Informatica & filosofia

Angelo Montanari
Dipartimento di Matematica, Informatica e Fisica
Università degli Studi di Udine

Udine, 18 aprile, 2020

Artificial Intelligence is in the zeitgeist

- Public Lecture at Knowledge Representation (KR) 2016: **Will AI end jobs, wars or humanity?** Toby Walsh (NICTA, University of New South Wales, Australia), Cape Town, SA, April 26, 2016

- **AI is definitely in the zeitgeist.**

The Chief Economist of the Bank of England some years ago predicted **AI will destroy 50% of jobs in the UK**.

Thousands of AI researchers signed an Open Letter predicting that AI could transform warfare and lead to an arms race of ``**killer robots**".

Stephen Hawking and others have predicted that **AI could end humanity itself**.

- What should we make of all these predictions, and what should we do to ensure a safe and prosperous future for all?

Intelligenza Artificiale: apocalittici e entusiasti

- E' possibile stendere un elenco di **applicazioni** presenti e future dell'IA che copre tutte le lettere dell'alfabeto, dalle **a**uto senza pilota alla **b**ionica, dalla **d**omotica alla (nuova) **r**obotica industriale.
- **Apocalittici**: Stephen Hawking, ma non solo.
- **Entusiasti** (la filosofia postumanistica/transumanistica e il superamento dell'umano): Marvin Minsky (la società della mente), Raymond Kurzweil (la singolarità tecnologica).
- Per quanto rilevante, l'Intelligenza Artificiale è da sempre un settore dell'**informatica** (già presente negli scritti di Alan Turing).

Un vocabolario filosofico dell'informatica

- Per un vocabolario filosofico dell'informatica:

- **declinazioni particolari** (in ambito informatico) di termini da sempre appartenenti al vocabolario filosofico, quali **intelligenza**, conoscenza, rappresentazione, ontologia, **memoria**, linguaggio, determinismo, tempo, infinito.

- **termini caratteristici** dell'informatica "filosoficamente" rilevanti, quali algoritmo, decidibilità, modello di calcolo, complessità, trattabilità.

A. Montanari, **Per un vocabolario filosofico dell'informatica**, in: Istanze Epistemologiche e Ontologiche delle Scienze Informatiche e Biologiche, a cura di G. Cicchese, A. Pettorossi, S. Crespi Reghizzi, V. Senni, Città Nuova, 2011, pp. 82-106.

Obiettivo dei seminari

- Obiettivo del breve ciclo di seminari: uno **sguardo dall'interno** sui concetti fondamentali dell'informatica in generale, e dell'intelligenza artificiale in particolare, e sulla loro valenza filosofica.
- Necessità di colmare alcuni vuoti della riflessione sulla "filosofia del digitale".
- Esistono numerosi libri di **divulgazione** riguardanti vari **aspetti scientifici e tecnologici** dell'informatica, esistono alcuni libri di **storia** dell'informatica, ma mancano libri che ne analizzino in modo sistematico le **implicazioni filosofiche** (i contributi fondamentali dell'informatica non fanno ancora parte della cultura condivisa).

Struttura del ciclo di seminari

- **Filosofia & informatica:**

I. **le parole dell'informatica** (algoritmo, decidibilità, modello di calcolo, complessità computazionale, trattabilità)

II. **parole note, nuovi significati** (linguaggio, determinismo, tempo, infinito)

- **Intelligenza** (artificiale)
- **Memoria** (artificiale)

Un vocabolario antropomorfico

- L'uso di un **vocabolario antropomorfico** nella descrizione delle caratteristiche e del funzionamento dei sistemi informatici
  - è particolarmente evidente nel caso dei sistemi di intelligenza artificiale (**intelligenza**, conoscenza, apprendimento, ragionamento),
  - ma si è verificato in misura più o meno rilevante in molti altri ambiti dell'informatica (**memoria**, comunicazione, interrogazione).

- **Ragioni** dell'uso di un vocabolario antropomorfico
  - Uomo/animale come modello in cibernetica (Norbert Wiener: Cybernetics or Control and Communication in the Animal and the Machine, MIT Press, 1962) e successivamente in diversi settori dell'informatica (Intelligenza Artificiale, Robotica, Bionica, ..).

