Elementi di Teoria della Complessità

Giuseppe Lancia

Dipartimento di Matematica e Informatica
Università di Udine

1 Problemi e Istanze

I problemi oggetto di studio in Ricerca Operativa possono essere di due tipi: *Problemi di Ottimizzazione* e *Problemi di Decisione* (o di *Riconoscimento*). Appartengono al primo tipo tutti i problemi in cui ad ogni soluzione è associato un *costo* (o un *profitto*) e pertanto siamo interessati a determinare la soluzione di costo minimo (risp., profitto massimo). Appartengono al secondo tipo i problemi nei quali siamo solamente interessati a verificare l'esistenza di almeno una soluzione.

Un *problema* (di ottimizzazione o di decisione) è definito in modo astratto, dando la descrizione di un'insieme di dati (input) e delle proprietà che deve possedere una possibile soluzione (output) per essere corretta. Sia i dati che la soluzione sono descritti in termini di alcuni parametri generici (quali dei simboli matematici come n, m, G, A, ..., per rappresentare numeri, grafi, insiemi, ...). Ad esempio, nel problema del commesso viaggiatore (TSP) l'input consiste in $n(n-1)/2$ numeri che denoteremo $c(\{i,j\})$, con $1 \leq i,j \leq n$ e $i \neq j$, rappresentanti i costi degli archi di un grafo sui nodi $\{1,\ldots,n\}$, completo e pesato. La soluzione è una permutazione $v_1,\ldots,v_n$ dei nodi (corrispondente a un circuito hamiltoniano) per la quale sia minimo il valore $\sum_{i=1}^{n-1} c(\{v_i,v_{i+1}\}) + c(\{v_n,v_1\})$. L'input in questo caso è specificato dai parametri generici n e $c()$ e anche le proprietà dell'output sono espresse rispetto ai parametri generici.

Un'*istanza* del problema è un qualsiasi caso specifico del problema in questione, ottenuto assegnando un preciso valore a tutti i parametri generici. Ad esempio, i seguenti valori definiscono un'istanza del TSP: $n = 4$,

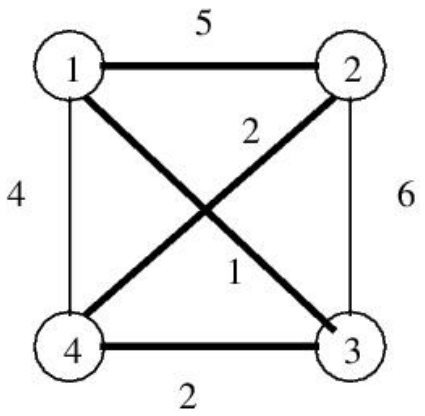

Figure 1: Un'istanza del problema TSP

$c(\{1,2\}) = 5$, $c(\{1,3\}) = 2$, $c(\{1,4\}) = 3$, $c(\{2,3\}) = 6$, $c(\{2,4\}) = 1$, $c(\{3,4\}) = 2$. La soluzione corretta di quest'istanza è la permutazione $1,3,4,2$ (si veda la Figura 2).

Definizione 1: Un *problema di ottimizzazione* Π è definito da:

O1. Un insieme $\mathcal{I}(\Pi)$ di *istanze*.

O2. Una famiglia $\mathcal{S}\Pi = \{S_I, I \in \mathcal{I}(\Pi)\}$ di insiemi di *soluzioni ammissibili*, dove l'insieme S_I è l'insieme delle soluzioni ammissibili per l'istanza I.

O3. Una *funzione obiettivo* $f_\Pi : \{(x, I), I \in \mathcal{I}(\Pi), x \in S_I\} \rightarrow R$ che ad ogni soluzione associa un valore, detto *costo*. (Si noti che per il calcolo del costo è solitamente necessaria la conoscenza dell'istanza oltre che della soluzione, per cui f_Π ha due parametri. Molto spesso però, per semplicità, si sottointende l'istanza e si esprime f come funzione di un parametro solo, ossia della soluzione ammissibile.)

Il generico problema di ottimizzazione Π consiste nel determinare, data una qualsiasi istanza $I \in \mathcal{I}(\Pi)$, una soluzione $x^* \in S_I$ tale che $f_\Pi(x^*, I) \leq f_\Pi(x, I)$ per ogni $x \in S_I$ (supponendo Π sia un problema di minimizzazione). La soluzione x^* è detta una *soluzione ottima*.

Definizione 2: Un *problema di decisione* Π è definito da:

D1. Un insieme $\mathcal{I}(\Pi)$ di *istanze*.

D2. Un sottoinsieme $\mathcal{Y}(\Pi) \subseteq \mathcal{I}(\Pi)$ di *istanze-sì* (YES-instances).

