

# Informatica Teorica



Anno Accademico 2022/2023  
Fabio Zanasi  
<https://www.unibo.it/sitoweb/fabio.zanasi>  
Quattordicesima lezione

# Nelle puntate precedenti

- Abbiamo studiato la complessità computazionale relativamente a modelli di calcolo (algoritmi) **deterministici** e **non-deterministici**.

# In questa lezione

- Introduciamo la **TM probabilistica** e la relativa classe **BPP**.
- Introduciamo il concetto di **Interactive proof** (dimostrazione interattiva).
- Da questi, approdiamo al concetto di **zero-knowledge proof**, fondamentale in crittografia (autenticazione, privacy, blockchain, ....).

Avvertenza: questa lezione non è parte del programma d'esame. Saremo un po' più informali, al fine di coprire argomenti più avanzati e interessanti.

# La classe BPP

# Algoritmi probabilistici: perché?

Storicamente:

Lo studio dei processi (pensiamo alla fisica e biologia, ma anche alla statistica e la finanza) non può prescindere da una descrizione **probabilistica** dei fenomeni.

Questo é riconosciuto nella matematica applicata da lungo tempo: pensiamo ad esempio ai metodi ‘Monte-Carlo’ nella teoria delle equazioni differenziali

Di particolare attualità, relativamente all'informatica:

Cybersecurity

Intelligenza artificiale (Machine Learning)

# Macchine di Turing probabilistiche

Una macchina di Turing probabilistica (PTM) è una TM con due funzioni di transizione,  $\delta_1$  e  $\delta_2$ .

Ad ogni passo di esecuzione, utilizziamo  $\delta_1$  con probabilità 1/2 e  $\delta_2$  con probabilità 1/2.

# Probabilità vs. Non-determinismo

Come per le TM non-deterministiche, l'esecuzione di una PTM dà luogo ad un **albero** di computazione.

La differenza sta in **come** interpretiamo tale albero.

Nel non-determinismo, ci interessa se **esiste un ramo accettante**.

Nella computazione probabilistica, ci interessa **la probabilità che l'input venga accettato**. Questa probabilità sarà una frazione del numero dei rami accettanti.

Concettualmente, le PTM sono un modello più realistico.

# Quando una PTM accetta?

Data l'esecuzione di una PTM  $M$  su input  $w$ , definiamo la probabilità di un ramo di computazione  $b$ :

$$Pr[b] = 2^{-k}$$

dove  $k$  è il numero di scelte probabilistiche effettuate lungo  $b$ . Definiamo la **probabilità che  $M$  accetti  $w$**  come:

$$Pr[M \text{ accetta } w] = \sum_b_{\text{accettante}} Pr[b]$$

# Complessità di tempo: la classe BPP

Data una funzione  $T : \mathbb{N} \rightarrow \mathbb{N}$ , diciamo che una PTM M **lavora in tempo  $T(n)$**  se, su qualsiasi input di lunghezza  $n$ , M ferma in al più  $T(n)$  passi, indipendentemente dalle scelte probabilistiche effettuate durante la computazione.

Diciamo **un linguaggio  $L$  è in  $BPTIME(T(n))$**  se esiste una PTM che lavora in tempo  $O(T(n))$  tale che:

- $w \in L$  implica  $\Pr[M \text{ accetta } w] \geq 2/3$
- $w \notin L$  implica  $\Pr[M \text{ rigetta } w] \geq 2/3$

$$BPP = \bigcup_k BPTIME(n^k)$$

# Alcune considerazioni

La definizione è robusta: possiamo rimpiazzare  $2/3$  con qualsiasi costante superiore a  $1/2$  senza alterare la classe dei linguaggi  $\text{BPTIME}(T(n))$  e  $\text{BPP}$ .

Come per la classe  $\text{NP}$ , esiste una definizione di  $\text{BPP}$  alternativa, basata su verificatori. In questo caso, il certificato sarà una sequenza di valori in  $[0,1]$ , indicativa di un percorso di computazione in una PTM.