Norbert Wiener

Il rapporto uomo/macchina

- Osservazione: ogni **discorso** sulle proprietà "antropomorfiche" delle macchine/calcolatori non riguarda tanto la macchina (il calcolatore) in sé, ma il modo in cui noi vediamo la macchina, e indirettamente noi stessi (ad esempio, Minsky rivendica la legittimità/utilità dell'uso di termini antropomorfici).
- Ciò vale, ad esempio, per la questione relativa alla **intelligenza/intenzionalità** delle **macchine**: parlare delle macchine è un modo (indiretto) per parlare di noi stessi (è una questione antropologica).

Il rapporto uomo/macchina

- Osservazione: ogni **discorso** sulle proprietà "antropomorfiche" delle macchine/calcolatori non riguarda tanto la macchina (il calcolatore) in sé, ma il modo in cui noi vediamo la macchina, e indirettamente noi stessi (ad esempio, Minsky rivendica la legittimità/utilità dell'uso di termini antropomorfici).
- Ciò vale, ad esempio, per la questione relativa alla **intelligenza/intenzionalità** delle **macchine**: parlare delle macchine è un modo (indiretto) per parlare di noi stessi (è una questione antropologica).

La nozione chiave di algoritmo

- Il concetto di **algoritmo** non è di per sé un contributo originale dell'informatica.
- Gli algoritmi sono noti da sempre non solo in matematica (l'algoritmo per il calcolo del massimo comun divisore di due numeri interi positivi fu inventato da Euclide nel quarto secolo a.C.), ma anche in numerose altre discipline (ad esempio, in architettura).
- Ciò che è nuovo è il ruolo centrale che essi assumono nell'informatica:

teoria degli algoritmi (o algoritmica)

Che cos'è un algoritmo

- Un **algoritmo** è una descrizione finita e non ambigua di una sequenza di operazioni che consente ad un certo agente (**modello di calcolo**) di risolvere un determinato **problema** (esempi: il problema della ricerca del massimo di un insieme di numeri naturali distinti, il problema dell'ordinamento di un certo insieme di numeri, ..).

- Caratteristiche fondamentali:
  - **uniformità**: un algoritmo si applica a tutte le istanze di un dato problema, potenzialmente infinite, non ad una singola istanza, e la sua formulazione non dipende dalla singola istanza, ma è la stessa per ogni istanza;
  - **effettività**: raggiungibilità della soluzione voluta in un tempo finito (equivalentemente, in un numero finito di passi).

Esempio: ricerca del massimo - 1

- **Esempio**. Consideriamo il **problema** della ricerca del massimo di un insieme di numeri naturali distinti. Assumiamo (per semplicità) che l'insieme contenga un numero di elementi **n** che è una potenza di 2 ($n = 2^k$, per qualche $k >= 1$ – esempio: se k=3, allora n=8) e gli elementi siano contenuti in una lista non ordinata.

- Possiamo risolvere il problema usando diversi **algoritmi**.
  - **Algoritmo 1.** Controllo se il primo numero è il massimo confrontandolo con tutti gli altri. Se non lo è, controllo se il secondo numero è il massimo. E così via.
  - **Algoritmo 2.** Confronto il primo numero col secondo. Se il primo è più grande del secondo, li scambio. Confronto il secondo numero col terzo. Se il secondo è più grande del terzo, li scambio. E così via. Al termine, il numero nell'ultima posizione della lista è il massimo.

Esempio: ricerca del massimo - 2

- **Algoritmo 3.** Confronto il primo col secondo metto il più grande dei due in prima posizione. Confronto il terzo col quarto e metto il più grande dei due in terza posizione .. Confronto il penultimo con l'ultimo e metto il più grande dei due in penultima posizione. A questo punto, confronto il primo col terzo e metto il più grande dei due in prima posizione. Confronto il quinto col settimo e metto il più grande dei due in quinta posizione. Il massimo uscirà dal confronto finale tra i numeri in prima posizione e in posizione $2^{k-1} + 1$ (ad esempio, se k = 3, in quinta posizione).

Alcune osservazioni.

- Un problema, diversi algoritmi.
- Come confrontare un algoritmo con un altro?
- Come valutare la bontà/qualità di un algoritmo che risolve un dato problema?