Il generico problema di decisione Π consiste nel verificare, data una qualsiasi istanza $I \in \mathcal{I}(\Pi)$, se $I \in \mathcal{Y}(\Pi)$.

Abbiamo già visto l'esempio di Π = problema del TSP. In esso, $\mathcal{I}(\Pi)$ è l'insieme di tutti i grafi completi pesati e $\mathcal{S}\Pi$ è l'insieme di tutte le permutazioni dei vertici di tali grafi (cioè, tutti i circuiti hamiltoniani). La funzione $f_\Pi(x, I)$, dato un circuito hamiltoniano x in un grafo pesato I, calcola il costo totale del circuito. Come esempio di problema di decisione, si consideri quello di determinare se un grafo è bipartito. In esso le istanze sono tutti i grafi, e le istanze-sì sono i grafi bipartiti. Analogamente si definisce il problema di decisione di determinare se un grafo è hamiltoniano.

2 Algoritmi e complessità

Un *algoritmo* $\mathcal{A}$ è un programma, scritto in un qualsiasi linguaggio di programmazione, che riceve un *input* e produce un *output*. Sia l'input che l'output sono stringhe sull'alfabeto ASCII

$$\{\texttt{'A'}, \ldots, \texttt{'Z'}, \texttt{'0'}, \ldots, \texttt{'9'}, \texttt{','}, \texttt{'.'}, \texttt{' '}, \texttt{'('}, \texttt{')'}, \texttt{'\{'}, \texttt{'\}'}, \ldots\}.$$

Per poter risolvere i problemi tramite algoritmi, bisogna fissare un modo per codificare istanze e soluzioni (che, come si è visto, sono generici oggetti matematici) come stringhe ASCII. Esistono vari modi di codificare oggetti matematici come stringhe, ma possiamo supporre che in ogni modo ragionevole:

- per rappresentare un'insieme (o una sequenza) di oggetti si usi una codifica in cui la memoria necessaria è la somma della memoria necessaria ad ogni singolo oggetto, ad esempio separandoli con `','` e racchiudendoli in parentesi `'{'` e `'}'` (risp. `'('` e `')'` per una sequenza).
- si rappresenti un numero intero usando i caratteri ASCII `'0'`, ..., `'9'`. Ad es., il numero 1244 è rappresentato dalla stringa `"1244"`, di lunghezza 4. Quindi occorrono $\lceil \log_{10} x \rceil$ caratteri per rappresentare il numero x.

- ogni indice (ad es. indice di nodo o di arco in un grafo) sia rappresentato dal corrispondente numero intero, codificato come sopra.

Ad esempio, la seguente stringa potrebbe essere una codifica per l'istanza di TSP di Figura 2:

`"4,(((1,2),5),((1,3),2),((1,4),3),((2,3),6),((2,4),1),((3,4),2))"`.

Questa stringa ha lunghezza 64, ossia, per codificare l'istanza, sono stati necessari 64 caratteri ASCII.

Definizione 3 : Supponiamo fissato un modo per codificare istanze e soluzioni come stringhe ASCII. Si dice che un algoritmo $\mathcal{A}$ *risolve un problema di ottimizzazione* Π se, ricevendo in input la codifica s_I di una generica istanza I di Π, produce in output la codifica di una soluzione ottima di I. Si dice che un algoritmo $\mathcal{A}$ *risolve un problema di decisione* Π se, ricevendo in input la codifica s_I di un'istanza I di Π, produce in output la stringa `"YES"` se $I \in \mathcal{Y}(\Pi)$ o `"NO"` se $I \notin \mathcal{Y}(\Pi)$.

Per un problema possono esistere più algoritmi, per cui si tratta di decidere quale sia il migliore. Due algoritmi possono essere confrontati principalmente guardando alla *quantità di memoria* e *al tempo di calcolo* richiesti. La risorsa più preziosa è senz'altro il tempo di calcolo, perciò, quando si confrontano algoritmi, si cerca di stabilire quale sia "il più veloce". Chiaramente, anche fissando un algoritmo, il tempo richiesto dipenderà dalla "complessità dei dati". con istanze che risultano più "facili" e altre più "difficili". La complessità di un istanza è senz'altro legata alla sua dimensione, e tanto più grande è un'istanza, tanto più tempo richiederà la sua soluzione. In modo formale, il tempo richiesto da un algoritmo viene determinato rispetto alle istanze più difficili (la cosiddetta analisi del *caso pessimo*), in funzione della loro dimensione, cioè del numero di caratteri necessari a codificarle come stringhe. Sia I un'istanza e s_I la sua codifica. Denoteremo con $|s_I|$ la *lunghezza* della stringa s_I, ossia il suo numero di caratteri. Dato un algoritmo $\mathcal{A}$ con input s_I, il *tempo* richiesto da $\mathcal{A}$ sulla stringa s_I è il numero di istruzioni (quali assegnazioni, operazioni matematiche, confronti,...) del linguaggio di programmazione, eseguite da $\mathcal{A}$ nel risolvere s_I, ed è denotato con $t_{\mathcal{A}}(s_I)$. Si noti che questo non è esattamente un tempo misurabile con un orologio, ma gli è direttamente proporzionale, visto che ogni istruzione richiede una durata prefissata (costante) per la sua esecuzione. La *complessità* (del caso

