complessa trasformazione da algoritmi in forme logiche ideata da Cook. Notando infatti che la relazione di riduzione è ovviamente transitiva, si può ridurre P_1 a P_S attraverso una catena di riduzioni da P_1 a P_2, da P_2 a P_3, ..., da P_{i-1} a P_i, da P_i a P_S, ottenute nel modo più libero.

Sulla via delle riduzioni successive si ottengono altri importanti risultati. Abbiamo precedentemente dimostrato che P_S si riduce al problema decisionale del sottografo completo, P_{SC}. Notiamo ora che $P_{SC} \in \mathcal{NP}$, poiché si può facilmente formulare un algoritmo polinomiale non deterministico che, per un dato grafo G, costruisca "in parallelo", in $O(k)$ passi, tutti i sottografi di k nodi e controlli se sono completi: poiché tali sottografi hanno $O(k^2)$ archi possibili, ogni controllo, e quindi l'intero algoritmo, richiede tempo di pari ordine. Dunque abbiamo individuato due problemi in $\mathcal{NP}$ di "pari grado di difficoltà", poiché P_S si riduce a P_{SC} (nostra dimostrazione), e P_{SC} si riduce a P_S (teorema di Cook). Questo fondamentale argomento sarà ora approfondito.

8.2.2 Problemi $\mathcal{NP}$-completi e $\mathcal{NP}$-ardui. Abbiamo visto che i due problemi P_S e P_{SC} appartenenti a $\mathcal{NP}$ sono riducibili ciascuno all'altro, e sono quindi computazionalmente equivalenti; essi sono anche i più difficili problemi in $\mathcal{NP}$, poiché ogni altro problema di questa classe può essere ridotto a P_S e di qui a P_{SC}, e quindi risolto con un algoritmo per P_S o per P_{SC}.

La classe dei problemi di massima difficoltà in $\mathcal{NP}$ non contiene però solo questi due: dai tempi della dimostrazione di Cook è stato scoperto che vi sono tanti altri problemi decisionali computazionalmente equivalenti a P_S e P_{SC}, e sono stati tutti raggruppati sotto il nome di problemi *$\mathcal{NP}$-completi*. La definizione formale è la seguente:

Un problema P è detto $\mathcal{NP}$-completo se $P \in \mathcal{NP}$ e P_S si riduce a P.

Poiché tutti i problemi in $\mathcal{NP}$ sono riducibili a P_S per il teorema di Cook, tutti i problemi $\mathcal{NP}$-completi si riducono scambievolmente l'uno all'altro. Se si scoprisse un algoritmo polinomiale deterministico per la soluzione di un qualsiasi problema $\mathcal{NP}$-completo P, tutti i problemi di $\mathcal{NP}$ potrebbero essere ridotti a P e poi risolti con lo stesso algoritmo. Avremmo dunque: $\mathcal{P} = \mathcal{NP}$.

Per dimostrare che un problema P è $\mathcal{NP}$-completo si deve dimostrare che $P \in \mathcal{NP}$, e che un altro qualsiasi problema già notoriamente $\mathcal{NP}$-completo si riduce a P. Entrambi i passi sono costruttivi: il primo consiste nell'individuare un algoritmo polinomiale non deterministico per risolvere P, compito in genere molto semplice, poiché la grande maggioranza dei problemi ammette spontaneamente una soluzione basata sulle determinazioni in parallelo di tutti i casi possibili, attraverso un albero di scelte di altezza polinomiale nella dimensione del problema. (Fanno eccezione i problemi intrinsecamente esponenziali, di cui, come già detto, si conoscono pochi esempi.) Il secondo passo, cioè la riduzione di un altro problema $\mathcal{NP}$-completo P' a P, richiede esperienza e fantasia, per individuare il problema P' più "simile" a P, e costruire la riduzione di P' a P.

