Appunti introduttivi sulle classi di complessità

A. Agnetis*

1 Introduzione

Lo scopo di queste note è quello di fornire una introduzione ad alcuni concetti di complessità computazionale che, nati in ambito strettamente informatico, hanno acquisito grande rilevanza nell'ambito della ricerca operativa. Spesso si usa dire che un determinato problema è *facile* se si riesce a risolverlo in modo efficiente. Per contro, appare naturale considerare *difficile* un problema per cui questo non sia possibile. Gli sviluppi della teoria della complessità computazionale che si sono avuti a partire dai primi anni '70, hanno consentito di definire questi concetti in modo preciso, e hanno fornito gli strumenti formali per una loro utilizzazione.

2 Problemi di ottimizzazione e problemi di riconoscimento

Com'è noto, un *problema* (del tipo di quelli studiati in questo corso) può essere definito come un *insieme di istanze*, e un'istanza è un *esempio specifico* di un particolare problema. Le istanze stanno ai problemi un po' come gli individui stanno alle specie animali. Ossia, possiamo definire la specie CANE come l'insieme di tutti i cani del mondo, e l'individuo Pongo come un'istanza della specie CANE. Cosi', quando parleremo di un particolare problema decisionale, dobbiamo fare riferimento a un'astrazione precisa, nel cui formato poi rientreranno tutte le istanze di quel problema. Ad esempio, possiamo definire il problema CAMMINO MINIMO nel modo seguente:

CAMMINO MINIMO (FORMA DI OTTIMIZZAZIONE)

- *Dati:* Un grafo $G = (N, A, w)$, orientato, pesato sugli archi con valori non negativi; due nodi s e t

*Dipartimento di Ingegneria dell'Informazione - Università di Siena

- *Domanda:* Quanto vale la lunghezza del cammino minimo?

Un'istanza di CAMMINO MINIMO è dunque descritta completamente da un grafo, dai suoi pesi, e dalla indicazione di due suoi nodi. L'output di un algoritmo che risolva tale problema sarà costituito dal *valore* della lunghezza minima, z^*. Questo problema è nella (consueta) forma di ottimizzazione. Per le nostre considerazioni, conviene introdurre il concetto di problema *in forma di riconoscimento* (alcuni autori usano il termine *in forma di decisione*, a mio avviso meno appropriato). Un problema è nella forma di riconoscimento, se è posto in una forma in cui la risposta può essere solo SI o NO. Ad esempio, in forma di riconoscimento il problema CAMMINO MINIMO può esprimersi nel seguente modo:

CAMMINO MINIMO (FORMA DI RICONOSCIMENTO)

- *Dati:* Un grafo $G = (N, A, w)$, orientato, pesato sugli archi con valori non negativi; due nodi s e t; un intero positivo k
- *Domanda:* Esiste un cammino di lunghezza $\leq k$?

Se un'istanza di un problema di riconoscimento ammette risposta SI, diremo che quella è un'istanza *affermativa*; in caso contrario, diremo che è un'istanza *negativa*.

A prima vista, le due versioni dello stesso problema possono sembrare non equivalenti. Se conosciamo il valore z^* della lunghezza del cammino minimo, è evidente che sapremo immediatamente rispondere al problema in forma di riconoscimento, in quanto, se $k \geq z^*$, la risposta sarà SI, e altrimenti sarà NO.

Ma è vero anche il viceversa. Infatti, supponiamo di saper risolvere il problema in forma di riconoscimento, e vogliamo la risposta a quello in forma di ottimizzazione. Consideriamo allora un upper bound U al valore della soluzione ottima (tale upper bound può essere estremamente banale, ad esempio se stiamo cercando il cammino minimo su un grafo possiamo scegliere come U la somma dei pesi di tutti gli archi del grafo). Si può dunque inizialmente risolvere il problema in forma di riconoscimento impostando k al valore $U/2$. Se la risposta è SI, vuol dire che il percorso minimo ha lunghezza non superiore a $U/2$, e dunque torneremo a risolvere un problema di riconoscimento con $k = U/4$. Se la risposta è NO, al passo successivo imposteremo invece $k = (3/4)U$. E cosi' via, sempre dimezzando l'intervallo di valori entro i quali può cadere z^*. Dopo al più un numero di passi pari a $log_2 U$, avremo individuato il valore ottimo z^*[1]. Dunque, se disponiamo

[1]Si fa qui l'ipotesi che tutti i numeri che compaiono nella definizione dell'istanza (in questo caso, le lunghezze degli archi del grafo) siano interi o razionali. Questo è abbastanza poco restrittivo dal punto di vista della soluzione di problemi reali.

di una procedura per la soluzione del problema in forma di riconoscimento, possiamo, attraverso una semplice ricerca binaria, pervenire alla soluzione del problema di ottimizzazione. Quel che è importante, è che se si sa risolvere il problema di riconoscimento in modo *efficiente*, con la stessa efficienza sarà possibile risolvere quello in forma di ottimizzazione. Per questo motivo, di qui in avanti *faremo sempre riferimento a problemi in forma di riconoscimento.*

Osserviamo che mentre molti problemi sono posti "naturalmentè' in forma di ottimizzazione (come CAMMINO MINIMO, ad esempio, ma anche MASSIMO FLUSSO e tanti altri), altri problemi sono inerentemente in forma di riconoscimento. Ad esempio:

CICLO HAMILTONIANO

- *Dati:* Un grafo $G = (N, A)$, non orientato
- *Domanda:* Esiste un ciclo hamiltoniano?