pessimo) dell'algoritmo $\mathcal{A}$ è espressa da una funzione $T_{\mathcal{A}} : \mathbb{N} \mapsto \mathbb{N}$, in cui il valore $T_{\mathcal{A}}(n)$ è il tempo impiegato da $\mathcal{A}$ nel risolvere l'istanza *più difficile* di dimensione n. La funzione $T_{\mathcal{A}}$ è definita, per ogni $n \in \mathbb{N}$ da

$$T_{\mathcal{A}}(n) := \max\{t_{\mathcal{A}}(s_I) \mid I \text{ tali che } |s_I| = n\}.$$

La complessità di un algoritmo va determinata analizzandone il codice. Ad esempio, si consideri il seguente codice per ordinare un vettore `v[]` (*Selection Sort*):

```
1. for i:=1 to n-1
2.   for j:=i+1 to n
3.     if (v[i] > v[j])
4.       { tmp:=v[i]; v[i]:=v[j]; v[j]:=tmp;}
```

In esso vi sono due cicli nidificati al cui interno (linee 3. e 4.) eseguiamo -nel caso peggiore- 4 istruzioni elementari (1 confronto, sempre, e le 3 assegnazioni). Possiamo immaginare che ogni esecuzione di un ciclo **for** richieda inoltre sempre almeno 2 istruzioni (per incrementare l'indice di ciclo e per verificare che non ecceda il limite), per cui il numero totale di istruzioni eseguite è

$$\sum_{i=1}^{n-1}(2 + \sum_{j=i+1}^{n}(2+4))$$

ossia $\sum_{i=1}^{n-1}(2+6(n-i)) = \frac{11}{2}n^2 - \frac{7}{2}n - 2$. Si noti che, la parte $\frac{7}{2}n$ è uguale a circa $1/n$ rispetto a $\frac{11}{2}n^2$, e quindi il suo contributo diventa sempre più irrilevante, al crescere di n. Quindi, ci si può concentrare su $\frac{11}{2}n^2$. Si nota che, passando da n a, ad esempio $2n$, il tempo passa da (circa) $\frac{11}{2}n^2$ a (circa) $\frac{11}{2}(2n)^2$ ossia a $4(\frac{11}{2}n^2)$. Quindi, raddoppiando la dimensione dell'istanza, il tempo è quadruplicato, e ciò non è dipeso dalla costante $\frac{11}{2}$ ma solo dal fatto che si trattava di una funzione quadratica in n. Questo tipo di considerazioni hanno portato allo sviluppo di una notazione apposita, detta *notazione "O–grande"* per l'analisi della complessità degli algoritmi.

Definizione 4: Date due funzioni $f : \mathbb{N} \mapsto \mathbb{R}$ e $g : \mathbb{N} \mapsto \mathbb{R}$, si dice che "$f$ è O–grande di g", scrivendo $f(n) = O(g(n))$, se esistono costanti $c > 0$ e N tali che $f(n) \leq cg(n)$ per ogni $n \geq N$.

Ad es. si ha $\frac{11}{2}n^2 = O(n^2)$ (ma anche $\frac{11}{2}n^2 = O(n^3)$) e $7n \log n = O(n^2)$, mentre $2^n \neq O(n^p)$ per ogni $p \in \mathbb{N}$. La notazione $O()$ serve a definire dei limiti superiori alla crescita di una funzione f, in termini di un'altra funzione g, più semplice da analizzare. Per esempio, dalla definizione, segue che un polinomio in n, quale $f(n) = a_0 + a_1 n + a_2 n^2 + \ldots + a_r n^r$, con $a_r \neq 0$, è $O(n^r)$. Inoltre, per ogni costante k, la funzione $f(n) = k$ è $O(1)$. Informalmente, per determinare la complessità di una funzione nella notazione $O()$ bisogna: (i) guardare solo il termine che cresce più velocemente con n; (ii) ignorarne le costanti moltiplicative.

La seguente definizione caratterizza gli algoritmi considerati "efficienti", che sono quelli di complessità polinomiale:

Definizione 5: Un algoritmo $\mathcal{A}$ si dice *polinomiale* se esiste una costante $p \in \mathbb{N}$ tale che $T_{\mathcal{A}}(n) = O(n^p)$ (e *non polinomiale* altrimenti).

