P, NP, NP Completezza ed NP-Hard

Tutorato 11: Conclusione corso, NP-Completezza ed NP-Hard

Automi e Linguaggi Formali

Gabriel Rovesti
Corso di Laurea in Informatica - Università degli Studi di Padova

28 Maggio 2025

Contents

1 Introduzione alla Teoria della Complessità

1.1 Motivazione: Il Problema del Domino

Per comprendere la distinzione tra problemi facili e difficili, consideriamo due varianti del gioco del domino:

Concetto chiave

Domino[1]: Dato un insieme di tessere del domino, disporre tutte le tessere in fila in modo che i numeri adiacenti corrispondano.
Domino[2]: Dato un insieme di tessere del domino, disporre alcune tessere in fila in modo che ogni numero compaia esattamente due volte.

Questi due problemi, apparentemente simili, hanno complessità computazionale molto diverse e ci introducono alle classi P e NP.

1.2 Tesi di Church Computazionale

Definizione

La **Tesi di Church Computazionale** afferma che tutti i formalismi di calcolo ragionevoli sono equivalenti a meno di fattori polinomiali nei tempi di calcolo.

Questo principio ci permette di considerare equivalenti:

- Macchine di Turing Deterministiche
- Linguaggi di programmazione concreti (Java, C++, Python, etc.)

Eccezioni notevoli: Computer quantistici e DNA Computing, che possono offrire vantaggi computazionali significativi per certi problemi.

2 Problemi Trattabili e Intrattabili

2.1 Definizione di Trattabilità

Definizione

Un problema è **trattabile** (facile) se esiste un algoritmo efficiente per risolverlo. Gli algoritmi **efficienti** sono algoritmi con complessità polinomiale: il loro tempo di esecuzione è $O(n^k)$ per qualche costante k.

Concetto chiave

Avere complessità polinomiale è una condizione minima per considerare un algoritmo efficiente. Un algoritmo con complessità più che polinomiale (es. esponenziale) è considerato non efficiente perché non è scalabile.

2.2 Esempio: Domino[1] è Trattabile

Il problema Domino[1] può essere risolto efficientemente attraverso una riduzione al problema del cammino Euleriano:

Procedimento di risoluzione

Riduzione da Domino[1] a Cammino Euleriano:

1. Costruire un grafo dove:
   - I vertici rappresentano i numeri che compaiono sulle tessere
   - Gli archi rappresentano le tessere del domino
2. Applicare l'algoritmo di Fleury per trovare un cammino Euleriano
3. Complessità: $O(n^2)$ dove n è il numero di archi

3 Le Classi di Complessità P e NP

3.1 La Classe P

Definizione

P è la classe dei linguaggi che possono essere decisi da una macchina di Turing deterministica che impiega tempo polinomiale.
Formalmente: $P = \{L \mid L \text{ è decidibile in tempo } O(n^k) \text{ per qualche } k\}$

Esempi di problemi in P:

- Raggiungibilità in un grafo: $PATH = \{\langle G, s, t\rangle \mid G \text{ contiene un cammino da } s \text{ a } t\}$
- Numeri relativamente primi: $RELPRIME = \{\langle x, y\rangle \mid \gcd(x, y) = 1\}$
- Domino[1]
- Problema del cammino Euleriano

3.2 La Classe NP

Definizione

NP è la classe dei linguaggi che ammettono un verificatore che impiega tempo polinomiale.
Equivalentemente: è la classe dei linguaggi che possono essere decisi da una macchina di Turing non deterministica che impiega tempo polinomiale.

3.3 Verificatori

Definizione

Un **verificatore** per un linguaggio A è un algoritmo V tale che:

$$A = \{w \mid V \text{ accetta } \langle w, c\rangle \text{ per qualche stringa } c\}$$

dove c è chiamato **certificato** e il verificatore usa queste ulteriori informazioni per stabilire se w appartiene al linguaggio.

Esempi di problemi in NP:

- Circuito Hamiltoniano: $HAMILTON = \{\langle G\rangle \mid G \text{ ha un circuito Hamiltoniano}\}$
- Numeri composti: $COMPOSITES = \{\langle x\rangle \mid x = pq \text{ con } p, q > 1\}$
- Domino[2]

Suggerimento

Per ogni problema in NP, se qualcuno ci fornisce una possibile soluzione (certificato), possiamo verificare in tempo polinomiale se è corretta, anche se trovare la soluzione potrebbe richiedere tempo esponenziale.