Dunque il fatto di riferirsi a problemi in forma di riconoscimento consente di trattare in modo unificato problemi 'intrinsecamentè di riconoscimento e problemi che, nella loro formulazione più naturale (ovvero, quella che viene dalla modellazione di un problema reale) sono invece in forma di ottimizzazione.

3 La classe P

Lo scopo della nostra analisi è quello di arrivare a definire concetti precisi per quanto riguarda la *difficoltà intrinseca* dei problemi. Iniziamo allora dai problemi più semplici.

Definizione 1 *(Classe P). Un problema* A *appartiene alla classe P se esiste un algoritmo di soluzione la cui complessità è polinomiale nelle dimensioni dell'input.*

Più sinteticamente, un problema si dice in P se può essere risolto in tempo polinomiale. Un problema appartenente alla classe P si dirà *polinomiale.* Conosciamo quindi già molti rappresentanti di questa classe. Oltre a CAMMINO MINIMO, troviamo ad esempio

CONNESSIONE

- *Dati:* Un grafo $G = (N, A)$, non orientato
- *Domanda:* E' connesso?

MATCHING BIPARTITO

- *Dati:* Un grafo bipartito $G = (U, V, A)$, non orientato; un intero k
- *Domanda:* Esiste un matching con almeno k archi?

MASSIMO FLUSSO

- *Dati:* Un grafo $H = (N, A, c)$, orientato, pesato sugli archi (capacità); due nodi s, t; un intero h
- *Domanda:* Esiste un flusso ammissibile da s a t di valore almeno h?

Si noti un fatto molto importante: noi stiamo caratterizzando un problema per mezzo dell'esistenza di un algoritmo per la sua soluzione. Nel momento in cui, preso un nuovo problema, riusciamo a trovare un algoritmo polinomiale per la sua soluzione, possiamo ascrivere a buon diritto questo problema alla classe P.

Dunque, l'appartenenza a P è sinonimo, per noi, di "problema facilé'. Questo di per sé può lasciare perplesso qualcuno – e per la verità, senza avere tutti i torti. Infatti, se si definisse un problema, e si trovasse poi per questo problema un algoritmo risolutivo di complessità $O(n^{15})$, in base alla nostra definizione tale problema apparterrebbe a P, ma potrebbe davvero essere considerato facile? La risposta a questa obiezione di principio viene in realtà dalla pratica. Infatti, per la stragrande maggioranza dei problemi polinomiali studiati e di interesse pratico, il grado del polinomio che esprime la complessità del migliore algoritmo risolutivo noto, è sufficientemente basso da consentire di identificare "facilità" con "polinomialità". L'esperienza mostra inoltre che, una volta trovato un algoritmo polinomiale per un problema, e quindi appurata la sua collocazione in P, spesso si riescono a individuare algoritmi più raffinati di complessità inferiore. In definitiva, se comunemente si identificano i problemi facili con quelli della classe P è perché questo è verificato nella maggior parte dei casi pratici.

Un'altra importante osservazione è che, in certi casi, benché esista un algoritmo polinomiale per un certo problema (che dunque è in P), non è detto che convenga utilizzare quell'algoritmo per la sua risoluzione. In altre parole, esistono eccezioni illustri al fatto di identificare la polinomialità di un algoritmo con la sua efficienza. Il caso più importante di questo tipo è dato dalla Programmazione Lineare (PL):

PL[2]

- *Dati:* Matrice A, vettori b e c; intero k
- *Domanda:* Esiste un $x \in \mathbb{R}^n$ tale che $Ax = b, x \geq 0$ e $c^T x \leq k$?

Il metodo del simplesso, scoperto nel 1947, notoriamente non è un algoritmo polinomiale. Fino al 1978 la questione se PL appartenesse o meno alla classe P rimase aperta. In quell'anno fu scoperto un algoritmo (il *metodo dell'ellissoide*, dovuto a Khachiyan) che consentiva di risolvere problemi di programmazione lineare in tempo polinomiale nelle dimensioni dell'input – ossia, nella fattispecie, il numero di vincoli m, di variabili n e il numero di bit necessari a rappresentare tutti i coefficienti delle matrici A, b e c. Tuttavia, ancora oggi il metodo del simplesso rimane l'algoritmo più usato (anche nei package commerciali) per la soluzione di problemi di PL, e questo perché nel caso medio il suo comportamento è molto migliore che nel caso peggiore.

4 La classe NP

Cominciamo ora a estendere i nostri orizzonti al di là della classe P.

Definizione 2 *(Classe NP). Un problema A appartiene alla classe NP se, data un'istanza a affermativa di tale problema, è possibile fornire un certificato dal quale si possa verificare in tempo polinomiale che tale istanza è affermativa.*

Dunque l'appartenenza a NP richiede molto meno rispetto all'appartenenza a P. Perché un problema sia in NP, non è necessario che esso sia risolubile in tempo polinomiale, ma solo che, se l'istanza è affermativa, ciò sia *verificabile* in tempo polinomiale. Questo è qualcosa di estremamente più semplice. Si consideri infatti ad esempio il problema CICLO HAMILTONIANO. Tale problema è senz'altro in NP. Infatti, consideriamo un grafo G, e supponiamo che qualcuno abbia già trovato un ciclo hamiltoniano su tale grafo. Per convincerci che tale istanza è dunque affermativa, non dovrà fare altro che fornirci una rappresentazione del ciclo hamiltoniano, ossia la successione degli archi che lo compongono, e noi, dal canto nostro, non dovremo fare altro che verificare che tale successione di archi effettivamente esiste nel grafo. Il tutto, essendo un ciclo hamiltoniano costituito da n archi, richiederà un tempo $O(n)$, e quindi polinomiale.