Ovviamente queste dimostrazioni di $\mathcal{NP}$-completezza sono divenute possibili dopo che Cook ebbe scoperto il primo problema $\mathcal{NP}$-completo, cioè P_S, da cui è stato possibile iniziare il procedimento di riduzione ad altri problemi. Non riporteremo altri esempi di riduzione, oltre a quello già visto da P_S a P_{SC}, invitando il lettore interessato ad approfondire l'argomento sul testo di Garey e Johnson; elenchiamo semplicemente alcuni tra i più significativi problemi $\mathcal{NP}$-completi (si noti che si tratta sempre di problemi decisionali, poiché appartengono tutti a $\mathcal{NP}$).

Un piccolo elenco di problemi $\mathcal{NP}$-completi

1. Problema decisionale della *soddisfattibilità* (noto).
2. Problema decisionale del *sottografo completo* (noto).
3. Problema decisionale del *numero cromatico*: dato un grafo G e un intero positivo k, stabilire se G può essere colorato con k colori.
4. Problema decisionale del *commesso viaggiatore*: dato un grafo G ove è assegnato un costo intero positivo ad ogni arco, e un intero positivo k, stabilire se G contiene un ciclo hamiltoniano in cui la somma dei costi degli archi è $\leqslant k$. (Gli archi possono essere orientati o non orientati.)
5. Problema decisionale dello *zaino* (noto: § 7.2.2).
6. Problema decisionale delle *scatole* (*bin packing*): dato un insieme $A = \{a_1, ..., a_n\}$ di interi positivi, e due interi positivi k, s, stabilire se esiste una partizione di A in k sottoinsiemi disgiunti $A_1, ..., A_k$, tale che la somma degli elementi in ogni A_i sia $\leqslant s$.

7. Problema decisionale delle *equazioni diofantine quadratiche*: dati tre interi positivi a, b, c, stabilire se esistono due interi positivi x, y tali che $ax^2 + by = c$. (Il problema generale di stabilire se una equazione a coefficienti interi in un numero arbitrario di variabili ammette una soluzione intera è provatamente indecidibile.)

8. Problema della *non equivalenza tra programmi*: dati un insieme $X = \{x_1, ..., x_n\}$ di variabili e due programmi A_1 e A_2 composti unicamente di frasi di assegnamento della forma

if $x_h = x_k$ **then** $x_i \leftarrow x_r$ **else** $x_i \leftarrow x_s$;

stabilire se esiste una scelta di valori iniziali $\{\bar{x}_1, ..., \bar{x}_n\}$ tale che le computazioni $A_1(\bar{x}_1, ..., \bar{x}_n)$ e $A_2(\bar{x}_1, ..., \bar{x}_n)$ generino valori finali diversi per qualche variabile.

I problemi 1, 2 e 3 sono spontanei modelli matematici di numerosissimi altri problemi; 4 è il capostipite dei problemi di percorsi, nella ricerca operativa; 5 e 6 sono modelli di allocazione di risorse (si pensi alla registrazione di n blocchi di dati su k nastri magnetici di capacità s, come interpretazione del problema delle scatole); 7 è uno degli esempi che interessano l'analisi matematica; 8 è il più semplice problema tratto dalla teoria della programmazione.

Individuata la classe dei problemi $\mathcal{NP}$-completi, possiamo chiederci se vi siano problemi "almeno altrettanto difficili", nel senso che una loro risoluzione in tempo polinomiale deterministico implicherebbe la soluzione in tempo uguale di tutti i problemi $\mathcal{NP}$-completi (e dunque di tutti i problemi in $\mathcal{NP}$). Questi nuovi problemi sono detti *$\mathcal{NP}$-ardui*,[3] e rispondono alla seguente definizione formale:

Un problema P è detto '$\mathcal{NP}$-arduo' se P_S si riduce a P.

