Intelligenza Artificiale classica

Scritto da Febogamedev

L'Intelligenza Artificiale classica comprende gli approcci che hanno caratterizzato una parte importante della ricerca sull'IA a partire dagli anni '50. Al suo interno ha avuto un ruolo centrale l'IA simbolica, basata sulla rappresentazione esplicita della conoscenza (Russell & Norvig, 2021).

Nell'approccio simbolico, le informazioni relative a un problema vengono rappresentate attraverso simboli, fatti, regole e relazioni, che il sistema può manipolare per effettuare inferenze, prendere decisioni o cercare una soluzione.

L'idea di fondo è che alcune forme di ragionamento possano essere modellate attraverso la manipolazione di rappresentazioni simboliche secondo regole formali (Russell & Norvig, 2021).

L'IA classica utilizza principalmente tre strumenti concettuali:

- Rappresentazione della conoscenza: I fatti, le proprietà e le relazioni di un dominio vengono formalizzati attraverso strutture come logica proposizionale, logica predicativa, reti semantiche, frame o ontologie.
- Inferenza: Procedure e algoritmi permettono di applicare regole alla conoscenza disponibile per ricavare nuove informazioni o conclusioni.
- Ricerca: Quando un problema presenta molte possibili soluzioni, algoritmi di ricerca, ottimizzazione o pianificazione possono esplorare lo spazio delle possibilità per individuare una soluzione adeguata.

Uno dei principali vantaggi di questi approcci è che le conoscenze utilizzate dal sistema sono spesso rappresentate in maniera esplicita. Questo permette di analizzare e modificare direttamente le regole che determinano il comportamento del sistema e, in molti casi, di ricostruire il processo che ha condotto a una determinata conclusione (Buchanan & Shortliffe, 1984).

L'IA classica presenta tuttavia anche alcuni limiti.

- L'esplosione combinatoria: in molti problemi, aumentando anche di poco le dimensioni del problema, il numero delle possibilità da esaminare può crescere molto rapidamente. Esplorare tutte le alternative può quindi diventare computazionalmente impraticabile (Russell & Norvig, 2021).
- La rappresentazione della conoscenza: nei sistemi simbolici una parte rilevante delle conoscenze necessarie deve essere identificata, formalizzata e inserita esplicitamente nel sistema. Nei domini complessi questa attività può diventare difficile, costosa e difficile da mantenere aggiornata.
- Sistemi basati prevalentemente su regole definite in anticipo possono avere difficoltà ad affrontare situazioni molto variabili, rumorose o non previste durante la progettazione: problemi come il riconoscimento delle immagini o l'elaborazione del linguaggio naturale richiedono infatti di gestire una quantità enorme di possibili variazioni.

Esempio
Un esempio classico di esplosione combinatoria è il problema del commesso viaggiatore (Travelling Salesman Problem).

Un commesso deve visitare un insieme di città e tornare al punto di partenza cercando il percorso complessivamente più breve.

Se fissiamo la città di partenza, il numero degli ordini possibili in cui visitare le altre città cresce approssimativamente secondo: $(n - 1)!$

Per esempio:
- 4 città → 3! = 6 possibili ordini.
- 10 città → 9! = 362.880 possibili ordini.
- 20 città → 19! ≈ 1,22 × 10^{17} possibili ordini.

Il numero di possibilità cresce quindi così rapidamente che, per problemi sufficientemente grandi, provare sistematicamente tutte le alternative diventa impraticabile (Russell & Norvig, 2021).

Queste difficoltà hanno spinto la ricerca verso approcci più adattivi e flessibili, capaci di apprendere direttamente dai dati e di generalizzare da esperienze precedenti: nasce così l'IA moderna, fortemente basata su tecniche di apprendimento e reti neurali.

L'IA classica non è però scomparsa.
Molti sistemi contemporanei combinano infatti approcci simbolici e modelli basati sull'apprendimento (Garcez & Lamb, 2020).
Per esempio, un assistente software può utilizzare un modello neurale per interpretare una richiesta formulata in linguaggio naturale e successivamente applicare regole esplicite per verificare vincoli, autorizzazioni o condizioni previste da un determinato processo.

Problema

Nell'IA classica molti problemi possono essere rappresentati come una ricerca all'interno di un insieme di possibili situazioni.

