NP-Completezza

e la complessità strutturale degli algoritmi

Simone Frassanito

Dipartimento di Elettronica per l'Automazione
Università degli Studi di Brescia

Cosa non è l'NP-Completezza

Si potrebbe pensare che...

... Il termine "NP" significhi *Non-P*, ma non bisogna lasciarsi trarre in inganno.

Invece...

... "NP" è acronimo di *Non deterministic Polinomial time* (tempo Polinomiale Non deterministico), perché (la classe) NP fu originariamente studiata nel contesto del non determinismo.

Qui vedremo una nozione più semplice, quella equivalente di verifica.

Prima però qualche nozione...

Complessità algoritmica

Gli algoritmi hanno differenti complessità computazionali. A noi interessano:

- il tempo polinomiale: $\exists k > 0 : \ O(n^k)$;
- il tempo superpolinomiale: considereremo $\exists k > 0 : \ O(k^n)$.

Perché solo polinomiale o superpolinomiale?

Per alcune considerazioni che non sono propriamente matematiche, ma hanno un notevole risvolto pratico:

- solitamente le complessità $O(n^k)$ hanno k piccoli;
- per numerosi modelli, un problema che può essere risolto in tempo polinomiale con un modello, può essere ancora risolto in tempo polinomiale con un altro;
- la classe dei problemi risolvibili in tempo polinomiale ha interessanti proprietà di chiusura.

Complessità algoritmica

Attenzione, perché

ci sono anche algoritmi per i quali non si sa qual è la complessità. Cosa significa: “non si sa”? Significa che non si è ancora riusciti a dimostrare che:

- $T(n) \leq O(n^k)$ oppure
- $T(n) \geq \Omega(k^n)$.

Cosa c'entra questo con l'NP-Completezza?

Gli informatici teorici

hanno raggruppato i problemi in due classi:

- La classe di complessità P;
- La classe di complessità NP.

Queste dividono i problemi basandosi su due caratteristiche:

- essere risolvibili in tempo polinomiale;
- essere verificabili in tempo polinomiale.

Cosa c'entra questo con l'NP-Completezza?

Risolvibilità

Risolvere un problema significa che data una sua istanza ne fornisco una soluzione.

Verificabilità

Verificare un problema significa che data una sua istanza e una sua soluzione, controllo se la soluzione risolve effettivamente l'istanza del problema.
Nel prosieguo chiameremo certificato la soluzione da controllare.

Cosa c'entra questo con l'NP-Completezza?

Classi di complessità

P è l'insieme dei problemi risolvibili in tempo polinomiale.
NP è l'insieme dei problemi verificabili in tempo polinomiale.

Ne consegue che $P \subseteq NP$, perché posso usare un qualsiasi algoritmo di P per generare la soluzione e confrontarla con un certificato.

Quello che non si sa...

... È se $P = NP$, oppure se $P \subsetneq NP$, ossia se ci sono problemi verificabili in tempo polinomiale che *non* sono risolvibili in tempo polinomiale.

NP-Completezza

Il problema è quindi capire se $P = NP$. Fortunatamente, la classe NP si può in un qualche modo partizionare in classi di problemi riducibili ad altri problemi, ritenuti "più difficili" di tutti gli altri.
Questa classe è la *Classe dei problemi NP-Completi (NP-C)*.

Dove sta il problema?

A oggi sappiamo risolvere i problemi NP-C solo in tempo esponenziale, *ma* è possibile che esitano algoritmi in tempo polinomiale per risolvere questi problemi.

NP-Completezza

I problemi NP-C godono di un'importante proprietà: se si scoprisse un algoritmo che risolve *uno solo* di questi problemi in tempo polinomiale, tutti i problemi di NP-C (e come conseguenza quelli di NP) sarebbero risolvibili in tempo polinomiale. Si dimostrerebbe che $P = NP$.

Facciamo sul "serio"...

Ora cercheremo di avere un'idea di come il problema è affrontato nell'ambito dell'informatica teorica. Quello che faremo sarà:

- formalizzare il concetto di "problema";
- astrarci dal particolare linguaggio usato per descrivere il problema;
- astrarci dal particolare problema;
- formalizzare le classi P e NP;
- capire come si "partizionare" NP nella classe NP-C;
- trovare un problema NP-C rappresentativo del (meta-)problema $P = NP$.