Se un problema ammette un algoritmo efficiente lo si considera un problema (relativamente) "facile". Quindi, dalla complessità degli algoritmi si deriva una nozione di complessità per i problemi.

Definizione 6: Un problema Π si dice *polinomiale* se esiste un algoritmo polinomiale che lo risolve.

Un algoritmo non polinomiale è in generale considerato poco efficiente, ed in molti casi può rivelarsi quasi inutile per la risoluzione pratica di un problema. Per rendersi conto di quanto drammatica possa essere la differenza tra polinomiale e non polinomiale, si consideri il seguente esempio. Il problema Π="Grafo Bipartito" (BIP), richiede, dato un grafo $G = (V, E)$ di determinare se G è bipartito. Dalla definizione, sappiamo che G è bipartito se e solo se esiste una ripartizione $V = B \cup W$ dei vertici tale che ogni lato collega un vertice in B con uno in W. Quindi, un possibile algoritmo, chiamiamolo $\mathcal{A}_1$, potrebbe tentare tutte le ripartizioni di V in due sottoinsiemi non vuoti, e, per ciascuna (ve ne sono $2^{n-1} - 1$), verificare che ogni lato colleghi estremi in sottoinsiemi diversi. Supponiamo che $\mathcal{A}_1$ sia implementato con complessità $T_{\mathcal{A}_1}(n) = 6 \cdot 2^{n-1}$ (ossia $O(2^n)$). Un diverso algoritmo $\mathcal{A}_2$, che descriveremo in seguito, cerca la ripartizione $B, W = V - B$ in modo più efficiente, facendo un lavoro complessivo proporzionale al numero di lati nel grafo, e quindi con una complessità $O(n^2)$ (si ricordi che $m \leq n^2$ in ogni grafo). Supponiamo di avere realizzato $\mathcal{A}_2$ con la complessità esatta $T_{\mathcal{A}_2}(n) = 50\ n^2$. Immaginiamo di avere a disposizione un calcolatore in grado di eseguire 10^6 istruzioni al

secondo. Per determinare se un grafo di $n = 10$ vertici è bipartito, il tempo richiesto da $\mathcal{A}_1$ è circa 0.003 secondi, mentre per $\mathcal{A}_2$ è 0.005 secondi (ossia, $\mathcal{A}_1$ è migliore di $\mathcal{A}_2$ in questo caso). Passando a $n = 30$ vertici, il tempo richiesto da $\mathcal{A}_1$ diventa circa un'ora, mentre $\mathcal{A}_2$ richiede circa 4 centesimi di secondo. Per $n = 50$ vertici, il tempo richiesto da $\mathcal{A}_1$ è all'incirca un secolo, contro un decimo di secondo per $\mathcal{A}_2$. Con un grafo di $n = 100$ vertici, il tempo richiesto da $\mathcal{A}_1$ è dell'ordine di milioni di miliardi di secoli, mentre $\mathcal{A}_2$ richiede circa mezzo secondo. Questi comportamenti sono tipici del confronto polinomiale–non polinomiale, e non tanto del fatto che si sia scelto di confrontare $O(n^2)$ (anzichè $O(n^3)$, $O(n^2 \log n)$, ...) con $O(2^n)$ (anziche' $O(1.5^n)$, $O(n!)$, ...).

3 Le classi P e NP

La teoria formale della complessità dei problemi è definita per i problemi di decisione. Questa non è una vera limitazione, in quanto, come vedremo in 3.1, è possibile ricondurre un problema di ottimizzazione a uno di decisione.

La prima classe importante è la classe dei problemi polinomiali, che abbiamo già incontrato, e a cui ora diamo il nome di classe P. Informalmente, i problemi in P sono i problemi considerati "facili", mentre quelli per cui non sono noti algoritmi polinomiali, sono considerati "difficili".

Definizione 7: La classe P è la classe di tutti i problemi di decisione Π che hanno un algoritmo polinomiale che li risolve.

La seconda classe che definiamo è la classe NP[1] .

Definizione 8: Sia Π un problema di decisione. Si dice che Π appartiene alla classe NP se esiste un algoritmo $\mathcal{A}_C$ polinomiale, il cui input è una coppia di stringhe (x, y), tale che: (i) per ogni istanza $I \in \mathcal{Y}(\Pi)$, esiste una stringa c (la codifica di un *certificato* per I) per la quale $\mathcal{A}_C$ risponde `"YES"` all'input (s_I, c) (ii) per ogni istanza $I \notin \mathcal{Y}(\Pi)$, non esiste alcuna stringa c per la quale $\mathcal{A}_C$ risponde `"YES"` all'input (s_I, c).