[2]Si noti che avremmo potuto introdurre PL semplicemente come il problema di determinare un $x \in \mathbb{R}^n$ tale che $Ax = b, x \geq 0$, in quanto la condizione $c^T x \leq k$ può essere assorbita nei vincoli, previa introduzione di una variabile di slack. Abbiamo voluto mantenere l'indicazione esplicita della funzione obiettivo, per richiamare più da vicino la familiare forma standard di un problema di programmazione lineare.

Non si tratta dunque di determinare in tempo polinomiale se l'istanza è affermativa o negativa, ma solo, data un'istanza affermativa, di avere la possibilità di verificarlo in tempo polinomiale.[3]

A questo punto dovrebbe essere chiaro che se un problema è polinomiale, allora appartiene senz'altro anche a NP. Infatti, si consideri un'istanza affermativa di un problema polinomiale. Per verificare che quell'istanza è affermativa, basterà eseguire l'algoritmo polinomiale di risoluzione (il che, appunto, richiede un tempo polinomiale), osservando che alla fine esso produrrà la risposta SI. Dunque, sappiamo che $P \subseteq NP$. Sulla relazione tra queste due classi torneremo in seguito.

L'importanza della classe NP è legata al fatto che tutti o quasi i problemi decisionali di qualche interesse pratico fanno parte di questa classe. L'obiettivo di un modello matematico di ottimizzazione, in ultima analisi, è quello di individuare la migliore soluzione da un insieme ammissibile i cui elementi sono tipicamente vettori, insiemi di archi o di nodi, partizioni, cicli etc. E' quindi ragionevole supporre che, calcolata una soluzione, sia almeno possibile verificarne la rispondenza ai requisiti del problema, in tempo polinomiale. Non si può pensare di progettare qualcosa in modo ottimo, se non è possibile nemmeno esaminare il contenuto del progetto in tempo ragionevole (ossia, polinomiale).

A questo punto può venire quasi il sospetto che tutti o quasi i problemi umanamente concepibili cadano in NP. Questo è tutt'altro che vero. Basti pensare a una variante apparentemente piccola del problema CICLO HAMILTONIANO:

CO - CICLO HAMILTONIANO

- *Dati:* Un grafo $G = (N, A)$, non orientato
- *Domanda:* Non esiste un ciclo hamiltoniano?

Un'istanza affermativa di questo problema è un grafo *non* hamiltoniano. Prendiamo allora un grafo di questo tipo: è possibile *certificare* in tempo polinomiale che tale grafo,

[3]Una analogia che può servire a esemplificare questi concetti è la seguente. Supponete di entrare in un negozio che vende grafi, e di chiedere:"Vorrei un grafo connesso". Il negoziante va nel retrobottega e torna con una scatola, che depone sul banco. Chiaramente, non volendo fare un acquisto avventato, intendete controllare di persona se il grafo è del tipo richiesto. Aprite allora la scatola e prendete il grafo per un nodo, lo scrollate (operazione che potete fare in tempo polinomiale) e, se nessun pezzo cade per terra, potete essere certi della bontà dell'acquisto. Questo lo potete fare perché il problema CONNESSIONE è in P. Ma se aveste voluto acquistare un grafo hamiltoniano, le cose sarebbero andate diversamente. Anche in questo caso il negoziante depone sul banco la scatola, voi la aprite e tirate fuori il grafo, ma certo non avete il tempo di vedere se nel grafo in esame esiste un ciclo hamiltoniano. Chiedete allora: "Siamo sicuri che questo grafo è hamiltoniano?". Se il negoziante dà risposte evasive (del tipo "non si preoccupi", "glielo assicuro io"), voi avete il diritto – essendo CICLO HAMILTONIANO in NP – di esigere una certificazione polinomiale che il grafo sia del tipo detto. Anzi, se è onesto, il negoziante dovrebbe accludere al grafo l'indicazione dell'elenco degli archi che formano il ciclo hamiltoniano, in modo che voi possiate verificare in breve tempo la sua esistenza, senza rischiare di buttare via i vostri soldi.

per l'appunto, non ha cicli hamiltoniani? Finora nessuno c'è mai riuscito, anche se, a tutt'oggi (2009), nessuno ha nemmeno dimostrato che ciò non sia possibile. Tuttavia questo fatto è oggi ritenuto da tutti estremamente improbabile, e dunque per quel che si sa il problema CO - CICLO HAMILTONIANO non appartiene a NP. D'altro canto, invece, possiamo facilmente certificare che un'istanza di CO - CICLO HAMILTONIANO è *negativa*, ossia, che un determinato grafo *ha* un ciclo hamiltoniano. Dunque, CO - CICLO HAMILTONIANO appartiene a un'altra classe, che contiene quei problemi per i quali si riesce in tempo polinomiale a certificare che un'istanza è negativa. Questa classe è in qualche modo simmetrica rispetto alla classe NP, e viene chiamata $co-NP$. Si noti che, per gli stessi motivi visti a proposito della classe NP, si ha che $P \subseteq co-NP$. Non indagheremo oltre la classe $co-NP$ e altre classi di complessità esterne a NP, ma è bene sapere che ne esistono molte, benché talune di interesse prevalentemente teorico.