Definizione di problema astratto

Cos'è un problema?

Un problema astratto Q^A è una relazione

$$Q^A : I \to S$$

dove ad ogni istanza $i \in I$ è associata una soluzione $s \in S$.
Es. Ricerca di un cammino hamiltoniano.

Problema di decisione

Un problema astratto di decisione Q_D^A è una relazione

$$Q_D^A : I \to \{0, 1\}$$

dove ad ogni istanza $i \in I$ è associata una "risposta".
Es. Esistenza di un cammino di lunghezza al più k.

Tipi di problemi

Problema di ottimizzazione come problema di decisione

Un problema astratto di ottimizzazione Q_O^A è un problema nel quale si massimizza/minimizza un valore.
Può essere riformulato in un problema Q_D^A se si pone una limitazione al valore da ottimizzare.
Es. La ricerca del cammino minimo diventa la ricerca di un cammino di lunghezza al più k.

Tipi di problemi

Dal fatto che:

$$Q_O^A \text{ facile} \Rightarrow Q_D^A \text{ corrispondente facile}$$

ne consegue (negando l'implicazione):

$$Q_D^A \text{ difficile} \Rightarrow Q_O^A \text{ corrispondente difficile}$$

Dato che a noi interessa determinare i problemi difficili, ossia quelli in NP, possiamo interessarci solo ai problemi Q_D^A.

Problemi concreti

Nell'atto della risoluzione, un problema astratto viene rappresentato in qualche modo tramite una codifica, ossia una corrispondenza:

$$c(\cdot) : O \rightarrow \Lambda^*$$

da un insieme O di oggetti astratti a una stringa Λ^* formata tramite un alfabeto Λ con cardinalità almeno 2 ($\#\Lambda \geq 2$).

Un problema Q^A espresso tramite una codifica è detto problema concreto, e lo indicheremo con Q^C.

Codifica

Considerato l'afabeto $\Sigma = \{0, 1\}$, possiamo definire la codifica che useremo da qui in avanti come:

$$c(\cdot) : I \rightarrow \Sigma^*$$

per la quale il problema Q_D^A viene sostituito da:

$$Q_D^C : \{0, 1\}^* \rightarrow \{0, 1\}$$

Codifica

Codifiche diverse possono portare a tempi computazionali diversi.

Una codifica si dice codifica ragionevole se:

- utilizza almeno due simboli:
- non introduce dati irrilevanti;
- non richiede una generazione esponenziale di dati per la rappresentazione di un'istanza del problema.

Correlazione polinomiale

Ogni codifica ha un tempo computazionale differente? No. Possiamo raggruppare tutte le codifiche che forniscono la stessa complessità computazionale.

Definizione

Se trovo due funzioni per le quali esistono algoritmi che ne calcolano l'output dato l'input in tempo polinomiale, tali che:

$$f_{12}(c_1(i)) = c_2(i) \quad \text{e} \quad f_{21}(c_2(i)) = c_1(i)$$

allora c_1 e c_2 si dicono correlate polinomialmente.

Se c_1 o c_2 sono *correlate polinomialmente*, allora usare una o l'altra è equivalente.

Tempo computazionale

Diciamo che un algoritmo A *risolve* un problema in tempo $O(T(n))$ per un istanza con lunghezza $n = |c(i)|$, se fornisce la soluzione in al più $O(T(n))$.

Tempo polinomiale

Diciamo che un algoritmo A *risolve* un problema in *tempo polinomiale* se: $\exists k$: A risolve il problema in $O(n^k)$

Assumeremo d'ora in poi che le codifiche siano ragionevoli e concise, in particolare si assumerà che la codifica di un intero sia correlata polinomialmente alla sua rappresentazione binaria, e che la codifica di un insieme finito sia correlata polinomialmente alla sua codifica come lista di elementi, chiusi tra parentesi e separati da virgole. Da questa codifica si possono derivare ragionevoli codifiche per altri oggetti matematici come n-uple, grafi e formule.

Classe di complessità P

Definizione (II)

La Classe di complessità P è l'insieme dei *problemi concreti* risolvibili in tempo polinomiale.

N.B. Non stiamo invalidando la definizione data prima relativa ai problemi astratti: le due cose coincidono.

Passaggio ai linguaggi formali

La scelta di trattare solo problemi Q_D consente la rappresentazione tramite linguaggi.