4 Riduzioni Polinomiali

4.1 Definizione di Riduzione

Definizione

Un linguaggio A è **riducibile in tempo polinomiale** al linguaggio B (scritto $A \leq_P B$), se esiste una funzione calcolabile in tempo polinomiale $f : \Sigma^* \to \Sigma^*$ tale che:

$$\forall w \in \Sigma^* : w \in A \iff f(w) \in B$$

La funzione f è detta **riduzione polinomiale** da A a B.

Concetto chiave

Se $A \leq_P B$ e $B \in P$, allora $A \in P$.
Questo significa che se un problema A è riducibile a un problema B, allora B è almeno tanto difficile quanto A.

5 NP-Completezza

5.1 Il "Re dei Problemi NP": CircuitSAT

Definizione

Il problema **CircuitSAT** è definito come:

$$\text{CircuitSAT} = \{\langle C \rangle \mid C \text{ è un circuito booleano soddisfacibile}\}$$

dove un circuito è soddisfacibile se esistono valori di input che producono output = 1.

Teorema

[Cook-Levin, 1973] CircuitSAT è NP-completo. L'esistenza di una macchina di Turing polinomiale per risolvere CircuitSAT implica che $P = NP$.

5.2 Problemi NP-Hard e NP-Completi

Definizione

Un problema è **NP-hard** se l'esistenza di un algoritmo polinomiale per risolverlo implica l'esistenza di un algoritmo polinomiale per ogni problema in NP.
Un problema è **NP-completo** se è sia NP-hard che appartenente alla classe NP.

5.3 Catena di Riduzioni Classiche

Procedimento di risoluzione

Catena di riduzioni fondamentali:

1. **CircuitSAT** $\leq_P$ **SAT**:
   - Dare nomi agli output delle porte logiche
   - Scrivere espressioni booleane per ogni porta
   - Combinarle in congiunzione logica
2. **SAT** $\leq_P$ **3SAT**:
   - Convertire ogni porta in CNF usando regole standard
   - Trasformare in 3-CNF aggiungendo variabili ausiliarie
3. **3SAT** $\leq_P$ **MaxIndSet**:
   - Creare 3 vertici per clausola (uno per letterale)
   - Archi di clausola: collegano letterali nella stessa clausola
   - Archi di consistenza: collegano letterali contraddittori

6 Esempi di Problemi NP-Completi

6.1 Problemi su Grafi

Insieme Indipendente Massimo:

$$\text{MaxIndSet} = \{\langle G, k\rangle \mid G \text{ ha un insieme indipendente di dimensione } k\}$$

Copertura di Vertici Minima:

$$\text{MinVertexCover} = \{\langle G, k\rangle \mid G \text{ ha una copertura di vertici di dimensione } k\}$$

Concetto chiave

Esiste una relazione diretta: I è un insieme indipendente in G se e solo se $V \setminus I$ è una copertura di vertici di G.

Circuito Hamiltoniano:

$$\text{HamCycle} = \{\langle G\rangle \mid G \text{ ha un circuito Hamiltoniano}\}$$

6.2 Problemi di Soddisfacibilità

Soddisfacibilità Booleana:

$$\text{SAT} = \{\langle \phi\rangle \mid \phi \text{ è una formula booleana soddisfacibile}\}$$

3-Soddisfacibilità:

$$\text{3SAT} = \{\langle \phi\rangle \mid \phi \text{ è una formula in 3-CNF soddisfacibile}\}$$

Suggerimento

Curiosamente, 2SAT è in P! Esiste un algoritmo polinomiale per la soddisfacibilità di formule in 2-CNF, che usa la propagazione di vincoli.