5 Riduzione tra problemi

Prima di procedere oltre nella nostra esplorazione della classe NP, occorre introdurre formalmente un concetto di importanza fondamentale, sia nella teoria della complessità computazionale che nella pratica modellistica.

Com'è noto, il problema MATCHING BIPARTITO può essere risolto efficientemente se si dispone di un algoritmo per MASSIMO FLUSSO. Basta infatti definire il grafo $H(N, A)$ di MASSIMO FLUSSO come $G(U, V, A)$ con l'aggiunta di due nodi e opportuni archi, indicare come s e t proprio i due nodi aggiunti e infine porre a 1 le capacità di tutti gli archi, e porre infine h uguale al valore k dell'istanza di MATCHING BIPARTITO. In questo modo abbiamo ricondotto la soluzione di un problema (MATCHING BIPARTITO) a quella di un altro (MASSIMO FLUSSO), o più precisamente, abbiamo *ridotto* MATCHING BIPARTITO a MASSIMO FLUSSO. Osserviamo che l'istanza di MASSIMO FLUSSO può essere costruita in modo molto efficiente a partire da quella di MATCHING BIPARTITO (si tratta in definitiva di generare lo stesso grafo, più 2 nodi e $|U| + |V|$ archi), e una volta risolta l'istanza di MASSIMO FLUSSO, la soluzione all'istanza di MATCHING BIPARTITO può essere ricostruita in modo pure molto efficiente – infatti, la semplice lettura degli archi intermedi attraversati da flusso ci dice com'è fatto il matching. La nostra, allora, è una *riduzione polinomiale*.

Definizione 3 *(Riduzione polinomiale tra problemi). Un problema A si riduce polinomialmente a un problema B se, data un'istanza a di A, è possibile costruire in tempo polinomiale un'istanza b del problema B tale che a è affermativa se e solo se b è affermativa.*

La notazione $A \rightarrow B$ indica che A si riduce a B (sottintenderemo "polinomialmente"). Si noti che se $A \rightarrow B$, risolvendo efficientemente B siamo in grado di risolvere efficientemente anche A. Quindi, possiamo dire che se $A \rightarrow B$, il problema B è *almeno tanto difficile* quanto lo è A. Un'altra osservazione di rilievo è che la riduzione tra problemi è chiaramente una relazione transitiva, ossia $A \rightarrow B$ e $B \rightarrow C$ implicano $A \rightarrow C$.

Consideriamo un importante problema classico della programmazione logica. Com'è noto, una qualunque espressione logica può essere posta in forma normale congiuntiva, ossia come *and* di *or*, ove x_i è la generica variabile diretta e $\bar{x}_i$ quella negata. Questa espressione è costituita da un certo numero di *clausole* $C_1, C_2, \ldots, C_m$, in ognuna delle quali compaiono alcune variabili booleane, dirette o negate; ciascuna variabile booleana diretta o negata prende il nome di *letterale*. Un insieme di valori delle variabili booleane, ciascuna delle quali può valere *vero* (V) o *falso* (F), prende il nome di *assegnamento di verità*.

SODDISFACIBILITÀ (SAT)

- *Dati:* Un'espressione booleana in forma normale congiuntiva, formata da m clausole e n variabili booleane $x_1, x_2, \ldots, x_n$.
- *Domanda:* Esiste un assegnamento di verità che la soddisfa, tale cioè che l'espressione assuma valore vero?

Consideriamo poi il generico problema di programmazione lineare intera in forma di riconoscimento, che indicheremo con PLI:

PLI[4]

- *Dati:* Matrice A, vettore b;
- *Domanda:* Esiste un x tale che $Ax = b, x \geq 0$, x intero?

E' ovvio che qualora vi siano delle disequazioni, queste possono essere poste in forma di uguaglianza aggiungendo opportune variabili di slack. Per lo stesso motivo non consideriamo esplicitamente la funzione obiettivo. Vogliamo mostrare il seguente risultato.

TEOREMA 1 $SAT \rightarrow PLI$.

[4]Al contrario di quanto fatto per PL, qui facciamo la scelta di non mettere in evidenza la condizione $c^T x \leq k$, che è da considerarsi quindi all'interno di $Ax = b$.

Dim. Data un'espressione booleana, consideriamo un'istanza di PLI costruita in questo modo. Associamo a ogni variabile booleana (x_i) una variabile binaria (y_i), e a ogni clausola un vincolo. Il vincolo k-esimo sarà dato dalla somma di quantità definite nel seguente modo: se nella clausola C_k compare x_i, allora nel vincolo abbiamo y_i, mentre se in C_k compare $\bar{x}_i$, allora nel vincolo compare $1 - y_i$. Nel vincolo, la somma di tutti tali termini è posta ≥ 1. Perché l'espressione booleana sia soddisfatta, è necessario e sufficiente che tutte le clausole siano soddisfatte. E' immediato verificare che tutte le clausole sono verificate se e solo se lo sono tutti i vincoli: infatti, se essi sono soddisfatti, basterà porre nell'assegnamento di verità $x_i = V$ se nella soluzione di PLI è $y_i = 1$ e $x_i = F$ se $y_i = 0$. Viceversa, se tutte le clausole sono soddisfatte, lo saranno anche tutti i vincoli ponendo $y_i = 1$ per le variabili booleane vere e $y_i = 0$ per quelle false. □

