6. NP Completezza e problemi “difficili”

Tutti gli algoritmi che abbiamo visto finora sono *algoritmi con complessità polinomiale*: se la dimensione dell'input è n il loro tempo di esecuzione è $O(n^k)$ per qualche costante k. Avere complessità polinomiale è un requisito minimo affinché un algoritmo possa essere considerato “efficiente”. Un algoritmo con complessità più che polinomiale (p.es. $O(2^n)$ o $O(n!)$) è un algoritmo non efficiente perché non è scalabile: può risolvere in tempi ragionevoli solamente istanze piccole del problema. I ricercatori e professionisti dell'informatica si sono resi conto piuttosto presto che non tutti i problemi possono essere risolti da un algoritmo polinomiale. In generale, i problemi per i quali esiste una soluzione polinomiale vengono considerati *trattabili*, mentre quelli che richiedono un algoritmo più che polinomiale sono detti *intrattabili* o anche *difficili*. Sappiamo che ci sono problemi che non possono essere risolti da nessun algoritmo, indipendentemente dalla sua complessità, come il ben noto “Halting Problem” di Turing. E' anche possibile dimostrare che ci sono problemi che richiedono un numero esponenziale di passi per essere risolti, come il problema della Torre di Hanoi. Tuttavia, stabilire con precisione qual'è il confine tra problemi trattabili ed intrattabili è piuttosto difficile. Esiste infatti una classe di problemi, detti “NP-completi”, per i quali molti ritengono che non esista un algoritmo polinomiale per risolverli, ma per i quali nessuno è riuscito a dimostrare formalmente un lower-bound più che polinomiale per la loro complessità.

Lo studio di questi problemi è particolarmente interessante perché molti problemi NP-completi sembrano molto simili a problemi che sappiamo risolvere in tempo polinomiale. Per ognuna delle seguenti coppie di problemi, uno dei componenti è risolvibile in tempo polinomiale mentre l'altro è NP-completo:

Cammini minimi e cammini massimi. Nel capitolo precedente abbiamo visto che trovare il cammino *minimo* in un grafo orientato si può risolvere in tempo polinomiale. Trovare un cammino semplice di lunghezza *massima* è invece un problema difficile.

Cammino Euleriano e Circuito Hamiltoniano. Dato un grafo $G = (V, E)$, un *cammino Euleriano* è un cammino che attraversa tutti gli *archi* del grafo esattamente una volta, potenzialmente passando più volte per lo stesso vertice. Un *circuito Hamiltoniano* è un ciclo semplice che attraversa tutti i *vertici* del grafo esattamente una volta (senza necessariamente attraversare

tutti gli archi) . Trovare un circuito Euleriano in un grafo è diverso dal trovare un cammino Hamiltoniano.

Colorazione dei vertici di un grafo. "Colorare" un grafo significa assegnare etichette, tradizionalmente chiamate "colori", agli elementi di un grafo soggetta a determinati vincoli. Dato un certo numero di colori k, il problema della *colorazione dei vertici* di un grafo non orientato $G = (V, E)$ richiede di trovare un modo di colorare i vertici del grafo tale che nessuna coppia di vertici adiacenti condivida lo stesso colore. La complessità del problema dipende dal *numero di colori* che si possono utilizzare: in particolare cambia quando si passa da due colori a tre colori.

Esercizio 6.1 Per ognuna delle coppie di problemi qui sopra, dire quale si può affrontare in tempo polinomiale e scrivere un algoritmo efficiente che lo risolva. ■

6.1 Problemi P, NP, coNP e NP-completi

Per classificare in maniera più precisa le complessità dei problemi è necessario introdurre qualche definizione. Come prima cosa, diciamo che un *problema di decisione* è un problema che ha come output un singolo valore booleano: SI/NO. Quindi definiamo le seguenti tre classi di problemi di decisione:

- P è la classe dei problemi di decisione che si possono risolvere in tempo polinomiale. Più precisamente, dei problemi che possono essere risolti in tempo $O(n^k)$ per qualche costante k, e dove n è la dimensione dell'input. Come abbiamo già visto, la classe P è la classe dei problemi trattabili.
- NP è la classe dei problemi caratterizzati dalla seguente proprietà: quando la risposta è SI, allora esiste un *certificato* di questo fatto che può essere *verificato* in tempo polinomiale. Intuitivamente, NP è la classe dei problemi dove possiamo controllare velocemente se la risposta è SI se qualcuno ci fornisce la soluzione.
- coNP è sostanzialmente l'opposto di NP. Se la risposta è NO, allora esiste un *certificato* di questo fatto che può essere *verificato* in tempo polinomiale.