Algoritmi e programmi

- Un **programma** è una formulazione di un algoritmo in uno specifico **linguaggio (di programmazione)** che può essere letto e interpretato da un computer.
- La formulazione di un algoritmo dipende dal soggetto/macchina (**modello di calcolo**) che lo deve eseguire: un algoritmo deve essere formulato utilizzando i comandi (istruzioni) di un linguaggio che il soggetto/macchina è in grado di comprendere ed eseguire.
- Tale dipendenza è all'origine della **gerarchia** dei linguaggi di programmazione, che spazia dai linguaggi di programmazione di alto livello, che utilizzano istruzioni di controllo complesse, ai linguaggi macchina, che utilizzano istruzioni direttamente eseguibili dalla macchina. **Compilatori/interpreti:** programmi che consentono di passare dagli uni agli altri.

Problemi e algoritmi

- La risposta alla domanda (ingenua): “esiste un algoritmo per ogni problema?” è negativa.
- L'esistenza di problemi per i quali non esistono algoritmi si può mostrare facendo vedere che esistono **funzioni non computabili** (funzioni per le quali non esiste un algoritmo in grado di computarle). Esistono, infatti, più funzioni che nomi per designarle e algoritmi per computarle. L'argomento può essere formalizzato usando il metodo della diagonale di Cantor.
- Una domanda meno ingenua è: “che tipo di problemi intendono risolvere gli algoritmi (ossia quali sono i **problemi algoritmici**) e quali fra questi problemi sono effettivamente in grado di risolvere?”

Problemi indecidibili

- Uno dei risultati più importanti (della logica matematica e) dell'informatica del secolo scorso è stato la scoperta dell'esistenza di problemi (algoritmici) per i quali non esistono algoritmi/programmi in grado di risolverli (**problemi non computabili / indecidibili**).
- E non si tratta di problemi marginali o irrilevanti, ma, in molti casi, di problemi di importanza fondamentale.
- **Esempio**. Non esiste un algoritmo in grado di stabilire se, dati un programma e un suo possibile input, l'esecuzione di tale programma sullo specifico input termina o meno (**problema della terminazione - Turing**).

La classe dei problemi di tassellatura

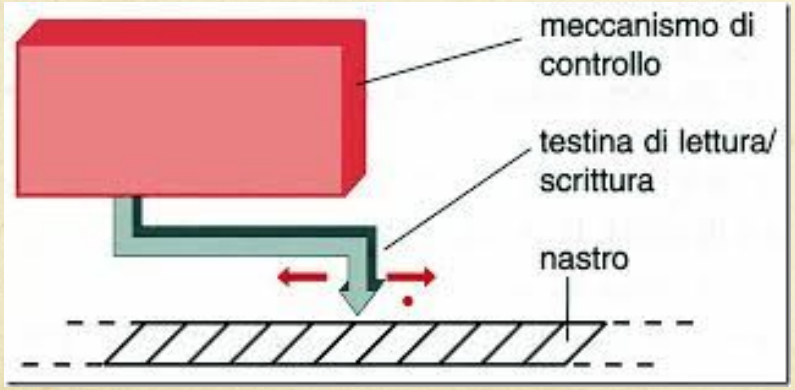
- Fra i problemi indecidibili, ve ne sono alcuni molto semplici.
- E' questo il caso di **molti problemi di tassellatura** (tiling problem).
- Un problema di tassellatura richiede di stabilire, ricevuto in ingresso un insieme finito di tipi diversi di piastrelle colorate (quadrati di dimensioni unitarie divisi in quattro parti dalle diagonali, ognuna delle quali colorata in un certo modo), se sia possibile o meno coprire una determinata superficie del piano con piastrelle dei tipi dati, rispettando alcune semplici condizioni sui colori delle piastrelle adiacenti (ad esempio, stesso colore).