Ad esempio, si consideri l'istanza di SAT costituita da:

$$F(x_1, x_2, x_3) = (x_1 + \bar{x}_2 + \bar{x}_3)(\bar{x}_1 + x_3)(\bar{x}_2 + \bar{x}_3)(x_2 + x_3) \tag{1}$$

L'istanza di PLI associata è:

$$y_1 + (1 - y_2) + (1 - y_3) \geq 1 \tag{2}$$
$$(1 - y_1) + y_3 \geq 1 \tag{3}$$
$$(1 - y_2) + (1 - y_3) \geq 1 \tag{4}$$
$$y_2 + y_3 \geq 1 \tag{5}$$
$$y_i \geq 0, i = 1, 2, 3 \tag{6}$$
$$y_i \leq 1, i = 1, 2, 3 \tag{7}$$
$$\tag{8}$$

Questo insieme di disequazioni ha soluzione per $y_1 = 1, y_2 = 0, y_3 = 1$, da cui possiamo immediatamente ricavare l'assegnamento di verità che soddisfa l'espressione booleana: $x_1 = V, x_2 = F, x_3 = V$.

6 Problemi NP-completi

Il concetto di riduzione polinomiale ci consente di introdurre una nuova classe di complessità.

Definizione 4 *(Problema NP-completo). Un problema A si dice NP-completo se:*

- $A \in NP$
- *Preso un qualunque problema $B \in NP$, B si riduce polinomialmente ad A.*

Dunque, se un problema è NP-completo, vuol dire che esso è almeno altrettanto difficile quanto *qualunque* altro problema in NP – almeno nel senso che diamo qui alla parola "difficoltà", misurata in termini di complessità computazionale. Dunque, i problemi NP-completi rappresentano i più difficili problemi in NP. Si noti che in base alla definizione di problema NP-completo, si ha che se si trovasse un algoritmo polinomiale per *uno qualunque* di essi, avremmo trovato un algoritmo polinomiale per *tutti* i problemi in NP! E' possibile questo? La stragrande maggioranza (benché non la totalità) dei ricercatori ritiene che un algoritmo polinomiale per un problema NP-completo non possa esistere. I problemi NP-completi mostrano in genere delle caratteristiche tipiche di intrattabilità computazionale, che li fanno apparire radicalmente diversi dai problemi in P. D'altra parte, non è stato finora mai dimostrato che ciò non possa accadere. La domanda dunque se $P \subset NP$ oppure $P = NP$, costituisce ancora il problema aperto forse più rilevante di tutta l'informatica teorica. Da un punto di vista pratico, comunque, è abbastanza universalmente accettato il fatto che, se un problema è dimostrato essere NP-completo, questo sia da considerarsi indice di difficoltà intrinseca del problema, e quindi gli approcci risolutivi vanno calibrati tenendo conto di questo fatto.

Il contributo fondamentale dato da Stephen Cook nel 1971 fu quello di dimostrare – attraverso strumenti formali che utilizzano modelli di calcolo quali le macchine di Turing – il primo risultato di NP-completezza.

Teorema 2 *SAT è NP-completo.*

Non riportiamo qui la dimostrazione, ma va sottolineato che questo teorema è di importanza fondamentale, per almeno due motivi. Il primo è che ha mostrato che la classe di problemi NP-completi non è vuota. Il secondo è che, a causa della transitività della relazione di riduzione polinomiale, questo risultato può essere utilizzato per dimostrare – in modo molto più diretto – *altri* risultati di NP-completezza.

Si consideri infatti un nuovo problema, e chiediamoci se esso sia NP-completo o meno. Per dimostrare che lo è, secondo la definizione vista sopra, dobbiamo dimostrare anzitutto che questo problema è in NP. Questa parte della dimostrazione è spesso estremamente semplice (ma non sempre, va detto). La parte sostanziale della dimostrazione consiste nel mostrare che qualunque problema in NP si riduce al nostro problema. Dal Teorema di Cook, però, discende che qualunque problema in NP si riduce a SAT. Allora, basta dimostrare che *SAT si riduce al nostro problema* per completare la dimostrazione. Prendiamo ad esempio PLI. Che PLI sia in NP è evidente: data un'istanza del problema, per certificare che essa è un'istanza affermativa basterà fornire il vettore della soluzione, per poi andare a inserire i valori delle variabili nei vincoli e verificare che essi sono soddisfatti. Ma ricordando il teorema 1, abbiamo che SAT si riduce polinomialmente a PLI.

Dunque, preso un qualunque problema A in NP, si ha $A \rightarrow SAT$ (Teorema di Cook) e $SAT \rightarrow PLI$, e dunque:

TEOREMA 3 *PLI è NP-completo.* □

In questo modo è possibile dimostrare, come vedremo tra poco, la NP-completezza di molti altri problemi. Una volta dimostrata la NP-completezza di un problema, questo può essere impiegato per dimostrare la NP-completezza di altri problemi ancora. Infatti, per dimostrare che un problema è NP-completo, basta prendere un qualunque problema già notoriamente NP-completo e ridurlo al problema in esame (se è in NP).

7 Esempi di problemi NP-completi

Vediamo nel seguito di dimostrare la NP-completezza di alcuni problemi di decisione. Un primo problema molto importante è il seguente:

CLIQUE

- *Dati:* Grafo G=(N,A), non orientato; intero k
- *Domanda:* Esiste un sottografo completo (clique) di G con almeno k nodi?