[1]La sigla NP **non** sta per Non Polinomiale, ma per *Non-deterministic Polynomial*. Infatti, esistono algoritmi polinomiali per i problemi in NP, ma tali algoritmi sono "non–deterministici" e funzionano solo su ipotetiche macchine non deterministiche, non costruibili nella realtà.

Detto con parole più semplici, NP è la classe di problemi di decisione che possono essere *verificati* in tempo polinomiale, ossia per i quali ogni istanza-sì I ha un "certificato" di qualche tipo che, in tempo polinomiale, può essere utilizzato per verificare che I è in effetti un'istanza-sì. Come esempio, si consideri il problema "Circuito Hamiltoniano" (HC). Questo problema è in NP, in quanto, per ogni istanza-sì, un possibile certificato è dato dagli archi di un qualsiasi circuito hamiltoniano, e un possibile algoritmo $\mathcal{A}_C$ è quello che verifica che tali archi esistano tutti, e che formino infatti un circuito hamiltoniano. Un altro esempio è dato dal problema k-CLIQUE: Dato un grafo G, è vero che G possiede un sottografo completo di almeno k nodi? k-CLIQUE è in NP, in quanto un certificato per un'istanza-sì, è un insieme di k nodi che siano a due a due adiacenti, cosa che può essere facilmente verificata in tempo polinomiale. Si noti che, perchè un problema sia in NP, non è richiesto che sia facile *trovare* un certificato per un'istanza-sì (e, in effetti, in generale non lo è!), ma solo *verificarne* la correttezza, supponendo che qualcuno lo abbia trovato per noi.

Si noti che, se il cerificato fosse troppo "lungo" —esponenziale rispetto alla dimensione dell'istanza— di certo non sarebbe possibile verificarlo in tempo polinomiale. Ne consegue che la lunghezza del certificato deve essere necessariamente polinomiale rispetto all'istanza, per cui si parla anche di *certificato polinomiale* o *certificato corto*.

Se un problema può essere risolto in tempo polinomiale, allora può essere senz'altro anche verificato in tempo polinomiale (basta prendere $\mathcal{A}_C$ uguale all'algoritmo che risolve Π, e usare un certificato vuoto). Quindi

TEOREMA **1**: $\mathrm{P} \subseteq \mathrm{NP}$.

Una grande questione oggetto di studio da decenni in teoria della complessità e non ancora risolta, è determinare se l'inclusione sia propria, ossia se

$$\mathrm{P} \neq \mathrm{NP}.$$

Vi sono ottime ragioni per pensare che in effetti $\mathrm{P} \neq \mathrm{NP}$. In particolare, esistono numerosi problemi in NP per i quali, nonostante anni di sforzi e studi approfonditi, non è stato possibile determinare algoritmi polinomiali. Invece, per molti di questi problemi è stato possibile dimostrare che l'esistenza di un algoritmo polinomiale implicherebbe l'esistenza di un algoritmo polinomiale per *tutti* i problemi in NP, rafforzando l'ipotesi che un tale algoritmo sia di fatto impossibile.

L'idea fondamentale nel valutare la complessità di un problema è la seguente: supponiamo che, per risolvere un certo problema Π_1, sia sufficiente essere in grado di risolverne un altro, Π_2, e dalla soluzione di quest'ultimo ricavare la soluzione del primo. Ma allora Π_2 è perlomeno "altrettanto difficile" di Π_1. Quindi, se Π_2 è un problema "facile", anche Π_1 lo sarà, mentre se già sappiamo che Π_1 è "difficile", anche Π_2 deve esserlo. Quest'idea è definita formalmente dal concetto di *trasformabilità* fra problemi[2].

Definizione 9: Dati due problemi di decisione Π_1 e Π_2, si dice che Π_1 è *trasformabile* polinomialmente in Π_2, scrivendo $\Pi_1 \propto_T \Pi_2$, se esiste un algoritmo polinomiale $\mathcal{A}$, che, ricevendo in input una codifica s_I di un'istanza $I \in \mathcal{I}(\Pi_1)$, produce in output la codifica $s_{I'}$ di un'istanza $I' \in \mathcal{I}(\Pi_2)$, tale che $I \in \mathcal{Y}(\Pi_1)$ se e solo se $I' \in \mathcal{Y}(\Pi_2)$.