Alcuni esempi significativi di problemi  $\text{BPP}$ :

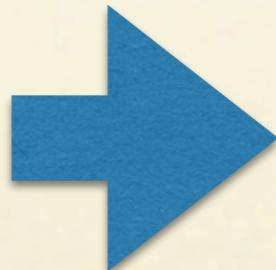
- verificare se un numero è primo (un algoritmo  $P$  è stato scoperto solo di recente).
- verificare se due polinomi sono equivalenti (un algoritmo  $P$  non è noto).

**BPP = P?** ... (sorprendentemente?) molti pensano di sì.

# Interactive and Zero-Knowledge Proofs

# Interactive proofs: cosa sono?

Finora, la cosa più vicina ad una **dimostrazione** che abbiamo osservato nello studio della complessità è la nozione di **certificato** nella definizione di un verificatore NP.



Prover  
(Soggetto che produce  
la dimostrazione)

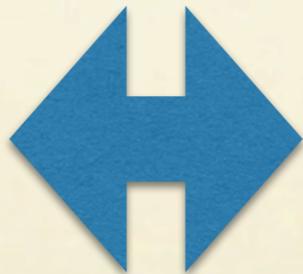
Verifier  
(Soggetto che verifica la  
correttezza della  
dimostrazione)

# Interactive proofs: cosa sono?

L'idea delle interactive proofs (IP) è di modellare una nozione di verifica più generale, dove il prover ed il verifier possono avere **una serie di interazioni** prima che il verifier sia convinto della bontà o meno della dimostrazione.



Prover  
(Soggetto che produce  
la dimostrazione)



Verifier  
(Soggetto che verifica la  
correttezza della  
dimostrazione)

# Interactive proofs: perché?

Perché tutto ciò dovrebbe essere più interessante di quanto abbiamo già studiato con i verificatori?

È stato scoperto che, nel momento in cui ammettiamo che il processo di verifica sia interattivo, è possibile definire algoritmi efficienti per problemi altrimenti ‘difficili’.

Esempio: il complemento di SAT è coNP-completo (ed in PSPACE). Riteniamo non abbia dimostrazioni di lunghezza polinomiale nel senso tradizionale, ma abbiamo scoperto che ha *interactive proofs* di lunghezza polinomiale.

Un altro esempio è TQBF.

Oltre lo studio della complessità, le interactive proofs si sono dimostrate molto importanti nello sviluppo di protocolli in crittografia, lo studio di algoritmi di approssimazione, e molti altri campi.

# Da NP a IP: l'idea generale

Nello scenario NP, il verifier analizza la dimostrazione trasmessa per iscritto dal prover.

Nello scenario IP, il verifier interroga il prover quante volte vuole, ed ogni domanda può dipendere dalle risposte precedenti. Alla fine, il prover decide se accettare o meno l'input.

Il verifier dovrebbe essere deterministico o probabilistico?

Il prover dovrebbe essere deterministico o probabilistico?

Se ammettiamo probabilità, come definiamo l'accettazione?

## Passo preliminare: Interazione in k round

Date  $V, P: \{0,1\}^* \rightarrow \{0,1\}^*$  e  $k \geq 0$ , scriviamo  $\langle V, P \rangle(x)$  per una **interazione in  $k$  round di  $V$  e  $P$  su input  $x \in \{0,1\}^*$** , definita come la sequenza di stringhe

$$a_1 = V(x)$$

$$a_2 = P(x, a_1)$$

...

$$a_{2i+1} = V(x, a_1, \dots, a_{2i}) \quad \text{per ogni } 2i < k$$

$$a_{2i+2} = P(x, a_1, \dots, a_{2i+1}) \quad \text{per ogni } 2i + 1 < k$$

Scriviamo  $\text{out}_V^k \langle V, P \rangle(x)$  per **l'output dell'interazione in  $k$  round di  $V$  e  $P$** , definito come  $V(x, a_1, \dots, a_k)$ . Assumiamo che questo output sia in  $\{0,1\}$ .