Tutti i problemi che abbiamo visto sopra possono essere verificati in tempo polinomiale:

Cammino Euleriano e Circuito Hamiltoniano. Il certificato in questo caso è costituito da una sequenza di vertici $v_1, \ldots, v_n$. È piuttosto semplice verificare in tempo polinomiale che la sequenza rappresenta un cammino oppure un ciclo con le proprietà richieste (visita tutti gli archi o tutti i vertici una sola volta). La complessità della verifica è $O(|E|)$ per un cammino Euleriano e $O(|V|)$ per un circuito Hamiltoniano.

Colorazione dei vertici di un grafo. Il certificato è l'assegnazione dei colori $color[v]$ ai vertici. Per verificare se è corretto basta iterare tra gli archi del grafo e controllare che nessuna coppia di vertici adiacenti condivida lo stesso colore. La complessità della verifica è quindi $O(|E|)$.

Cammini minimi e cammini massimi. Trovare un cammino minimo o massimo è un *problema di ottimizzazione* che non ammette soluzioni SI/NO. Possiamo trasformarlo in un problema di decisione riformulandolo nel seguente modo: "dato un grafo $G = (V, E)$, un nodo sorgente s, un nodo destinazione d e un valore k, esiste un cammino semplice da s a d di peso inferiore/superiore a k?" Il certificato in questo caso è costituito da un cammino dalla sorgente alla destinazione: la verifica si effettua semplicemente controllando che il peso del cammino rispetti il vincolo rispetto al valore k.

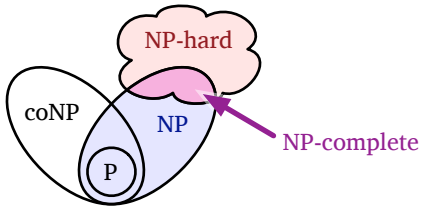

Figura 6.1: Come *pensiamo* sia fatto il mondo.

Le considerazioni qui sopra ci mostrano immediatamente che tutti i problemi considerati appartengono alla classe NP. Alcuni di essi (ma non tutti) si possono risolvere in tempo polinomiale, e quindi appartengono anche alla classe P. Effettivamente, è facile vedere che ogni problema in P è anche in NP: possiamo verificare che la risposta SI è corretta usando direttamente l'algoritmo polinomiale che li risolve, senza bisogno del certificato! Di conseguenza, $\mathrm{P} \subseteq \mathrm{NP}$ e, simmetricamente, anche $\mathrm{P} \subseteq \mathrm{coNP}$.

Il problema aperto più importante di tutta l'informatica—ed uno tra i più importanti di tutta la *scienza*—è quello di stabilire se le classi P e NP siano realmente diverse oppure no. Il Clay Mathematical Institute di Cambridge ha posto il problema di P contro NP come il primo dei suoi "Problemi del Millennio" e offre un premio di un milione di dollari a chiunque fornisca la soluzione. Un altro problema aperto più sottile è quello di stabilire se le classi NP e coNP siano diverse. Anche in questo caso, riuscire a verificare un certificato per la risposta SI in modo efficiente non implica necessariamente che possiamo fare la stessa cosa per il NO. In genere si ritiene che $\mathrm{NP} \neq \mathrm{coNP}$, ma anche in questo caso nessuno è ancora riuscito a dimostrarlo formalmente.

Un certo problema Π è detto NP-*hard* se l'esistenza di un algoritmo polinomiale per risolverlo implica l'esistenza di un algoritmo polinomiale *per ogni problema in* NP. Detto in altro modo:

$$\Pi \text{ è NP-hard} \iff \text{Se } \Pi \text{ si può risolvere in tempo polinomiale, allora } \mathrm{P} = \mathrm{NP}$$

Intuitivamente, se siamo in grado di risolvere un problema NP-hard in modo efficiente, allora possiamo risolvere in modo efficiente *ogni problema* di cui possiamo verificare facilmente una soluzione, usando la soluzione del problema NP-hard come sottoprocedura.

Un problema è NP-*completo* se è sia NP-hard che appartenente alla classe NP (o "NP-easy"). I problemi NP-completi sono i problemi più difficili della classe NP. Trovare un algoritmo polinomiale per risolvere *uno qualsiasi* dei problemi NP-completi vuol dire trovare un algoritmo polinomiale per *ogni problema* NP. Esistono *migliaia* di problemi di cui è stato dimostrata la NP-completezza, e di conseguenza l'esistenza di un algoritmo polinomiale per uno qualsiasi (e quindi per tutti) di essi è ritenuta estremamente improbabile. Tuttavia, poiché nessuno è finora riuscito a dimostrare il contrario, non possiamo eliminare del tutto la possibilità che i problemi NP-completi siano in realtà risolvibili in modo efficiente.

La Figura 6.1 riassume la situazione delle classi di complessità viste in questa sezione. Progettare ed implementare algoritmi richiede la conoscenza dei principi di base della teoria della NP-completezza. Infatti, stabilire che un problema è NP-completo costituisce una prova piuttosto forte della sua intrattabilità. Di conseguenza, conviene forse cercare di risolvere il problema con un algoritmo di approssimazione, oppure identificare un caso particolare trattabile, piuttosto che cercare un algoritmo efficiente per il caso generale che forse non esiste nemmeno. Inoltre, ci sono molti problemi a prima vista del tutto analoghi ad altri problemi che sappiamo risolvere in modo efficiente che sono in realtà NP-completi. Quindi è importante acquisire familiarità con questa importante classe di problemi anche per chi non si occupa degli aspetti teorici dell'informatica.