Il sistema parte da una situazione iniziale e dispone di determinate azioni che permettono di modificarla. Il suo compito consiste nell'individuare una sequenza di azioni che conduca a una situazione che soddisfi l'obiettivo (Russell & Norvig, 2021).

Per esempio, immaginiamo un sistema che debba trovare un percorso all'interno di un labirinto.
- La posizione di partenza rappresenta lo stato iniziale.
- L'uscita rappresenta lo stato obiettivo.
- Ogni spostamento possibile modifica la posizione del sistema e genera quindi un nuovo stato.

Uno stato è una rappresentazione della situazione corrente che contiene le informazioni rilevanti per affrontare il problema.

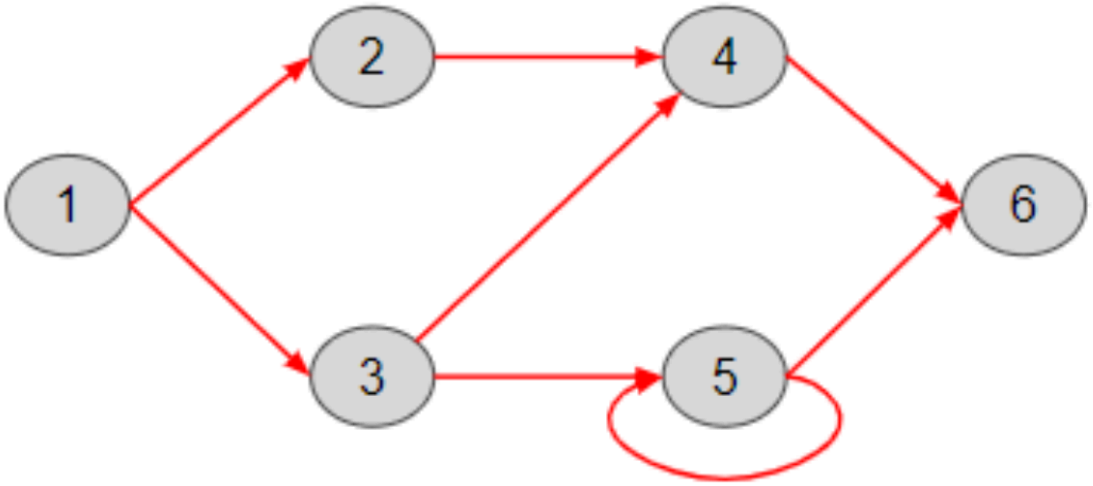
Nel caso del labirinto potrebbe essere sufficiente conoscere la posizione corrente; in problemi più complessi uno stato può contenere molte altre informazioni.

L'insieme degli stati possibili e delle transizioni che permettono di passare da uno stato all'altro prende il nome di **spazio degli stati** (Russell & Norvig, 2021).

Lo spazio degli stati può essere rappresentato attraverso un **grafo orientato**, nel quale i nodi rappresentano gli stati e gli archi rappresentano le azioni che permettono di passare da uno stato a un altro.
Se una trasformazione è reversibile, possono esistere due archi orientati in direzioni opposte.

Esempio
Stato A → azione → Stato B

non implica necessariamente che sia possibile effettuare anche il percorso inverso.



La rappresentazione completa del problema comprende quindi non soltanto gli stati, ma anche la situazione iniziale, le azioni disponibili, l'obiettivo ed eventualmente i costi associati alle diverse azioni (Russell & Norvig, 2021).

Per verificare se uno stato soddisfa l'obiettivo viene utilizzato un **test dell'obiettivo** o **goal test**.
Nel problema del labirinto, per esempio, il test potrebbe semplicemente verificare:
La posizione corrente corrisponde all'uscita?
Se la risposta è positiva, il problema è stato risolto.

Una soluzione è una sequenza di azioni che permette di passare dallo stato iniziale a uno stato obiettivo (Russell & Norvig, 2021).

Questa sequenza prende il nome di **percorso (path)**.
Stato iniziale → azione → stato → azione → stato → … → stato obiettivo

Uno stesso problema può avere più soluzioni.
In un labirinto, per esempio, potrebbero esistere diversi percorsi capaci di raggiungere l'uscita. Alcuni potrebbero essere più brevi, più veloci o richiedere meno risorse di altri.
Per questo può essere associato alle azioni o ai percorsi un costo.