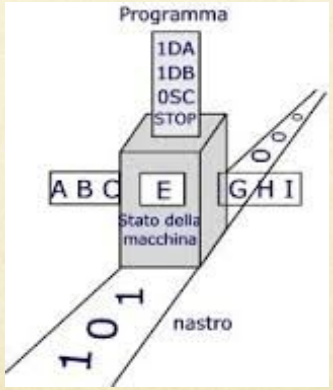
Modelli di calcolo

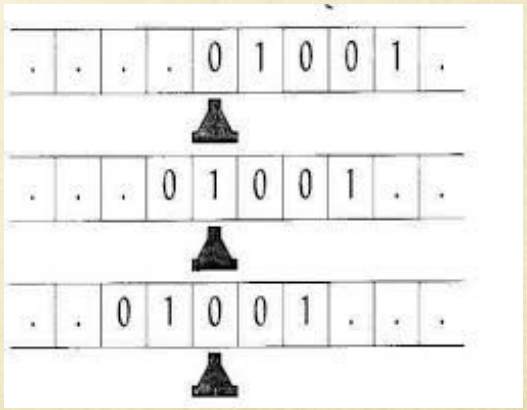
- Un **modello di calcolo**, o di computazione, è una macchina in grado di eseguire algoritmi.

- Il modello di calcolo di riferimento in informatica è un modello molto semplice detto **macchina di Turing** (MdT).

- Come funziona una MdT?







[Immagini disponibili in rete.]

La tesi di Church

Quali problemi algoritmici possono essere risolti con una MdT?

- La **tesi di Church** afferma che le MdT sono in grado di risolvere **tutti i problemi algoritmici** effettivamente risolubili.
- E' una tesi, non un teorema (la nozione di risolubilità effettiva non è formalizzata), ma da tutti accettata.

La macchina di Turing universale

- Turing mostrò anche come creare una singola MdT **(MdT universale)** in grado di fare tutto ciò che può essere fatto da una qualsiasi MdT.
- Una tale MdT riceve in ingresso una MdT **MT** e un input per essa e si comporta esattamente come si comporterebbe **MT** su tale input.
- E' il modello matematico del **calcolatore universale**.
- **MT** e il suo input sono trattati come dati dalla MdT universale che si comporta come un interprete degli attuali linguaggi di programmazione.

Problemi intrattabili

- La situazione è, in verità, ancora "peggiore": vi sono problemi decidibili la cui soluzione risulta troppo onerosa dal punto di vista delle risorse di tempo di calcolo e/o di spazio di memoria necessarie ad un algoritmo per risolverli (**problemi intrattabili**).
- E' questo il caso degli **scacchi**: un programma che volesse esaminare gli effetti di tutte le possibili (sequenze di) mosse in modo da poter scegliere, ad ogni passo, la miglior mossa in assoluto richiederebbe l'analisi di un numero di mosse molto maggiore del numero totale di protoni presenti nell'universo (necessità di **euristiche**).

La complessità computazionale

- La classificazione dei problemi sulla base dell'ammontare di risorse (**tempo** e/o **spazio**) richiesto per la loro soluzione da parte di un qualche strumento di calcolo (ad esempio, una MdT) prende il nome di (teoria della) **complessità computazionale.**
- In generale, tempo e spazio necessari dipendono (sono funzione di) dalla dimensione dell'input: **analisi asintotica** (al crescere della dimensione dell'input).
- A parità di dimensione, la complessità può variare al variare dell'input; di norma, **analisi del caso peggiore** (non è l'unica possibile, né sempre la più ragionevole).

Come misurare le complessità?

- Come **misurare** la complessità temporale (di un algoritmo)?
- Il tempo si misura calcolando il **numero di azioni** elementari effettuate durante l'esecuzione di un algoritmo/programma espresso in funzione della dimensione dell'input.
- A seconda della struttura di tale funzione, si parla di tempo logaritmico, di tempo polinomiale, di tempo esponenziale nella dimensione dell'input.

Trattabilità e intrattabilità

- Distinguiamo tra algoritmi (buoni) che richiedono un tempo polinomiale, o inferiore, e algoritmi (cattivi) che richiedono un tempo esponenziale, o superiore (trascuriamo ciò che sta nel mezzo).
- Un problema per il quale esiste una soluzione algoritmica buona è detto **trattabile**.
- Un problema che ammette solo soluzioni algoritmiche cattive è detto **intrattabile**.