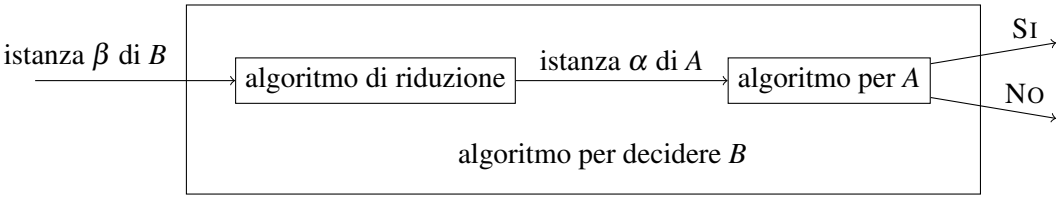

Figura 6.2: Usare un algoritmo di riduzione per risolvere B dato un algoritmo per A.

6.2 Come dimostrare che un problema è NP-completo

Ogni dimostrazione di NP-completezza di compone di due parti:

- una prima parte dove si dimostra che il problema che siamo considerando appartiene alla classe NP;
- una seconda parte dove si dimostra che il problema è NP-hard.

Dimostrare che un problema è in NP vuol dire dimostrare che esiste un algoritmo polinomiale per verificare un certificato per il SI. Per la prima parte si procede quindi in modo simile a quanto fatto finora per analizzare algoritmi. Le tecniche che si usano per dimostrare che un problema è NP-hard sono fondamentalmente diverse: in questo caso stiamo affermando che quel problema è difficile da risolvere, non che è facile da risolvere. Non stiamo dimostrando l'esistenza di un algoritmo efficiente, ma che nessun algoritmo efficiente è probabile che esista. Faremo affidamento su due concetti fondamentali per dimostrare che un problema è NP-hard:

Riduzioni tra problemi diversi.

Per dimostrare che un certo problema è NP-hard si procede tipicamente con una *dimostrazione per riduzione*. *Ridurre* un problema A ad un altro problema B significa descrivere un algoritmo che risolve il problema A sotto l'assunzione che esista un algoritmo per risolvere il problema B. Fare riduzioni di problemi è una tecnica molto diffusa nell'informatica, anche se spesso viene chiamata con in nome diverso come “scrivere subroutine”, “funzioni ausiliarie” o “programmazione modulare”. Dimostrare che un problema è NP-hard significa trasformare un problema in un altro, andando però nella direzione opposta a quella che viene utilizzata normalmente:

Per dimostrare che il problema A è NP-hard dobbiamo ridurre un problema NP-hard ad A.

Quindi, per dimostrare che il problema che stiamo analizzando è difficile da risolvere dobbiamo descrivere un algoritmo per risolvere un problema *diverso*, che sappiamo essere difficile, utilizzando un ipotetico algoritmo per il primo problema come sottoprocedura. Il ragionamento procede essenzialmente per assurdo: l'esistenza di una riduzione implica che se il problema che stiamo analizzando è facile da risolvere, allora anche l'altro problema sarebbe facile da risolvere. Ma questo è un assurdo perché sappiamo che l'altro problema è difficile da risolvere, quindi anche il primo problema deve esserlo. La Figura 6.2 mostra lo schema di una riduzione di questo tipo.

Un primo problema NP-completo

Le tecniche di riduzione richiedono l'esistenza di un problema che sappiamo essere NP-hard per dimostrare che un altro problema è anch'esso NP-hard. Abbiamo quindi bisogno di un “primo problema” NP-hard per poter individuare tutti gli altri. Il problema che utilizzeremo è quello della *soddisfaciblità Booleana*, che chiameremo semplicemente **SAT**. L'input di **SAT** è una formula Booleana come

$$(a \vee b \vee c \vee \overline{d}) \leftrightarrow ((b \wedge \overline{c}) \vee \overline{(\overline{a} \rightarrow d)} \vee (a \wedge b)),$$

ed il problema chiede di determinare se è possibile assegnare dei valori booleani (VERO/FALSO) alle variabili $a, b, c, \ldots$, in modo che il valore di verità della formula sia VERO.

Il fatto che questo semplice problema è NP-completo è stato dimostrato da Steve Cook nel 1971, ed in maniera indipendente da Leonid Levin nel 1973. Poiché queste note sono (volutamente) vaghe sulle definizioni precise, ci limiteremo ad enunciarlo senza dimostrazione.

Teorema 6.2.1 — Cook-Levin. Il problema della soddisfacibilità Booleana è NP-completo.