Come esempio, si consideri il problema del *Cammino Hamiltoniano* (HP): dato un grafo, determinare se esiste un cammino che passa per tutti i nodi una e una sola volta. Si può far vedere che il problema del ciclo hamiltoniano può essere trasformato in quello del cammino hamiltoniano. Dato un grafo G su cui si cerca un ciclo hamiltoniano, si prenda un qualsiasi nodo v. Si definisca poi $G' = (V', E')$ con nodi $V \cup \{v', a, a'\}$. Il nodo v' serve da "gemello" del nodo v (Figura 3). Infatti, in E' ci sono tutti gli archi di E, e, in più, gli archi in $\delta(v)$ sono duplicati, ossia per ogni arco $\{i, v\}$ in $\delta(v)$ si crei un arco $\{i, v'\}$ in E'. Infine si aggiungano i due archi $\{v, a\}$ e $\{v', a'\}$. In G', ogni possibile cammino hamiltoniano deve paritire da a e finire in a', quindi v e v' sono attraversati come secondo e penultimo nodo. La parte del cammino tra v e v' corrisponde a un ciclo hamiltoniano in G, sicchè tale cammino esiste se e solo se G ha un ciclo hamiltoniano. Quindi, se sapessimo trovare cammini hamiltoniani in modo efficiente, potremmo trovare anche cicli hamiltoniani in modo efficiente, e pertanto trovare cammini hamiltoniani è perlomeno altrettanto difficile che trovare cicli hamiltoniani. Per esercizio, si disegni una trasformazione inversa, in cui la ricerca di un cammino hamiltoniano è trasformata nella ricerca di un ciclo hamiltoniano.

Definizione 10: Un problema di decisione Π si dice NP-*completo* se

[2]A Turingopoli l'unico medico era un dentista. Questi sapeva curare qualsiasi mal di denti, ma nient'altro. Un giorno al piccolo Mario venne un forte mal di stomaco con crampi. Non sapendo come curarlo, suo papà gli diede una forte sberla e gli ruppe un dente. Così, lo trasformò in un problema per il dentista.

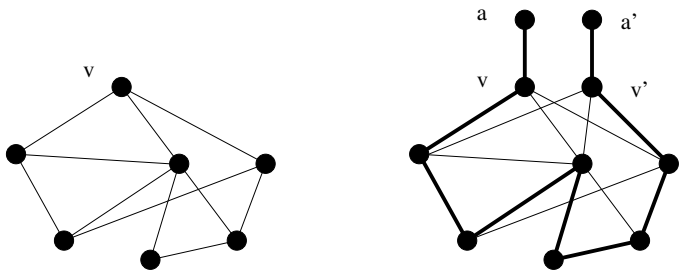

Figure 2: Trasformazione di HC in HP

1. $\Pi \in$ NP.

2. Per ogni $\Pi' \in$ NP, si ha $\Pi' \propto_T \Pi$.

L'esistenza di almeno un problema NP-completo fu stabilita da Stephen Cook nei primi anni '70, dimostrando che il problema della SODDISFACIBILITÀ (SAT) di un'espressione booleana è NP-completo. In questo problema, ci si chiede se, data una formula booleana fatta di variabili booleane legate da operatori `AND`, `OR` e `NOT`, esistono dei valori per le variabili per i quali la formula vale `TRUE`.

TEOREMA **2**: (S. Cook) SAT è NP-completo.

Una volta noto un problema NP-completo Π', è possibile dimostrare che un secondo problema Π è NP-completo facendo vedere che $\Pi' \propto_T \Pi$:

TEOREMA **3**: Sia Π' NP-completo e $\Pi \in$ NP. Allora, se $\Pi' \propto_T \Pi$, anche Π è NP-completo.

La semplice dimostrazione consiste, in pratica, nel fare vedere che vale la proprietà transitiva di $\propto_T$. In questo modo, Richard Karp ed altri dimostrarono che molti problemi notoriamente difficili sono NP-completi. Tra questi, il circuito hamiltoniano. In virtù della trasformazione vista poco fa, e del fatto che si può vedere facilmente che HP$\in$ NP, ne consegue che il problema del cammino hamiltoniano è NP-completo. Altri problemi NP-completi famosi includono k-CLIQUE (esiste una clique di almeno k nodi?), k-STABLESET (esiste un insieme stabile di almeno k nodi?), 3-COLORABILITY

(esiste una colorazione dei nodi di un grafo con tre colori, in modo che ogni arco abbia gli estremi di colore diverso?) e molti altri che incontreremo in seguito.

I problemi NP-completi sono i più difficili problemi in NP, e pertanto i maggiori candidati a non essere problemi polinomiali. Infatti, vale il

TEOREMA **4**: Sia Π NP-completo. Allora, se $\Pi \in P$, si ha $P = NP$.

