APA Modulo 1 Lezione 11

Elena Zucca

1 aprile 2020

Teoria della NP-completezza

- significa "Non deterministic Polinomial time"
  classe NP fu originariamente studiata nel contesto degli algoritmi non deterministici
- noi utilizzeremo nozione equivalente più semplice, quella di verifica
- i problemi vengono suddivisi in termini di complessità computazionale in due classi principali:
  - quelli risolvibili con un algoritmo polinomiale ($T(n) = O(n^k)$) sono considerati trattabili (facili)
  - quelli per cui non esiste un algoritmo polinomiale ($T(n) = \Omega(k^n)$) non trattabili (difficili)

Questo perché:

- solitamente le complessità $O(n^k)$ hanno k piccoli, e possono essere ulteriormente migliorate
- per diversi modelli di calcolo, un problema che può essere risolto in tempo polinomiale in un modello, può esserlo anche in un altro
- la classe dei problemi risolvibili in tempo polinomiale ha utili proprietà di chiusura
  i polinomi sono chiusi per addizione, moltiplicazione e composizione
- per esempio se l'output di un algoritmo polinomiale è utilizzato come input per un altro algoritmo polinomiale
  l'algoritmo composto è polinomiale

Quello che vedremo, informalmente

- classe P = problemi risolvibili in tempo polinomiale
- classe NP = problemi verificabili in tempo polinomiale
- risolvere un problema = data un'istanza fornire una soluzione
- verificare un problema = data un'istanza e una possibile soluzione, controllare se questa risolve effettivamente l'istanza del problema
- intuitivamente P $\subseteq$ NP
  se so risolvere un problema in tempo polinomiale so anche verificarlo in tempo polinomiale

Quello che vedremo, informalmente

- problema aperto: $\mathrm{P} = \mathrm{NP}$ oppure $\mathrm{P} \subset \mathrm{NP}$?
- esistono problemi verificabili in tempo polinomiale che non sono risolvibili in tempo polinomiale?
- all'interno della classe NP vi è una classe di problemi "più difficili" di tutti gli altri
- questa è la classe dei problemi NP-completi (NP-C)
- sappiamo risolvere questi problemi in tempo esponenziale, ma non sappiamo se esistono algoritmi in tempo polinomiale per risolverli
- "più difficili" significa: se si scoprisse un algoritmo che risolve uno di questi problemi in tempo polinomiale, tutti i problemi di NP sarebbero risolvibili in tempo polinomiale
- si dimostrerebbe quindi che $\mathrm{P} = \mathrm{NP}$

Nel seguito

- formalizziamo la nozione di problema
- mostriamo come astrarre dal particolare linguaggio usato per descrivere il problema
- formalizziamo le classi P, NP, NP-C
- troviamo un (primo) problema in NP-C
- descriviamo altri problemi in NP-C

Problemi astratti

Definizione

Un problema (astratto) è una relazione $\mathcal{P} \subseteq I \times S$, dove I è l'insieme degli input (o istanze del problema) e S è l'insieme delle (possibili) soluzioni.

In generale per ogni istanza la soluzione può non essere unica (per esempio può esserci più di un cammino minimo).

Problemi di decisione

Definizione

Un problema (astratto) di decisione $\mathcal{P}$ è un problema (astratto) in cui ogni input ha come soluzione vero oppure falso, ossia $\mathcal{P}\colon I \to \{T, F\}$.

Perché consideriamo problemi di decisione?

- problemi di ricerca = si cerca una soluzione
  per esempio: ricerca binaria che restituisce indice, ordine topologico
- problemi di ottimizzazione = ci sono diverse soluzioni e se ne cerca una che sia "ottima"
  per esempio: minimo albero ricoprente, cammini minimi
- teoria della complessità computazionale su problemi di decisione
- perché: dato un problema di altro tipo $\mathcal{P}$, è possibile dare un problema di decisione $\mathcal{P}_d$ tale che risolvendo il primo si sappia risolvere il secondo
- se problema di ricerca: vero se e solo se una soluzione esiste
- se problema di ottimizzazione: si pone limitazione al valore da ottimizzare

Perché consideriamo problemi di decisione?