# Interactive proofs deterministiche

Un linguaggio  $L$  **ha un interactive proof system deterministico su  $k$  round**, scritto  $L \in dIP(k)$ , se esiste una TM deterministica  $V$  che su input  $x, a_1, \dots, a_k$  lavora in tempo polinomiale in  $x$  e tale che:

**(Completezza)**  $x \in L \Rightarrow \exists P: \{0,1\}^* \rightarrow \{0,1\}^*. \text{out}_V^k \langle V, P \rangle(x) = 1$

**(Correttezza)**  $x \notin L \Rightarrow \forall P: \{0,1\}^* \rightarrow \{0,1\}^*. \text{out}_V^k \langle V, P \rangle(x) = 0$

$$dIP := \bigcup_{c \geq 1} dIP(n^k)$$

# Interactive proofs deterministiche

**Esempio (sketch):** in un interactive proof system deterministico per 3SAT il verifier chiederà, per ciascuna clausola in una data formula, che il prover annunci il valore di verità di ciascun letterale nella formula. Accetta se ciascuna clausola è soddisfacibile e nessuna risposta è in contraddizione con le precedenti (perché ad esempio assegna a y valore sia vero che falso).

**Osservazione:** l'interazione è data dal fatto che la richiesta viene fatta clausola per clausola. Tuttavia, il verifier avrebbe anche potuto chiedere conto di tutte le clausole in una volta sola. Infatti, più in generale, abbiamo:

$$\mathbf{dIP} = \mathbf{NP}$$

# Interactive proofs probabilistiche

Perciò il determinismo annulla qualsiasi vantaggio che potrebbe essere dato dall'interazione di verifier e prover.

Per esprimere il vero potenziale delle interactive proofs, dobbiamo considerare la loro variante **probabilistica**.

# Interazione in $k$ round (probabilistica)

Dati  $m \geq 1$ , e  $V: (\{0,1\}^{\star} \times [0,1]^m) \rightarrow \{0,1\}^{\star}$ ,  
 $P: \{0,1\}^{\star} \rightarrow \{0,1\}^{\star}$  e  $k \geq 0$ , scriviamo  $\langle V, P \rangle(x)$  per una  
**interazione (probabilistica) in  $k$  round di  $f$  e  $g$**  su input  
 $x \in \{0,1\}^{\star}$  e random m-bit  $r \in [0,1]^m$ , definita come la  
sequenza di stringhe

$$a_1 = V(x, r)$$

$$a_2 = P(x, a_1)$$

...

$$a_{2i+1} = V(x, a_1, \dots, a_{2i}, r) \quad \text{per ogni } 2i < k$$

$$a_{2i+2} = P(x, a_1, \dots, a_{2i+1}) \quad \text{per ogni } 2i + 1 < k$$

Scriviamo  $\text{out}_f^k \langle f, g \rangle(x)$  per **l'output**, definito come  
 $V(x, a_1, \dots, a_k, r)$ . Assumiamo che questo output sia in  $\{0,1\}$ .

# Interactive proofs probabilistiche

Un linguaggio  $L$  **ha un interactive proof system su  $k$  round**, scritto  $L \in IP(k)$ , se esiste una PTM  $V$  che su input  $x, a_1, \dots, a_k$  lavora in tempo polinomiale in  $x$  e tale che:

**(Completezza)**

$$x \in L \Rightarrow \exists P: \{0,1\}^* \rightarrow \{0,1\}^* . Pr[\text{out}_V^k \langle V, P \rangle(x) = 1] \geq 2/3$$

**(Correttezza)**

$$x \notin L \Rightarrow \forall P: \{0,1\}^* \rightarrow \{0,1\}^* . Pr[\text{out}_V^k \langle V, P \rangle(x) = 0] \geq 2/3$$

$$IP := \bigcup_{c \geq 1} IP(n^k)$$

IP(0) = BPP

# Un esempio di interactive proof system



Morpheus (prover)

Neo è daltonico. Come fa Morpheus a convincerlo che le due pillole hanno colore diverso?



Neo (verifier)

1. Morpheus mette la pillola rossa nella mano sinistra di Neo, e l'azzurra nella destra, dicendogli di che colore sono.
2. Morpheus si gira di spalle e Neo lancia una moneta. Se esce testa, Neo tiene le pillole nelle mani di partenza, se esce croce le scambia.
3. Neo chiede a Morpheus di girarsi e dire se ha scambiato le pillole o meno. Naturalmente Morpheus può farlo facilmente.
4. Se le pillole fossero dello stesso colore, Morpheus avrebbe solo il 50% di possibilità di dare la risposta corretta. Perciò Neo ripete la procedura 2-3 un numero sufficiente di volte per convincersi che Morpheus non sta tirando a indovinare.