Infatti, in questo caso, per qualsiasi problema Π_0 in NP esisterebbe un algoritmo polinomiale, ottenuto richiamando in sequenza l'algoritmo di trasformazione da Π_0 a Π seguito dall'algoritmo polinomiale che risolve Π. Quindi, se un qualsiasi problema NP-completo fosse risolvibile in tempo polinomiale *tutti* gli infiniti problemi in NP sarebbero risolvibili in tempo polinomiale! Questa circostanza è talmente improbabile che, qualora di fronte a un problema NP-completo, conviene abbandonare ogni sforzo verso la ricerca di un algoritmo polinomiale e dirigerli alla ricerca di algoritmi alternativi. Tra questi, i principali sono

- algoritmi esponenziali nel caso pessimo ma veloci sulla maggior parte delle istanze incontrate nelle applicazioni reali (algoritmi di tipo *branch-and-bound*).
- algoritmi polinomiali, che non garantiscono di trovare sempre la soluzione esatta ma che garantiscono "di non essere troppo lontani dalla soluzione esatta" (*algoritmi approssimati*).
- algoritmi polinomiali, che non garantiscono alcun margine sull'errore rispetto alla soluzione ottima, ma che, nella pratica, restituiscono quasi sempre soluzioni di qualità molto buona. (*algoritmi euristici*).

3.1 Problemi NP-difficili (NP-hard)

Per quanto la teoria della complessità sia definita per i problemi di decisione, la si può facilmente estendere ai problemi di ottimizzazione.

Definizione 11: Dati due problemi Π_1 e Π_2, si dice che Π_1 è *riducibile* polinomialmente a Π_2, scrivendo $\Pi_1 \propto \Pi_2$, se esiste un algoritmo polinomiale $\mathcal{A}$, che risolve Π_1 facendo uso, una o più volte, di una ipotetica "istruzione-macchina" (ossia un'istruzione di costo $O(1)$) in grado di risolvere Π_2.

Si noti che non è richiesto che i problemi siano in NP. Inoltre, è chiaro che alla trasformazione corrisponde un tipo speciale di riduzione, in cui l'unica chiamata alla procedura per Π_2 è effettuata alla fine della costruzione dell'istanza di Π_2.

Siccome la composizione di due polinomi è ancora un polinomio, ne consegue che se Π è un problema polinomiale, e se $\Pi' \propto \Pi$, allora anche Π' è un problema polinomiale. Anche in questo caso, quindi, possiamo dedurre l'improbabilità che Π sia un problema polinomiale, qualora sia noto che Π' sia un problema difficile. In particolare, Π' potrebbe essere NP-completo.

Definizione 12: Sia Π' un problema NP-completo. Un problema Π si dice NP-*difficile* (NP-*hard*) se $\Pi' \propto \Pi$.

Si noti che non è richiesto che $\Pi \in$ NP. Come esempio, dimostriamo che il problema TSP è NP-hard, facendo vedere che HC$\propto$TSP. Infatti, dato un grafo $G = (V, E)$, un algoritmo per determinare se esista un circuito hamiltoniano potrebbe procedere in questo modo. Prima assegna un costo $c_{ij} = 1$ ad ogni lato $\{i, j\} \in E$ ed un costo $c_{ij} = n+1$ ad ogni lato $\{i, j\} \notin E$, e poi richiama l'istruzione-macchina per la soluzione del TSP, con i suddetti costi. E' immediato verificare che la soluzione ottima del TSP ha costo n se e solo se c'era un ciclo hamiltoniano in G e costo $> n$ altrimenti. Quindi, si può risolvere HC dal confronto dell'ottimo di TSP con n.

Per concludere questa sezione, mostriamo come anche la trasformazione inversa sia possibile, ossia come si possa risolvere un problema di ottimizzazione (e quindi non in NP) tramite un problema di decisione. Supponiamo che il problema di ottimizzazione Π richieda di trovare il valore ottimo (ad esempio, massimo) di una certa funzione f a valori in $\mathbb{N}$. Allora si può definire un problema di decisione Π' con istanze (I, k), dove $I \in \mathcal{I}(\Pi)$ e $k \in \mathbb{N}$, del tipo "Esiste $x \in S_I$ con $f(x) \geq k$?". Il massimo k per cui Π' ha soluzione è il valore ottimo di Π e quindi lo si può determinare, ad esempio, con un ciclo in cui si aumenta k, partendo da 1. In questo modo, si può trovare la clique di dimensione massima con al più $O(n)$ chiamate di una procedura per k-CLIQUE. Quindi, una procedura polinomiale per k-CLIQUE implicherebbe un algoritmo polinomiale per MAXCLIQUE.