TEOREMA 4 *CLIQUE è NP-completo.*

Dim. CLIQUE è chiaramente in NP, dal momento che la presenza di una clique con almeno k nodi puè essere certificata dall'indicazione dei suoi nodi, e si tratta quindi solo di verificare che, presi comunque due nodi di questo insieme, c'è un arco che li collega. Riduciamo ora SAT a CLIQUE. Data un'istanza di SAT, si costruisca un'istanza di CLIQUE come segue. Il grafo G avrà m gruppi di nodi, uno cioè per ogni clausola dell'istanza di SAT. In ciascun gruppo, avremo un nodo per ogni letterale della corrispondente clausola. Indichiamo allora con (j, l_i) il nodo del gruppo corrispondente alla clausola j e al letterale l_i (il quale, ricordiamo, può essere uguale a x_i o a $\bar{x}_i$). L'insieme degli archi è invece definito come segue. Esiste un arco tra due nodi (j, l_i) e (r, l_s) se $j \neq r$ e $l_i \neq \bar{l}_s$. In altre parole, gli archi uniscono nodi di gruppi diversi corrispondenti a letterali che non sono l'uno la negazione dell'altro. A questo punto ci chiediamo se G contiene una clique con m nodi, ossia definiamo $k = m$. (Più di m non può averne in quanto in tal caso almeno due nodi dovrebbero far parte dello stesso gruppo, mentre i nodi dello stesso gruppo non sono collegati da archi.) Se G contiene una tale clique, i letterali corrispondenti ai nodi dei vari gruppi che fanno parte della clique sono tutti 'compatibili', nel senso che se tra

essi vi compare il letterale l_i, di certo non vi compare $\bar{l}_i$. Dunque, ponendo al valore V tutti questi letterali, si ottiene un assegnamento di verità che soddisfa tutte le clausole, e dunque l'istanza di SAT è affermativa. D'altro canto, se una tale clique in G non esiste, vuol dire che non è possibile selezionare m letterali tutti mutuamente compatibili, uno da ogni clausola, e dunque l'istanza di SAT è negativa. Osserviamo infine che l'istanza di G può essere costruita in tempo polinomiale, dal momento che, indicando con m e n il numero di clausole e di variabili booleane in SAT, i nodi di G sono $O(mn)$ e gli archi sono $O(m^2n^2)$. □

Finora dunque abbiamo visto che, a partire da SAT, abbiamo potuto dimostrare la NP-completezza di altri due problemi: PLI e CLIQUE. Quest'ultimo risulta utile in molte altre dimostrazioni di NP-completezza, come vediamo di seguito.

STABLE SET

- *Dati:* Grafo $G' = (N', A')$, non orientato; intero k'
- *Domanda:* Esiste un insieme di nodi di G' a due a due non adiacenti (stable set) con almeno k elementi?

Teorema 5 *STABLE SET è NP-completo.*

Dim. Chiaramente, STABLE SET $\in NP$. Mostriamo ora che CLIQUE $\rightarrow$ STABLE SET. Data un'istanza di CLIQUE, costituita da un grafo G e un intero k, costruiamo un'istanza di STABLE SET nel seguente modo: G' ha lo stesso insieme di nodi di G, e in G' è presente un arco (i, j) se e solo se quell'arco *non* è presente in G. In altre parole, definiamo G' come il grafo complemento di G. Si ponga poi $k' = k$. E' evidente che in G' è presente un insieme stabile di cardinalità k se e solo se in G è presente una clique di cardinalità k. L'osservazione che G' può essere costruito in tempo $O(|A|)$ completa la dimostrazione. □

Un altro problema analogo è il seguente, per il quale occorre dare la seguente definizione. Dato un grafo e un sottoinsieme S dei suoi nodi, un arco si dice *coperto* da S se almeno uno dei suoi due estremi fa parte di S. Una *copertura* (o *node cover*) degli archi del grafo è un insieme di nodi tale da coprire tutti gli archi.

NODE COVER

- *Dati:* Grafo $G'' = (N'', A'')$, non orientato; intero k''
- *Domanda:* Esiste una copertura degli archi di G'' con al più k'' nodi?

Teorema 6 *NODE COVER è NP-completo.*

Dim. Al solito, NODE COVER $\in NP$, in quanto, dato l'elenco dei nodi che formano una node cover, basta verificare che ciascun arco ha almeno un estremo in tale insieme. Riduciamo ora STABLE SET a NODE COVER. Tale riduzione è immediata se si fa la seguente osservazione: dato un grafo G'', consideriamo un insieme di nodi S su tale grafo. Se tale insieme di nodi copre tutti gli archi, ciò vuol dire che non c'è nessun arco che ha *ambedue gli estremi* fuori di S, ossia nell'insieme $N'' - S$. In altre parole, i nodi di $N'' - S$ formano un insieme stabile. Ma allora ecco che chiedersi se esiste un insieme stabile di cardinalità almeno k' *equivale* a chiedersi se, sullo stesso grafo, esiste un node cover di cardinalità al più pari a $|N'| - k'$. Formalmente, data un'istanza di STABLE SET, l'istanza di NODE COVER si ottiene semplicemente ponendo $G'' = G'$ e $k'' = |N| - k'$. □