# Un esempio di interactive proof system

Questa idea si può applicare allo sviluppo di interactive proof systems per la soluzione di diversi problemi (in PSPACE) che sono il complemento di problemi in NP.

- Decidere se una formula non è soddisfacibile
- Decidere se due grafi non sono isomorfi
- ...

In generale, abbiamo:

$$\mathbf{IP} = \mathbf{PSPACE}$$

# Alcune considerazioni

Permettere la randomizzazione anche al prover non produce una classe diversa da IP.

Sostituire la costante  $2/3$  con  $1$  nel requisito di completezza non produce una classe diversa da IP.

Sostituire la costante  $1/3$  con  $0$  nel requisito di correttezza ci fa `collassare' nella classe NP (il verifier diventa deterministico).

# Alcune considerazioni

Nonostante il prover sia una funzione arbitraria, possiamo dimostrare che, dato un verifier  $V$ , un prover ottimale (= che massimizza la probabilità che  $V$  accetti) può essere calcolato in spazio polinomiale.

In IP la funzione del prover dipende solo dal messaggio del verificatore, non dal risultato del lancio di moneta utilizzato dal verificatore per produrlo.

Questa variante è detta *private-coin* interactive proofs. Sono studiate anche le *public-coin* interactive proofs (dette anche Arthur-Merlin proofs).

# Zero-knowledge proofs: idea

Solitamente esaminare la dimostrazione di un enunciato ci arricchisce di **più informazioni** della semplice asserzione che l'enunciato è vero.

È una situazione sempre desiderabile?



# Zero-knowledge proofs: idea

Prendiamo lo scenario di un'azienda che ha un magazzino con accesso riservato solo a personale selezionato.



Al personale selezionato vengono rivelati due numeri primi  $P$  e  $Q$ , generati a caso.

Alla guardia all'ingresso del magazzino viene rivelato solo il numero  $N = P \times Q$ .

È possibile per un membro del personale dimostrare al guardiano che sa i numeri di accesso senza rivelare quali sono?

# Zero-knowledge proofs: idea

È possibile utilizzando una *zero-knowledge proof* (ZKP).

Le ZKP sono interactive proof systems (probabilistici), a cui viene aggiunta una clausola, di *zero-knowledge*, la quale richiede che il verifier non apprenda alcuna informazione dall'interazione a parte il fatto che l'enunciato sia vero secondo il prover.

# Zero-knowledge proofs: definizione

Supponiamo che  $P$  sia in NP, quindi che lavori su input  $(x, u)$  dove  $u$  è un certificato per  $x$ .

## (Completezza)

$$x \in L \Rightarrow \exists P: \{0,1\}^* \rightarrow \{0,1\}^*. \Pr[\text{out}_V^k \langle V, P \rangle(x) = 1] \geq 2/3$$

## (Correttezza)

$$x \notin L \Rightarrow \forall P: \{0,1\}^* \rightarrow \{0,1\}^*. \Pr[\text{out}_V^k \langle V, P \rangle(x) = 0] \geq 2/3$$

## (Perfect zero-knowledge)

$$\forall P: \{0,1\}^* \rightarrow \{0,1\}^*. \exists S. \Pr[\text{out}_V^k \langle V, P \rangle(x) = 0] \equiv S(x)$$

$S$  è una PTM polinomiale

Sono variabili aleatorei identicamente distribuite sebbene  $S$ , a differenza di  $P$ , non abbia accesso ad alcun certificato/dimostrazione di  $x$ .

# Zero-knowledge proofs: applicazioni

- **Autenticazione, privacy**
- **Tecnologie per la blockchain**
- ...
- **Disarmo nucleare?**



# L'unico Turing Award italiano



Shafi Goldwasser e Silvio Micali, vincitori del Turing Award (2012) per i loro contributi alla crittografia, tra cui l'idea di utilizzare le ZKP nell'ambito della sicurezza informatica.