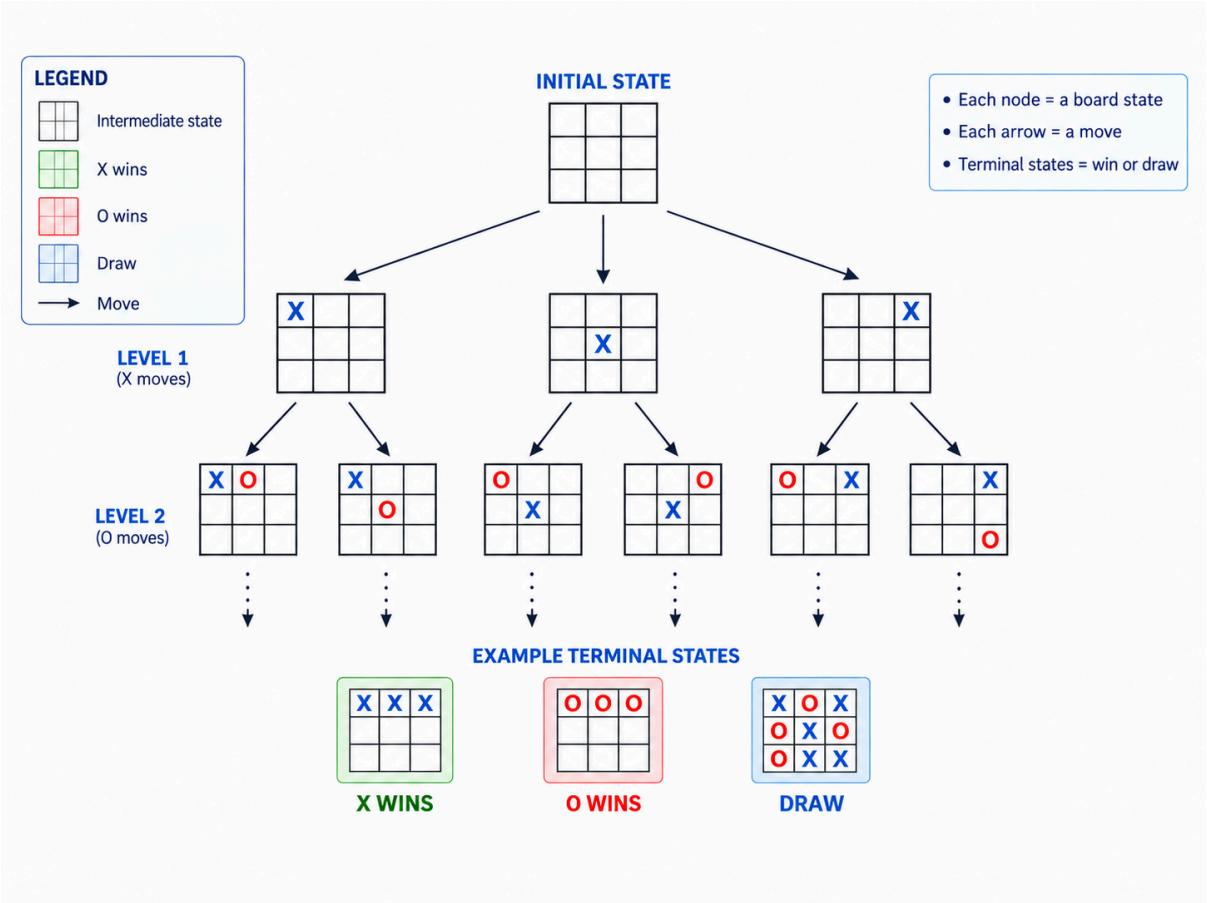
Il costo può rappresentare, per esempio, la distanza percorsa, il tempo necessario, il consumo energetico o il numero di azioni effettuate (Russell & Norvig, 2021).

L'insieme delle possibili soluzioni prende il nome di **spazio delle soluzioni**.
Esso comprende i percorsi che, partendo dallo stato iniziale, conducono a uno stato che soddisfa l'obiettivo.

Quando lo spazio degli stati è molto grande, esplorare tutte le possibilità può essere troppo costoso.
Il sistema deve quindi decidere quali stati esplorare per primi.

Per orientare questa ricerca possono essere utilizzate delle stime che indicano quali stati appaiono più promettenti.
Nel problema di navigazione, per esempio, una possibile stima potrebbe essere la distanza stimata tra la posizione corrente e la destinazione.
La stima non stabilisce necessariamente quale sia la soluzione corretta: serve soprattutto a guidare la ricerca, evitando di esplorare indiscriminatamente tutte le possibilità (Russell & Norvig, 2021).