Dimostrare che un certo problema è NP-completo implica che – a meno che sia $P = NP$ – non è possibile trovare un algoritmo che in tempo polinomiale consenta di risolvere *qualsiasi* istanza di quel problema. Ciò non toglie però che *casi particolari* di un dato problema possano essere risolti invece in tempo polinomiale. Un esempio di questo tipo ci è dato ad esempio dal problema NODE COVER *su grafi bipartiti*, che chiameremo per brevità NODE COVER BIPARTITO. Il problema di determinare il matching di massima cardinalità (in forma di ottimizzazione) com'è noto può formularsi come:

$$max \quad \sum_{(i,j)\in A} x_{ij} \tag{9}$$

$$\sum_{(i,j)\in\delta(i)} x_{ij} \quad \leq \quad 1, i \in N \tag{10}$$

$$x_{ij} \quad \geq \quad 0, (i,j) \in A \tag{11}$$

dove i vincoli di interezza sono stati rilassati grazie alla totale unimodularità della matrice. Se noi scriviamo il duale di questo problema, otteniamo:

$$min \quad \sum_{i\in N} y_i \tag{12}$$

$$y_i + y_j \quad \geq \quad 1, (i,j) \in A \tag{13}$$

$$y_i \quad \geq \quad 0, i \in N \tag{14}$$

Nel duale c'è una variabile per ogni nodo, e un vincolo per ogni arco. Ciascuno dei suoi vincoli (13) esprime il fatto che, per ogni arco, la variabile associata ad almeno uno dei suoi due nodi estremi deve valere almeno 1: dunque, è proprio una formulazione di NODE COVER, ed essendo ovviamente anche la matrice dei coefficienti di tale problema totalmente unimodulare, anch'esso è risolubile come programmazione lineare – quanto

basta per asserire che NODE COVER BIPARTITO è in P. [5] Attenzione comunque a non trarre conclusioni erronee. Infatti, MATCHING su grafi non bipartiti è comunque risolubile in tempo polinomiale, mentre non è cosi', come abbiamo visto, per NODE COVER.

Uno dei problemi di ottimizzazione più importanti e più studiati è il problema dello zaino (KNAPSACK 0-1):

$$\max \quad \sum_{j=1}^{n} c_j x_j \tag{15}$$

$$\sum_{j=1}^{n} p_j x_j \quad \leq \quad b \tag{16}$$

$$x_j \quad \in \quad \{0,1\} j = 1, \ldots, n \tag{17}$$

In forma di riconoscimento tale problema può essere espresso nel seguente modo.

KNAPSACK 0-1

- *Dati:* Due vettori c, p a n componenti intere, due interi k e b
- *Domanda:* Esiste un vettore $x \in \{0,1\}^n$ tale che $c^T x \geq k$ e $p^T x \leq b$?

Per arrivare a dimostrare la NP-completezza di KNAPSACK 0-1, occorre passare per due problemi di riconoscimento che sono comunque molto importanti di per sé.

KNAPSACK ESATTO

- *Dati:* Un vettore a a n componenti intere, un intero b
- *Domanda:* Esiste un vettore $x \in \{0,1\}^n$ tale che $a^T x = b$?

PARTITION

- *Dati:* Un vettore w a h componenti intere, $w_1, w_2, \ldots, w_h$.
- *Domanda:* Esiste una bipartizione $(S, \bar{S})$ delle componenti di w tale che $\sum_{j \in S} w_j = \sum_{j \in \bar{S}} w_j$?

Nel seguito, indichiamo con 3-SAT il caso particolare di SAT in cui ogni clausola consiste esattamente di 3 letterali. Anche 3-SAT è un problema NP-completo.

TEOREMA 7 *KNAPSACK ESATTO è NP-completo.*

[5] Si noti che, per la dualità debole, la cardinalità di un qualunque matching costituisce un lower bound per la cardinalità di un qualunque node cover. Tale risultato poteva ottenersi anche facendo considerazioni puramente combinatorie. La dualità forte ci assicura poi che all'ottimo vale l'uguaglianza.

Dim. Chiaramente KNAPSACK ESATTO $\in$ NP, dal momento che un'istanza affermativa può essere certificata semplicemente esibendo il vettore a n componenti che verifica l'equazione. Mostriamo ora che 3-SAT $\rightarrow$ KNAPSACK ESATTO. Data un'espressione booleana con n variabili logiche e m clausole, definiamo un insieme di equazioni algebriche tra variabili binarie, come segue. Le equazioni sono divise in due gruppi, di n e m equazioni. Le variabili binarie che compaiono in queste equazioni sono complessivamente $2n+2m$. Alla variabile logica x_i associamo due variabili binarie, y_i e z_i. Inoltre, alla clausola j-esima, associamo le variabili binarie q_{2j-1} e q_{2j}. Le due variabili associate a ciascuna variabile logica corrispondono ai due possibili valori, V o F, di tali variabili, e ciascuna equazione del primo gruppo esprime il fatto che ogni variabile logica assume esattamente uno dei due possibili valori in un eventuale assegnamento di verità, ossia $y_i + z_i = 1$. Le equazioni del secondo gruppo, invece, esprimono il fatto che ogni clausola deve essere soddisfatta. L'equazione corrispondente alla clausola j-esima impone allora che la somma delle variabili binarie corrispondenti ai letterali che compaiono in tale clausola, più altre due variabili, ossia q_{2j-1} e q_{2j}, deve essere pari a 3. In questo modo, tale equazione è soddisfatta se e solo se almeno una delle variabili binarie corrispondenti ai letterali della clausola j-esima assume valore V. Riscrivendo il sistema di equazioni lineari in variabili booleane sotto forma matriciale, abbiamo