- esempio: trovare il cammino minimo tra una coppia di nodi in un grafo è un problema di ottimizzazione
- problema di decisione corrispondente: dati due nodi determinare se esiste un cammino lungo al più k
- se abbiamo un algoritmo per risolvere il primo problema, un algoritmo che risolve il secondo si ottiene eseguendo il primo e poi confrontando il valore minimo ottenuto con k
- il problema di decisione $\mathcal{P}_d$ è più facile di $\mathcal{P}$
- se proviamo che $\mathcal{P}_d$ è "difficile" indirettamente proviamo che lo è anche $\mathcal{P}$
- da ora in poi quindi parleremo semplicemente di "problema" intendendo problema di decisione

Algoritmo che risolve un problema

- non ci interessa fissare un particolare formalismo (per esempio i programmi in un certo linguaggio) per esprimere gli algoritmi
- quindi algoritmo = funzione $A \colon I \to \{T, F\}$
  in generale parziale, l'algoritmo potrebbe non terminare per qualche input
- dato un problema $\mathcal{P} \colon I \to \{T, F\}$, un algoritmo A risolve $\mathcal{P}$ se:
  per ogni input $i \in I$, $A(i) = \mathcal{P}(i)$

Classi di complessità di problemi

- un problema $\mathcal{P}$ è nella classe $\mathsf{Time}(f(n))$ se e solo se esiste un algoritmo di complessità temporale $O(f(n))$ che lo risolve, dove n è la dimensione dell'input
- analogamente, $\mathcal{P}$ è nella classe $\mathsf{Space}(f(n))$ se e solo se esiste un algoritmo di complessità spaziale $O(f(n))$ che lo risolve
- $\mathsf{P} = \bigcup_{k \geq 0} \mathsf{Time}(n^k)$
- $\mathsf{PSpace} = \bigcup_{k \geq 0} \mathsf{Space}(n^k)$
- $\mathsf{ExpTime} = \bigcup_{k \geq 0} \mathsf{Time}(2^{n^k})$

Proprietà (con giustificazione informale)

- P $\subseteq$ PSpace
  un algoritmo che impiega un tempo polinomiale può accedere al più a un numero polinomiale di locazioni di memoria
- PSpace $\subseteq$ ExpTime
  assumendo per semplicità locazioni di memoria binarie, n^c locazioni di memoria possono trovarsi in al più 2^{n^c} stati diversi
- non si sa se queste inclusioni siano strette (sono problemi aperti)
- P $\subset$ ExpTime: esistono problemi che possono essere risolti in tempo esponenziale ma non polinomiale, quindi provabilmente intrattabili
  per esempio, le torri di Hanoi

Problemi concreti

- le definizioni precedenti sono semi-formali nel senso che non sappiamo cosa sia la "dimensione" dell'input
- per essere più precisi, possiamo uniformare tutti i possibili input codificandoli come stringhe binarie

Definizione

Un problema concreto $\mathcal{P}$ è un problema il cui insieme di istanze è l'insieme delle stringhe binarie, ossia $\mathcal{P}\colon \{0,1\}^\star \to \{T, F\}$.

- è equivalente considerare qualunque alfabeto con cardinalità almeno 2

Codifica

- un problema astratto $\mathcal{P}\colon I \to \{T, F\}$ può essere rappresentato in modo concreto tramite una codifica, ossia una funzione iniettiva:

  $$c\colon I \to \{0,1\}^\star$$

- il problema concreto $c(\mathcal{P})\colon \{0,1\}^\star \to \{T, F\}$ è definito da
- $c(\mathcal{P})(x) = T$ se e solo se $x = c(i)$ e $\mathcal{P}(i) = T$
- ossia, assumiamo convenzionalmente che la soluzione sia falso sulle stringhe che non sono codifica di nessun input

Introduciamo il significato della classe NP con un esempio

le formule in forma normale congiuntiva (CNF) e le formule quantificate sono definite nel modo seguente:

$$
\begin{array}{llll}
l & ::= & x \mid \overline{x} & \textit{letterale} \\
c & ::= & l_1 \vee \ldots \vee l_n & \textit{clausola} \\
\phi_{CNF} & ::= & c_1 \wedge \ldots \wedge c_n & \textit{formula in } \mathrm{CNF} \\
\phi_Q & ::= & \phi_{CNF} \mid \exists x.\phi_Q \mid \forall x.\phi_Q & \textit{formula quantificata}
\end{array}
$$

Esempi

- formula in CNF:
  $x \wedge (\overline{y} \vee z) \wedge (\overline{x} \vee w \vee y \vee \overline{z}) \wedge (\overline{w} \vee x)$
- formula quantificata:
  $\forall x.\exists y.\exists z.(x \vee z) \wedge y$

Problema della soddisfacibilità *SAT*

- data una formula in forma normale congiuntiva, determinare se esiste un'assegnazione di valori di verità alle variabili che la renda vera
- esempio: $x \wedge (\overline{y} \vee z) \wedge (\overline{x} \vee w \vee y \vee \overline{z}) \wedge (\overline{w} \vee x)$ è soddisfacibile
- $x = T$, $y = T$, $z = T$, w qualunque
- $x \wedge (\overline{x} \vee y) \wedge (\overline{y} \vee z) \wedge \overline{z}$ non è soddisfacibile