7 Strategie per Dimostrare NP-Completezza

7.1 Schema Standard di Dimostrazione

Procedimento di risoluzione

Per dimostrare che un problema B è NP-completo:

1. **Mostrare che** $B \in NP$: Costruire un verificatore polinomiale per B
2. **Mostrare che** B **è NP-hard**: Ridurre un problema già noto come NP-hard a B
3. **Descrivere la riduzione**: Spiegare come trasformare istanze del problema noto in istanze di B
4. **Dimostrare la correttezza**: Mostrare che la riduzione preserva soluzioni positive e negative
5. **Analizzare la complessità**: Verificare che la riduzione operi in tempo polinomiale

7.2 Scegliere il Problema di Partenza

Suggerimento

Linee guida per scegliere da quale problema NP-completo ridurre:

- Se il problema richiede di assegnare bit agli oggetti: usa SAT
- Se richiede di assegnare etichette o partizionare: usa 3-Coloring
- Se richiede di organizzare oggetti in ordine: usa Circuito Hamiltoniano
- Se richiede di trovare un piccolo sottoinsieme: usa MinVertexCover
- Se richiede di trovare un grande sottoinsieme: usa MaxIndependentSet
- Se il numero 3 appare naturalmente: prova 3SAT o 3-Coloring
- In caso di dubbio: usa 3SAT (è molto versatile)!

8 Dualità tra Problemi Facili e Difficili

8.1 Contrasti Sorprendenti

La teoria della NP-completezza rivela contrasti affascinanti tra problemi apparentemente simili:

Problemi NP-Completi	Problemi in P
Circuito Hamiltoniano (visita ogni vertice una volta)	Circuito Euleriano (visita ogni arco una volta)
Copertura di vertici (min. insieme di vertici)	Copertura di archi (max. matching)
3-colorazione di grafi (3 colori)	2-colorazione di grafi (bipartizione)

9 Implicazioni Pratiche

9.1 Perché Studiare la NP-Completezza?

Concetto chiave

Stabilire che un problema è NP-completo costituisce una prova forte della sua intrattabilità computazionale. Questo suggerisce di:

- Identificare casi particolari trattabili
- Cercare soluzioni approssimate
- Usare euristiche invece di algoritmi esatti
- Non perdere tempo cercando algoritmi polinomiali che probabilmente non esistono

9.2 La Questione del Millennio: P vs NP

Teorema

[Problema del Millennio] La questione se $P = NP$ è uno dei sette Problemi del Millennio del Clay Institute, con una taglia di \$1.000.000 per la risoluzione.

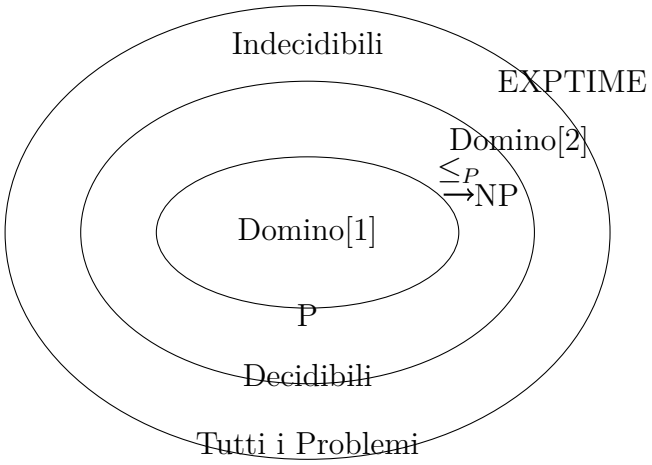
Errore comune

Un errore comune è pensare che tutti i problemi computazionali possano essere risolti con algoritmi sufficientemente sofisticati o computer più potenti. La teoria della NP-completezza dimostra invece che esistono limitazioni fondamentali intrinseche a certi problemi.

10 Conclusioni

10.1 Gerarchia della Complessità

La teoria della complessità computazionale ci ha mostrato una gerarchia di problemi:



10.2 Lezioni Apprese

Concetto chiave

Le lezioni principali di questo studio sono:

1. Non tutti i problemi computazionali sono uguali
2. Piccole modifiche a un problema possono cambiarne drasticamente la difficoltà
3. Le riduzioni sono uno strumento potente per classificare i problemi
4. La teoria della complessità guida la progettazione di algoritmi pratici
5. Capire i limiti computazionali è importante quanto trovare algoritmi efficienti

La distinzione tra P e NP rimane una delle questioni più profonde dell'informatica teorica, con implicazioni che vanno dalla crittografia all'ottimizzazione, dall'intelligenza artificiale alla biologia computazionale.