$$Et = d \tag{18}$$

ove E è una matrice con $n+m$ righe e $2n+2m$ colonne, t è un vettore di incognite costituito dalle variabili $y_1, \ldots, y_n, z_1, \ldots, z_n, q_1, \ldots, q_{2m}$ e d è un vettore in cui le prime n componenti sono pari a 1 e le rimanenti m sono pari a 3. Per costruzione, esiste un vettore binario t che soddisfa la (18) se e solo se l'istanza di 3-SAT è soddisfacibile. Osserviamo che ciascuna delle prime n righe di E contiene due 1 e tutti gli altri elementi sono 0, mentre ciascuna delle altre m righe contiene esattamente 5 elementi pari a 1. Possiamo a questo punto costruire un'istanza di KNAPSACK ESATTO come segue. Il vettore a ha un numero di componenti pari al numero di variabili di (18), ossia $2n + 2m$. Gli elementi del vettore a si ottengono interpretando come numeri decimali le colonne di E, ossia, formalmente, $a_k := \sum_{i=1}^{n+m} 10^{i-1} e_{ik}$, mentre analogamente l'intero b si ottiene interpretando il vettore d come un intero decimale, ossia $b := \sum_{i=1}^{n+m} 10^{i-1} b_i$. A questo punto è immediato verificare che l'istanza di KNAPSACK ESATTO cosi' costruita ha soluzione se e solo se (18) ha soluzione. Infine, il calcolo dei coefficienti a_k richiede complessivamente un tempo $O(mn)$, il che completa la dimostrazione. $\square$

Un esempio può servire a chiarire la riduzione di cui sopra. Sia l'istanza di 3-SAT data da:

$$F(x_1, x_2, x_3) = (x_1 + \bar{x}_2 + \bar{x}_3)(\bar{x}_1 + x_2 + x_3)(x_1 + x_2 + \bar{x}_3)(x_1 + x_2 + x_3) \tag{19}$$

il corrispondente sistema di equazioni binarie sarà

$$\begin{aligned} y_1 + z_1 &= 1 \\ y_2 + z_2 &= 1 \\ y_3 + z_3 &= 1 \\ y_1 + z_2 + z_3 + q_1 + q_2 &= 3 \\ z_1 + y_2 + y_3 + q_3 + q_4 &= 3 \\ y_1 + y_2 + z_3 + q_5 + q_6 &= 3 \\ y_1 + y_2 + y_3 + q_7 + q_8 &= 3 \end{aligned}$$

e l'istanza di KNAPSACK ESATTO sarà quindi con 14 variabili, indicate con $u_i, i = 1, \ldots, 14$ (corrispondenti alle y_i, z_i e q_j nell'ordine):

$$\begin{aligned} &1001011u_1 + 100111u_2 + 10101u_3 + 1000100u_4 + 101000u_5 + 11010u_6 + \\ &1000u_7 + 1000u_8 + 100u_9 + 100u_{10} + 10u_{11} + 10u_{12} + u_{13} + u_{14} = 1113333 \end{aligned} \tag{20}$$

Si noti che è essenziale che il numero di letterali in ogni clausola di SAT sia limitato: se avessimo considerato un'istanza di SAT generico, questa riduzione non era valida se qualche clausola avesse avuto più di 9 letterali: in quel caso un elemento di d doveva valere almeno 10, e dunque non potevamo più interpretare gli elementi di d come cifre di un numero decimale.

A questo punto facciamo un passo ulteriore:

Teorema 8 *PARTITION è NP-completo.*

Dim. Essendo ovviamente PARTITION in NP, mostriamo che KNAPSACK ESATTO $\rightarrow$ PARTITION. Data un'istanza di KNAPSACK ESATTO, indichiamo con α la somma di tutte le componenti del vettore a, e supponiamo che sia $b < \alpha/2$ (il caso opposto è del tutto simmetrico). Definiamo allora un'istanza di PARTITION con $h := n + 2$, ponendo $w_j := a_j, j = 1, \ldots, n$, e $w_{n+1} := 2\alpha$ e $w_{n+2} := \alpha + 2b$. Nell'istanza di PARTITION cosi' ottenuta, la somma dei pesi è pari a $\sum_{j=1}^{h} w_j = \alpha + 2\alpha + \alpha + 2b = 4\alpha + 2b$. Dunque, l'istanza è affermativa se esiste il modo di suddividere gli $n + 2$ interi in due insieme di peso $2\alpha + b$ ciascuno. Evidentemente, se tale istanza è affermativa, gli interi w_{n+1} e w_{n+2} non possono stare nello stesso insieme della partizione, poiché altrimenti tale

insieme supererebbe ampiamente il valore $2\alpha + b$. Allora l'insieme contenente w_{n+1} può contenere solo altri interi, per un peso totale di b. Ma essendo questi interi w_j uguali ai corrispondenti a_j dell'istanza di KNAPSACK ESATTO, essi inducono una soluzione anche di quel problema. Viceversa se l'istanza di PARTITION è negativa, non v'era modo di creare questo sottoinsieme di peso b e quindi anche l'istanza di KNAPSACK ESATTO era negativa. □

Infine, è abbastanza semplice a questo punto dimostrare il fatto che KNAPSACK 0-1 è NP-completo:

TEOREMA 9 *KNAPSACK 0-1 è NP-completo.*

Dim. PARTITION si riduce a KNAPSACK 0-1 semplicemente scegliendo $n := h$, $c_j := w_j$, $p_j := w_j$ e sia k che b pari a $(\sum_{j=1}^{h} w_j)/2$. □