Problema delle formule quantificate

- data una formula quantificata, determinare se è vera
- esempio: $\forall x.\exists y.\exists z.(x \vee z) \wedge y$ è vera
- infatti: se $x = T$, $(x \vee z) \wedge y$ è vera per $y = T$ e z qualunque
- se $x = F$, $(x \vee z) \wedge y$ è vera per $y = T$ e $z = T$
- invece: $\forall x.\exists y.(x \vee y) \wedge \overline{y}$ non è vera

Algoritmo esponenziale

- prendiamo come dimensione dell'istanza del problema il numero di variabili
- per decidere se una formula CNF è soddisfacibile, proviamo a valutarla per tutte le possibili assegnazioni di valori di verità alle variabili
- sono 2^n

Pseudocodice ricorsivo

env (environment) = assegnazione di valori di verità alle variabili

```
eval(∃x.ϕ,env)
  return eval(ϕ,env.add(x,true))
    or eval(ϕ,env.add(x,false))

eval(∀x.ϕ,env) =
  return eval(ϕ,env.add(x,true)
    and eval(ϕ,env.add(x,false))

eval(ϕ1 ∧ ... ∧ ϕn, env) =
  eval(ϕ1,env) and ... and eval(ϕ_n,env)

eval(ϕ1 ∨ ... ∨ ϕn, env) =
  eval(ϕ1,env) or ... or eval(ϕn,env)
```

Pseudocodice ricorsivo

env (environment) = assegnazione di valori di verità alle variabili

```
eval(x,env) = env(x)
eval(x̄, env) = not eval(x,env)
```

```
sat(φ) //con variabili x1 ... xn
  return eval(∃x1....∃xn.φ)
```

Cosa mostra questo esempio

- spesso un algoritmo di decisione genera, in caso positivo, una "prova", detta certificato, che dimostra la verità della proprietà da verificare
- per esempio, nel caso della soddisfacibilità, l'assegnazione di valori alle variabili che rende vera la formula
- nel caso della soddisfacibilità verificare la validità di un certificato è facile

```
sat_ver(φ,env)
  return eval(φ,env)
```

- nel caso delle formule quantificate il "certificato" stesso consta di un numero esponenziale di assegnazioni di valori di verità a variabili
- questo suggerisce l'idea di usare come classificazione la difficoltà di verificare se un certificato è valido per un problema
- informalmente, definiamo NP la classe dei problemi che ammettono certificati verificabili in tempo polinomiale

Quindi:

- dato che possiamo verificare in tempo polinomiale se un'assegnazione di valori alle variabili rende vera una formula in forma normale congiuntiva
- il problema della soddisfacibilità è nella classe NP
- mentre non possiamo dirlo per il problema delle formule quantificate
- altro esempio: problema del ciclo hamiltoniano in un grafo non orientato
- è un ciclo semplice che contiene ogni nodo (quindi passa esattamente una volta per ogni nodo)
- questo problema è in NP (un certificato è un ciclo), mentre non è noto un algoritmo polinomiale che lo risolva

Algoritmo di verifica

Definizione

Un algoritmo di verifica per un problema (astratto) $\mathcal{P} \subseteq I$ è un algoritmo $A \colon I \times C \to \{T, F\}$, dove C è un insieme di certificati, e

$$A(x,y) = T \text{ per qualche } y \text{ se e solo se } x \in \mathcal{P}.$$

Nel caso dei problemi concreti, si ha $I = C = \{0,1\}^\star$.

Classe NP

Classe dei problemi che possono essere verificati da un algoritmo polinomiale. Più precisamente:

Definizione

Un problema $\mathcal{P}$ è nella classe NP se e solo se esistono un algoritmo di verifica polinomiale A e una costante $k > 0$ tali che

$$\mathcal{P} = \{x \mid \exists y.A(x, y) = T, |y| = O(|x|^k)\}$$

Definizione equivalente

- si può anche definire NP come la classe dei problemi per cui esiste un algoritmo polinomiale non deterministico che li risolve
- cioè , la risposta è positiva se c'è almeno una computazione con risposta positiva

```
sat_non_det(φ) //con variabili x₁ ... xₙ
  b₁ = true or b₁ = false
  ...
  bₙ = true or bₙ = false
  return eval(φ, x₁ -> b₁, ..., xₙ -> bₙ)
```

- legame con la verifica in tempo polinomiale (informalmente): un algoritmo non deterministico può sempre essere visto come composto di due fasi (vedi esempio sopra):
- una prima fase non deterministica che costruisce un certificato
- una seconda fase deterministica che controlla se è valido

- è facile vedere che P $\subseteq$ NP, perché un algoritmo deterministico è un caso particolare di algoritmo non deterministico
- oppure equivalentemente è un caso particolare di algoritmo di verifica in cui si ignora il certificato