Definizione: un linguaggio L è...

... Un qualsiasi insieme di stringhe costituite da simboli di un alfabeto Λ. Ad es:

$$\Lambda = \Sigma = \{0, 1\} \quad \text{da cui} \quad L = \{10, 11, 101, 111, 1011, \dots\}$$

Equivalenza $Q \sim L$

Nella teoria dei linguaggi, l'insieme delle istanze di qualunque problema di decisione Q_D è un sottoinsieme di Σ^*, e:

$$Q_D \sim L = \{x \in \Sigma^* : \ Q_D(x) = 1\}$$

Analogamente, con un algoritmo A si vorrebbe avere:

$$Q_D \sim L = \{x \in \Sigma^* : \ A(x) = 1\}$$

così da utilizzare il paradigma dei linguaggi formali per esprimere sinteticamente la relazione tra problemi di decisione e algoritmi che li risolvono.

Equivalenza $Q \sim L$

C'è però il problema della *Fermata di Turing*, per il quale:

$$(x \notin L) \not\Rightarrow (A(x) = 0)$$

Riconoscere VS Decidere

Facciamo dunque una distinzione:

- *Riconoscere* un linguaggio L significa che $\forall x \in \Sigma^*,\ x \in L \Rightarrow A(x) = 1$;
- *Decidere* un linguaggio L significa che $\forall x \in \Sigma^*,\ x \in L \Leftrightarrow A(x) = 1$, ossia

$$\forall x \in \Sigma^*, \begin{cases} x \in L \Rightarrow A(x) = 1 \\ x \notin L \Rightarrow A(x) = 0 \end{cases}$$

Classe P

Definizione (III)

Secondo il paradigma della teoria dei linguaggi, possiamo definire la *Classe di complessità P* come:

$$P = \{L \subseteq \Sigma^* : \exists A : \ L \text{ deciso in tempo polinomiale}\}$$

Ora siamo completamente indipendenti e dalla codifica e dal problema.

Vale anche:

$$P = \{L \subseteq \Sigma^* : \exists A : \ L \text{ riconosciuto in tempo polinomiale}\}$$

Finora abbiamo visto algoritmi che *risolvono* dei *problemi*, fornendo una *soluzione*.

Si può fare qualcos'altro? Sì, possiamo ad esempio *verificare* se una *soluzione* data risolve un *problema*.

Esempio

La *ricerca* di un ciclo hamiltoniano su un grafo è risolvibile in tempo esponenziale ma la *verifica* che un ciclo dato è hamiltoniano può essere fatta in tempo polinomiale.

Algoritmi di verifica

Definizione

Un algoritmo di verifica funzione dell'istanza $x \in I$ e del certificato $y \in Y$ è una relazione:

$$A: I \times Y \to \{0,1\}$$

$$(x,y) \mapsto \begin{cases} 1 & \text{y è soluzione dell'istanza x} \\ 0 & \text{altrimenti} \end{cases}$$

Definizione

Un linguaggio verificabile è tale per cui:

$$L = \{x \in \Sigma^* : \exists y \in \Sigma^* : A(x,y) = 1\}$$

Classe di complessità NP

Definizione

Possiamo allora definire la Classe di complessità NP come:

$$L \in NP \Leftrightarrow L = \{x \in \Sigma^* : \exists c,k > 0 : \exists y \in \Sigma^*, |y| = O(|x|^c) : \exists A(x,y), T_A(|y|) = O(|y|^k) : A(x,y) = 1\}$$

Problema $P = NP$

- Il problema del ciclo hamiltoniano è in $NP \Rightarrow NP \neq \emptyset$;
- $L \in P \Rightarrow L \in NP$.

$(L \in P \Rightarrow L \in NP) \Rightarrow (P \subseteq NP)$ ma non è detto che $P \supseteq NP$, quindi i casi sono:

- $P = NP$;
- $P \subseteq NP$.

La domanda rimane se $P = NP$.

La classe NP-C

La *Classe di complessità NP-C* è l'insieme dei problemi "più difficili" di NP.