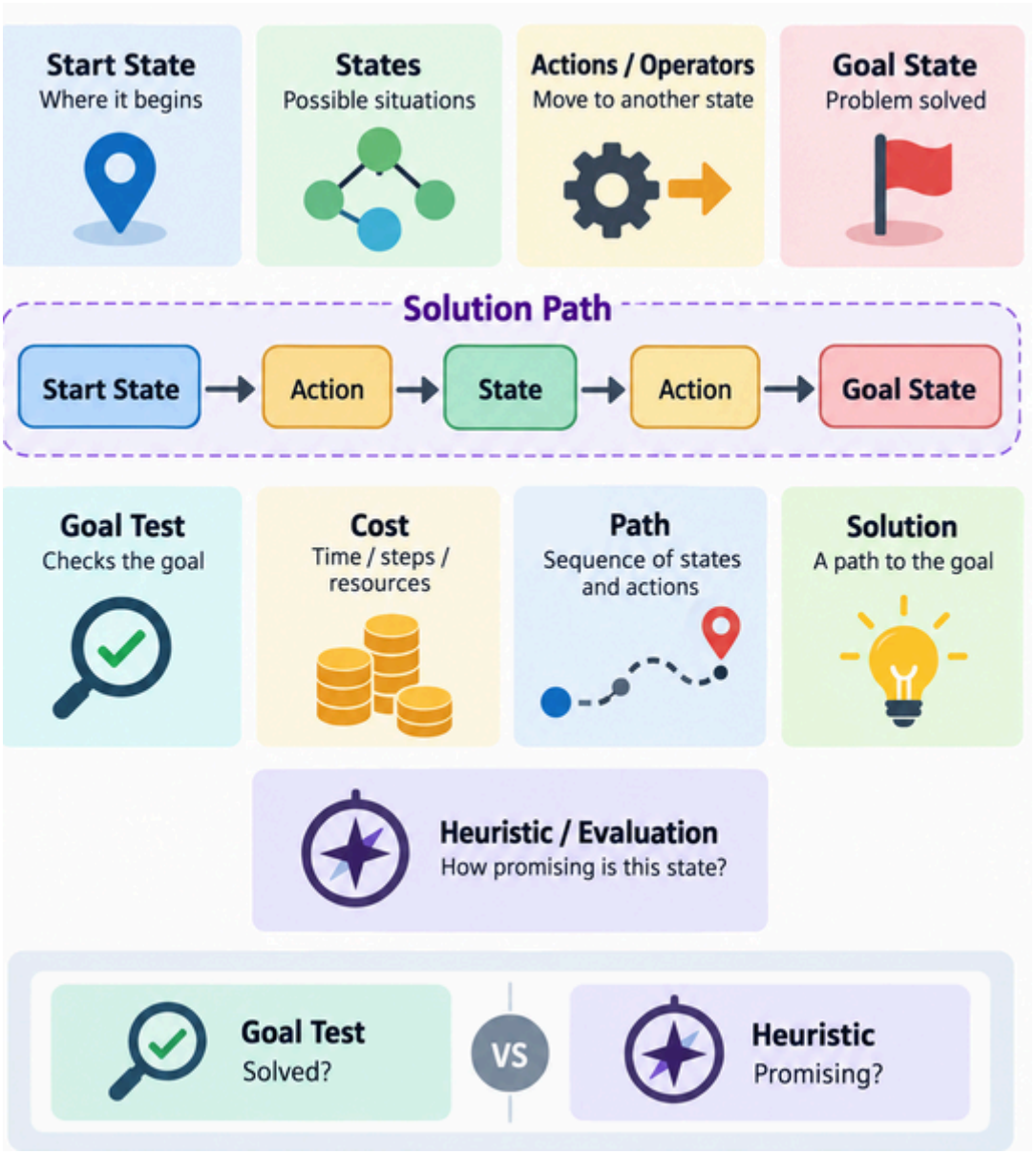
Nei giochi competitivi, come tris o scacchi, la situazione diventa più complessa perché non tutte le transizioni dipendono dallo stesso agente.
Dopo una mossa del sistema interviene infatti l'avversario, che può scegliere una propria azione.
Lo spazio degli stati assume quindi la forma di un albero di gioco, nel quale il sistema deve considerare non soltanto le proprie mosse, ma anche le possibili risposte dell'avversario.
Nel tris, per esempio, ogni configurazione della griglia rappresenta uno stato e ogni mossa genera un nuovo stato.
Una funzione di valutazione può assegnare un valore alle diverse configurazioni, considerando più favorevoli quelle che aumentano la possibilità di vittoria (Russell & Norvig, 2021).
Per esempio, avere due croci allineate con una terza casella libera può essere considerato uno stato favorevole, anche se non rappresenta ancora una vittoria.