La classe dei problemi $\mathcal{NP}$-ardui contiene dunque quella dei problemi $\mathcal{NP}$-completi, poiché non si richiede ai primi di appartenere a $\mathcal{NP}$. La dimostrazione che un problema P è $\mathcal{NP}$-arduo si esegue per riduzione a P di un qualsiasi problema $\mathcal{NP}$-completo o $\mathcal{NP}$-arduo. .

Vi sono essenzialmente due motivi, non mutuamente esclusivi, per cui un problema $\mathcal{NP}$-arduo P può non appartenere a $\mathcal{NP}$. Il primo motivo, aderente allo spirito della definizione, è che P non possa effettivamente essere risolto in tempo polinomiale anche nel modello non deterministico: P è cioè intrinsecamente esponenziale, o indecidibile. Il secondo motivo, assai più banale, è che P non sia un problema decisionale (si ricordi che per definizione la classe $\mathcal{NP}$ contiene solo problemi decisionali). La situazione di fatto è che per molti problemi $\mathcal{NP}$-ardui, siano essi decisionali o meno, non è stato finora possibile dimostrare se esiste un algoritmo polinomiale non deterministico per risolverli.

[3] Nella letteratura in lingua inglese si è sostanzialmente consolidato il termine *$\mathcal{NP}$-hard*, anche se ne permangono alcuni usi diversi. "$\mathcal{NP}$-arduo" è la nostra proposta per un corrispondente termine italiano.

Tipiche dimostrazioni di $\mathcal{NP}$-arduità partono da due formulazioni di un problema: una formulazione *decisionale* P, che indaghi sull'*esistenza* di una soluzione con particolari proprietà, e una formulazione *risolvente* P^r, che chieda di *determinare* una tale soluzione. Chiaramente P si riduce a P^r, perché la determinazione di una soluzione per P^r costituisce prova di esistenza, e quindi soluzione per P. Se P è tra i problemi di cui si è dimostrata la $\mathcal{NP}$-completezza, allora P^r è $\mathcal{NP}$-arduo (non $\mathcal{NP}$-completo, poiché $P^r \notin \mathcal{NP}$, non essendo problema decisionale).

Per esempio, il problema decisionale P_S della soddisfattibilità si riduce al problema P_S^r di *determinare* un insieme di valori delle variabili che rendano *vero* il valore della forma normale congiuntiva. Poiché P_S è $\mathcal{NP}$-completo, *P_S^r è $\mathcal{NP}$-arduo.*

Ancora si può dare di un problema una formulazione di *ottimo* P^o, che cerchi una soluzione che risponde nel miglior modo possibile alle proprietà richieste dalla versione decisionale P. Nuovamente P si riduce a P^o, poiché, determinata una soluzione per P^o, si può controllare immediatamente se essa soddisfa o meno le condizioni poste da P, ottenendo quindi la prova di esistenza da esso richiesta.

Per esempio il problema decisionale del commesso viaggiatore P_{CV} (stabilire se un dato grafo ha un ciclo hamiltoniano di costo $\leqslant k$) ammette il corrispondente problema di ottimizzazione P_{CV}^o, che chiede di determinare il ciclo hamiltoniano di costo minimo. Se si trova questo ciclo, se ne può confrontare il costo c con k, e stabilire in conseguenza se P_{CV} ha soluzione affermativa ($c \leqslant k$) o negativa ($c > k$): P_{CV} si riduce dunque a P_{CV}^o. Poiché P_{CV} è $\mathcal{NP}$-completo, *P_{CV}^o è $\mathcal{NP}$-arduo*.

Similmente i problemi $\mathcal{NP}$-completi indicati ai punti 2, 3, 5 e 6 nell'elenco dato in precedenza ammettono corrispondenti problemi $\mathcal{NP}$-ardui nella forma P^r o P^o, la cui formulazione è lasciata al lettore.

8.2.3 Qualche approfondimento: $\mathcal{NP}$-equivalenza e complementazione. La discussione del paragrafo precedente può lasciarci non completamente soddisfatti, perché non indica se i problemi $\mathcal{NP}$-ardui fin qui tro-