Definizione

Per definirla bisogna introdurre il concetto di riducibilità: un linguaggio L_1 è *riducibile in tempo polinomiale* ad un linguaggio L_2 e si scrive $L_1 \leq_p L_2$, se esiste una funzione calcolabile in tempo polinomiale $f : \Sigma^* \rightarrow \Sigma^*$ tale che:

$$\forall x \in \Sigma^*,\ x \in L_1 \Leftrightarrow f(x) \in L_2$$

Lemma

$$L_1 \leq_p L_2 \Rightarrow (L_2 \in P \Rightarrow L_1 \in P),\ \forall L_1, L_2 \subseteq \Sigma^*$$

La classe NP-C

La riduzione consente di dimostrare che un problema è arduo almeno quanto un altro, a meno di un fattore di tempo polinomiale.

Definizione

Un linguaggio $L \subseteq \Sigma^*$ è NP-Completo se:

1. $L \in NP$
2. $\forall L' \in NP,\ L' \leq_p L$

La classe NP-C

Teorema

$$\exists L \in \text{NP-C risolvibile in tempo polinomiale} \Rightarrow (P = NP)$$

$$\Updownarrow$$

$$\exists L \in \text{NP-C } \textit{non} \text{ risolvibile in tempo polinomiale} \Rightarrow (P \neq NP)$$

Per questo motivo, per rispondere al quesito $P = NP$, "basta" trovare un problema NP-C risolvibile in tempo polinomiale.

La classe NP-C

Osservazione

Se un progettista di algoritmi dimostra che un problema è NP-C, ha l'"indicazione" che il problema è intrattabile, e quindi può scegliere di:

- cercare un algoritmo esatto in tempo polinomiale;
- implementare subito un algoritmo approssimato.

Un problema NP-C

Abbiamo definito P, NP e NP-C.
Come determinare se un problema è in NP-C?

Ridurre tutti i linguaggi NP a uno specifico linguaggio L è faticoso: sarebbe utile avere un metodo più rapido per dimostrare che un linguaggio L è in NP-C. Si può procedere secondo:

1. $\exists L' \in$ NP-C
2. $L \in NP$
3. $L' \leq_p L \Rightarrow L \in$ NP-C

Cosa ce ne facciamo di questo problema NP-C?

Avere un problema NP-C significa quindi potere dimostrare che altri problemi sono in NP-C in modo semplice.
Ora non ci resta che trovare almeno un problema NP-C.

Il problema NP-C di cui parlaremo è la soddisfattibilità di circuiti.

Soddisfattibilità di circuiti

Essendo SdC il primo problema NP-C, dobbiamo trovarlo necessariamente riducendo tutti i linguaggi di NP a SdC.

Il problema SdC è in NP-C perché:

- $SdC \in NP$
- $\forall L \in NP,\ L \leq_p SdC$

Soddisfattibilità di formule

Ora consideriamo il problema SdF di verificare la soddisfattibilità di una formula booleana.

Anche il problema SdF è in NP-C, questa volta perché:

1. $SdC \in$ NP-C (appena visto)
2. $SdF \in NP$ (semplice da vedere)
3. $SdC \leq_p SdF \Rightarrow SdF \in$ NP-C (unica cosa da dimostrare)

Altri problemi

Molti altri problemi che conosciamo sono in NP-C:

- 3-SdF FNC:
  la SdF in forma normale congiuntiva con clausole formate da 3 lettere distinte;
- Il problema della cricca:
  in un grafo non orientato, una *cricca* è un sottoinsieme di vertici tale che ogni coppia di suoi vertici è connessa da un arco (ossia è un sottografo completo);
- Copertura di vertici;
- Somma di un sottoinsieme;
- Ciclo hamiltoniano;
- Commesso viaggiatore.

Approfondimenti

- Introduzione agli algoritmi, § 36
  Cormen, Leiserson, Rivest - 1999
- Sudoku, Campo minato come problemi NP su
  `http://linux.andreagozzi.com/content/files/np_and_co/presentation.pdf`
- Wikipedia

A questo punto chi vuole può mettersi alla ricerca di un algoritmo in tempo polinomiale che risolva un qualsiasi problema NP-C!

Il *Clay Institute* ha messo in palio 1 000 000 $ per chi dipana il dubbio (`http://www.claymath.org/millennium/`).

Non appena risolto questo problema, si possono fare risolvere gli altri 6 ad un calcolatore, quindi il premio sarebbe di 7 000 000 $. Appena risolvete il problema non dimenticatevi di risolvere gli altri...