Possiamo quindi descrivere un problema rappresentato attraverso uno spazio degli stati mediante alcuni elementi fondamentali:

- Stato iniziale: La situazione dalla quale parte il problema.
- Stati: Le possibili configurazioni rilevanti del problema.
- Azioni o operatori: Le trasformazioni che permettono di passare da uno stato a un altro.
- Stato obiettivo: Uno stato che soddisfa le condizioni richieste per risolvere il problema.
- Test dell'obiettivo: La procedura che verifica se uno stato soddisfa l'obiettivo.
- Costo: Il valore eventualmente associato alle azioni o ai percorsi.
- Percorso: La sequenza di stati e azioni che collega lo stato iniziale a uno stato raggiunto.
- Soluzione: Un percorso che conduce dallo stato iniziale a uno stato obiettivo.

A questi elementi può aggiungersi una **funzione euristica** o, in alcuni problemi come i giochi, **una funzione di valutazione**.
Queste funzioni assegnano un valore agli stati intermedi per stimare quanto siano promettenti e orientare la ricerca verso le alternative più favorevoli.



A partire da questa rappresentazione possiamo distinguere, in modo semplificato, diverse attività di problem solving.

- La ricerca consiste nell'individuare una soluzione valida all'interno dello spazio degli stati. Per esempio, trovare un percorso che permetta di uscire da un labirinto.
- L'ottimizzazione consiste invece nel cercare, tra diverse soluzioni possibili, quella migliore rispetto a un determinato criterio. Per esempio, trovare il percorso più breve tra due punti.
- La pianificazione consiste nel costruire una sequenza di azioni capace di raggiungere un obiettivo tenendo conto di condizioni, vincoli e risorse. Per esempio, organizzare le consegne di diversi camion rispettando orari, capacità dei veicoli e destinazioni (Russell & Norvig, 2021).

Lo spazio degli stati è quindi uno dei principali strumenti utilizzati dall'IA classica per trasformare alcuni problemi in strutture che possono essere esplorate attraverso algoritmi.

Quando lo spazio contiene un numero molto elevato di possibilità, diventa necessario evitare di esplorarlo completamente e individuare strategie capaci di orientare la ricerca verso gli stati più promettenti. Da questa esigenza nasce l'importanza delle euristiche.

Nei prossimi paragrafi approfondiremo due strumenti particolarmente rappresentativi dell'IA classica:

- I sistemi esperti, basati sulla rappresentazione esplicita della conoscenza e sull'inferenza.
- Le euristiche, utilizzate per guidare la ricerca tra molte possibili alternative.

Sistemi esperti

I sistemi esperti sono uno degli esempi più rappresentativi dell'IA simbolica.
Sono stati sviluppati con l'obiettivo di rappresentare in forma computazionale una parte delle conoscenze e delle procedure decisionali utilizzate da uno specialista umano in un determinato settore.

Un sistema esperto è un programma progettato per affrontare problemi all'interno di un dominio specifico, come medicina, chimica, ingegneria o configurazione di sistemi informatici, utilizzando conoscenze rappresentate esplicitamente e procedure di inferenza (Buchanan & Shortliffe, 1984).

Viene definito "esperto" perché utilizza conoscenze specialistiche per formulare valutazioni, suggerimenti o conclusioni relative al dominio per il quale è stato progettato.

La struttura tipica di un sistema esperto comprende:

- Base di conoscenza: contiene fatti, relazioni e regole relative al dominio.
- Motore inferenziale: applica le regole alle informazioni disponibili per ricavare nuove conclusioni.
- Interfaccia utente: permette di fornire al sistema i dati relativi al caso da analizzare e di ricevere le conclusioni prodotte.
- Modulo di spiegazione: quando presente, permette di mostrare quali fatti e quali regole sono stati utilizzati durante l'inferenza. Questo rende generalmente più semplice ricostruire il processo attraverso cui il sistema è arrivato a una determinata conclusione (Buchanan & Shortliffe, 1984).