Bisogna però fare attenzione nel caso in cui l'istanza del problema di ottimizzazione contenga valori numerici, quali ad esempio, dei pesi sui nodi di un grafo. Infatti, si supponga di voler determinare la clique di peso massimo (WCLIQUE) in un grafo pesato sui nodi. Ragionando come sopra, per

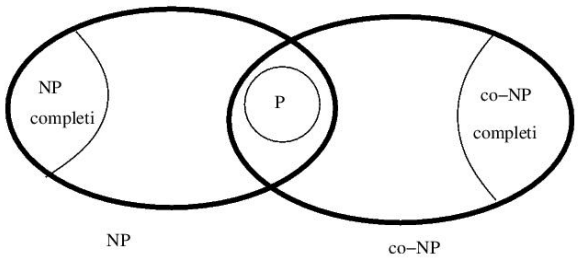

Figure 3: Mappa delle classi di complessità.

trovare la soluzione di WCLIQUE si può pensare di iterare, per $k = 1, 2, \ldots$, il problema k-WCLIQUE: esiste una clique di peso almeno k? Questo però potrebbe richiedere $O(v)$ passi, dove v è il valore della soluzione ottima. Tale v può risultare dell'ordine di nM dove M è il peso massimo di un nodo. La dimensione dell'istanza, però, è dell'ordine di $n \log M$, per cui v è *esponenziale* rispetto alla dimensione dell'istanza. In questo caso, la soluzione è quella di determinare l'ottimo non già con una ricerca sequenziale, ma con una ricerca binaria, nell'intervallo $\{1, \ldots, nM\}$. Tale ricerca richiede al più $\log nM$ passi, e quindi risulta polinomiale rispetto alla lunghezza dell'istanza.

3.2 La "Geografia" delle classi di problemi

Alla classe NP resta associata una nuova importante classe di problemi, detta co-NP, ottenuta *complementando* i problemi in NP.

Definizione 13: Sia Π un problema di decisione. Il problema di decisione Π' è detto *complementare* di Π se $\mathcal{I}(\Pi) = \mathcal{I}(\Pi')$ e $\mathcal{Y}(\Pi') = \mathcal{I}(\Pi) - \mathcal{Y}(\Pi)$. Si definisce co-NP l'insieme di tutti i problemi complementari di problemi in NP.

Come esempio si consideri il problema HC. Il suo complementare NONHC consiste nel verificare, dato un grafo G, se G *non* contiene alcun circuito hamiltoniano. In modo simile, si definisce il problema complementare di BIP, ossia il problema NONBIP di verificare se G *non* è un grafo bipartito.

Per definizione, i problemi NONHC e NONBIP sono in co-NP. Inoltre, dalla definizione segue che un problema è in co-NP se esiste un certificato polinomiale per tutte le istanze la cui risposta è `"NO"`. In particolare BIP gode

certamente di questa proprietà: un certificato polinomiale per un grafo non bipartito è un qualsiasi ciclo dispari. Quindi $BIP \in$ NP$\cap$co-NP. Questo non è un caso, ma deriva dal fatto che BIP è un problema polinomiale. Infatti, si può abbastanza facilmente dimostrare che P $\subseteq$ NP $\cap$ co-NP. Diversa è la situazione per HC: infatti, un possibile certificato che dimostri che un grafo non ha circuiti hamiltoniani, è l'elenco di tutti i circuiti, e un algoritmo verificatore può contare i nodi di ciascun circuito. Un tale elenco non è certamente corto (polinomiale) nella lunghezza dell'istanza. Tuttavia, non sono noti certificati più brevi per questo problema, e non siamo in grado di affermare che $HC \in$ co-NP. Di nuovo, questo non è casuale, ma deriva dal fatto che HC è NP-completo. Infatti, se per un qualsiasi problema Π NP-completo si avesse $\Pi \in$ co-NP, allora ne seguirebbe (esercizio) che NP = co-NP, un'alternativa rituenuta dagli esperti altrettanto improbabile che l'uguaglianza P = NP. Ne consegue che, allo stato delle conoscenze attuali, il grafico più attendibile per la geografia delle classi di complessità dei problemi è quello di Figura 4. Si noti però che, dovesse risultare P = NP, l'intero grafico collasserebbe ad un'unica classe, P.

4 Esercizi

Esercizio Si dimostri che i seguenti problemi sono NP-completi:

1. Dato un grafo $G = (V, E)$ e $V' \subseteq V$, esiste un albero di supporto le cui foglie siano tutte contenute in V'?

2. Dato un grafo $G = (V, E)$ e un intero k, esiste un albero di supporto con al più k foglie?

3. Dato un grafo $G = (V, E)$ e un intero k, esiste un albero di supporto in cui ogni nodo ha grado al più k?

4. Dato un grafo $G = (V, E)$, esiste un albero di supporto in cui ogni nodo interno (i.e., non foglia) ha nodo esattamente 4?

5. Dato un grafo $G = (V, E)$, esiste un sottografo completo di dimensione $\geq |V|/1000$?

6. Dato un grafo $G = (V, E)$, e un intero k esiste un sottografo "quasi" completo di dimensione $\geq k$ (dove per quasi completo, si intende che il

sottografo contiene almeno $k(k-1)/4$ archi, ossia almeno la metà del numero di archi di un sottografo completo)?

7. Dato un grafo $G = (V, E)$, esiste un albero di supporto in cui il numero di foglie è $\leq |V|/1000$?