Complessità degli algoritmi e dei problemi

- Da quanto detto in precedenza, emerge una distinzione tra
- **complessità degli algoritmi**
- **complessità dei problemi**

- Limiti superiori e inferiori alla complessità dei problemi: se coincidono, **soluzioni ottimali.**
- **Esempio**. Il problema della calcolo del massimo di un insieme di numeri naturali distinti.

Parole note, nuovi significati

- Linguaggio
- Determinismo/Non Determinismo
- (Tempo) Infinito

Linguaggio naturale e linguaggio formale

- Come costruire insiemi infiniti di parole **(linguaggi)** a partire da un **alfabeto** finito di simboli.
- Ogni alfabeto finito può essere codificato in un **alfabeto binario**. L'**espressività** di un linguaggio non dipende dal suo alfabeto, ma dalle regole per la costruzione delle parole.
- Distinzione fondamentale: linguaggi formali e naturali.

- Un **linguaggio formale** è un insieme, in generale infinito, di parole (finite o infinite), costruite a partire da un alfabeto finito attraverso un opportuno insieme di regole.
- Delle tecniche di elaborazione automatica del **linguaggio naturale** si occupa, invece, un settore importante dell'intelligenza artificiale.

I linguaggi formali

- Tre punti di vista alternativi (ma equivalenti) sui linguaggi formali:

- linguaggi **generati** da una **grammatica** (linguaggio = insieme delle parole generate dalla grammatica)
- linguaggi **accettati** da una **macchina/automa** (linguaggio = insieme delle stringhe accettate dall'automa)
- linguaggi **definiti** da una **formula** (linguaggio = insieme dei modelli della formula)

Linguaggi tra macchine e grammatiche

- Il punto di vista più originale è quello che stabilisce un legame tra linguaggi formali e macchine/automi.
- Una **concezione operazionale** dei linguaggi (formali): un linguaggio (formale) è una macchina che riconosce insiemi di oggetti.
- La caratterizzazione dei linguaggi formali in termini di grammatiche è essenzialmente dovuta a Chomsky**.**
- La **gerarchia di Chomsky** distingue quattro livelli di grammatiche. Ogni **livello di grammatiche** può essere messo in corrispondenza con una **classe di macchine.**

Non determinismo

- Il non determinismo in informatica può essere visto come uno strumento per **astrarre** parti sconosciute o complesse dei sistemi.
- **Macchine** deterministiche e non deterministiche (equivalenza/non equivalenza).
- **Algoritmi** deterministici e non deterministici (le istruzioni di scelta non deterministica).
- **Automi/macchine a stati finiti** deterministici e non deterministici (il caso delle MdT).

MdT non deterministica

- La proprietà distintiva di una MdT non deterministica è che, dati il simbolo contenuto nella cella corrente e lo stato corrente del controllo finito, essa non è obbligata ad eseguire un'unica azione univocamente determinata, ma può scegliere fra diverse possibili azioni.
- A fronte di un dato input, una MdT non deterministica ammette **più computazioni** alternative.
- Condizione di accettazione di una parola da parte di una MdT non deterministica definita in termini dell'**esistenza** di **una computazione** di successo
- Il comportamento di una MdT non deterministica può essere simulato da una MdT deterministica con una perdita esponenziale di efficienza.

Infinito: problema o risorsa?

- L'infinito come problema: dobbiamo garantire che un programma produca l'output atteso in un tempo finito, ossia che sia **terminante.**
- Vi è un'altra classe molto ampia di programmi/sistemi di fondamentale importanza, detti **programmi/sistemi reattivi** (ad esempio, i sistemi operativi), la cui funzione è quella di mantenere nel tempo una certa modalità di interazione con l'ambiente in cui operano.
- Tali programmi/sistemi sono inerentemente **non terminanti** (computazioni infinite).

La logica temporale

- Come specificare e verificare in modo automatico le proprietà attese dei programmi/sistemi reattivi:

- proprietà di sicurezza (safety), vitalità (liveness) e precedenza

- La **logica temporale** quale linguaggio di specifica.

- Sistemi a stati finiti e sistemi a **stati infiniti.**

- Rappresentazione finita di **oggetti infiniti** (computazioni infinite, spazio degli stati infinito). Il ruolo dell'**astrazione.**