La base di conoscenza rappresenta esplicitamente le informazioni che il sistema possiede sul proprio dominio.

Una modalità molto comune consiste nell'utilizzare regole nella forma:
SE condizione → ALLORA conclusione

Per esempio:
SE il paziente presenta febbre alta E tosse → ALLORA considera l'ipotesi di un'infezione respiratoria.

Queste regole non costituiscono necessariamente conoscenze universali: rappresentano conoscenze e criteri inseriti nel sistema dagli esperti e dai progettisti.

Il motore inferenziale è il componente che utilizza la base di conoscenza per elaborare le informazioni relative a un caso specifico.
A partire dai fatti disponibili, può verificare quali regole risultano applicabili e utilizzarle per derivare nuove informazioni, formulare ipotesi o suggerire possibili azioni.
Due strategie classiche di inferenza sono il **forward chaining** e il **backward chaining**.

Nel forward chaining, il sistema parte dai fatti disponibili e applica progressivamente le regole compatibili fino a raggiungere nuove conclusioni (Russell & Norvig, 2021).

Esempio
Di seguito è riportato un esempio di forward chaining.
l sistema si trova davanti a un paziente e trova queste due condizioni:

- Il paziente ha la febbre.
- Il paziente ha la tosse.

A partire da queste condizioni, il sistema individua la regola applicabile: SE febbre E tosse → considera l'ipotesi X.

Conclusione: il sistema considera l'ipotesi X.

Nel backward chaining, invece, il sistema parte da un'ipotesi o da un obiettivo e cerca di verificare quali fatti e quali regole siano necessari per sostenerlo (Russell & Norvig, 2021).

Esempio
Sotto un esempio di backward chaining.

Il sistema vuole verificare l'ipotesi: Il paziente presenta la condizione X?
Nella base di conoscenza trova la regola:
SE febbre E tosse → ALLORA condizione X

Il sistema cerca quindi di verificare:

- Il paziente ha febbre?
- Il paziente ha tosse?

Se entrambe le condizioni risultano vere, l'ipotesi X può essere supportata.

Tra gli esempi storici più importanti troviamo DENDRAL, sviluppato per supportare l'identificazione di strutture molecolari attraverso dati di spettrometria di massa; MYCIN, sviluppato negli anni Settanta per fornire raccomandazioni relative alla diagnosi e al trattamento di alcune infezioni batteriche; e XCON, utilizzato da Digital Equipment Corporation per configurare sistemi informatici complessi (Russell & Norvig, 2021).

I sistemi esperti possono ottenere buone prestazioni quando operano in domini circoscritti e sufficientemente formalizzabili.
Uno dei loro principali vantaggi è che le conoscenze e le regole sono rappresentate esplicitamente. Questo permette di modificarle direttamente e, in molti casi, di ricostruire il processo che ha condotto a una determinata conclusione (Buchanan & Shortliffe, 1984).

I sistemi esperti presentano però anche importanti limiti.

- Acquisizione della conoscenza: la loro efficacia dipende dalla qualità e dalla copertura della base di conoscenza. Raccogliere le conoscenze degli specialisti, selezionare quelle rilevanti e trasformarle in fatti e regole formalizzate può essere un processo lungo e complesso. Questa difficoltà è nota come **knowledge acquisition bottleneck** (Buchanan & Shortliffe, 1984).
- Aggiornamento della conoscenza: i sistemi esperti classici non apprendono automaticamente dall'esperienza. Se il dominio cambia o emergono nuove conoscenze, la base di conoscenza deve essere modificata o ampliata manualmente.

A questi problemi si aggiunge la difficoltà di mantenere coerente una base di conoscenza molto ampia: nuove regole possono infatti entrare in conflitto con quelle già presenti o produrre effetti non previsti (Buchanan & Shortliffe, 1984).

I sistemi esperti mostrano quindi molto chiaramente una delle caratteristiche fondamentali dell'IA simbolica: la conoscenza utilizzata dal sistema viene rappresentata esplicitamente e separata dal meccanismo che la utilizza per effettuare inferenze (Buchanan & Shortliffe, 1984).

Euristica

Nell'IA classica uno dei problemi più frequenti consiste nell'esplorare uno spazio degli stati molto grande.

In teoria, un algoritmo potrebbe esaminare sistematicamente tutte le possibili sequenze di azioni fino a trovare una soluzione. Nella pratica, però, il numero degli stati può crescere così rapidamente da rendere una ricerca completa troppo costosa in termini di tempo e risorse.

Per affrontare questo problema vengono utilizzate le **euristiche**.

Un'euristica è una regola o una funzione che fornisce una stima utile per orientare la ricerca, indicando quali stati sembrano più promettenti rispetto al raggiungimento dell'obiettivo (Russell & Norvig, 2021).

L'euristica non determina direttamente quale sia la soluzione corretta. Il suo compito è aiutare l'algoritmo a decidere quali alternative esplorare prima.

Per esempio, immaginiamo di voler trovare un percorso all'interno di una città.

- Una ricerca non informata può esplorare gli stati secondo criteri generali, senza disporre di informazioni specifiche su quali strade conducano più probabilmente verso la destinazione.
- Un'euristica potrebbe invece utilizzare la distanza in linea d'aria dalla destinazione: una posizione apparentemente più vicina all'obiettivo viene considerata più promettente e può essere esplorata prima delle altre.

L'euristica non garantisce necessariamente che quella strada conduca al percorso migliore. Un ostacolo, una strada chiusa o la conformazione della rete stradale potrebbero rendere preferibile un percorso inizialmente più distante.

Una buona euristica dovrebbe quindi essere abbastanza informativa da distinguere gli stati più promettenti, ma anche poco costosa da calcolare. Se ottenere la stima richiedesse quasi lo stesso lavoro necessario per risolvere direttamente il problema, l'euristica perderebbe gran parte della propria utilità (Russell & Norvig, 2021).

È importante distinguere la funzione euristica dalla funzione di valutazione. I due concetti sono collegati, ma non sono equivalenti.

- Una funzione euristica, indicata spesso con *h(n)*, viene tipicamente utilizzata nei problemi di ricerca per stimare il costo ancora necessario per raggiungere l'obiettivo a partire da uno stato. Per esempio, *h(n)* può essere la distanza stimata dalla posizione corrente alla destinazione.
- Una funzione di valutazione, invece, assegna un valore a uno stato per indicare quanto quella situazione sia favorevole o desiderabile. Questo è particolarmente utile nei giochi competitivi, dove non sempre ha senso parlare semplicemente di "distanza dall'obiettivo" (Russell & Norvig, 2021).

La distinzione è importante perché le due funzioni rispondono a domande differenti:

- Funzione euristica: "Quanto potrebbe costare raggiungere l'obiettivo da questo stato?"
- Funzione di valutazione: "Quanto è favorevole questo stato?"

Esempio

Negli scacchi, per esempio, una funzione di valutazione potrebbe considerare elementi come il materiale disponibile, la sicurezza del re, il controllo del centro e la posizione dei pezzi per assegnare un punteggio alla configurazione della scacchiera.

Nel tris, una configurazione con due croci allineate e una terza casella ancora libera potrebbe ricevere una valutazione positiva, perché rappresenta una situazione favorevole per il giocatore che utilizza le croci.

Le euristiche presentano comunque alcuni limiti:

- Dipendono dal problema: una buona euristica per un determinato dominio può essere inutile in un altro.
- Possono essere inaccurate: una stima sbagliata può orientare la ricerca verso stati poco promettenti.

- Richiedono conoscenza del dominio: progettare manualmente una buona euristica significa spesso conoscere abbastanza bene la struttura del problema.
- Non migliorano necessariamente la ricerca: un'euristica poco informativa o costosa da calcolare può offrire pochi vantaggi o persino peggiorare le prestazioni (Russell & Norvig, 2021).

Una proprietà particolare, importante in alcuni algoritmi di ricerca, è **l'ammissibilità**.
Un'euristica è ammissibile quando non sovrastima mai il costo minimo reale necessario per raggiungere l'obiettivo (Russell & Norvig, 2021).

Esempio
Immagina che da una certa posizione manchino realmente almeno 10 km per raggiungere la destinazione.

Se l'euristica stima:
- 8 km, è ammissibile, perché sottostima il costo reale.
- 10 km, è ammissibile, perché coincide con il costo reale minimo.
- 12 km, non è ammissibile, perché sovrastima il costo reale minimo.

Quindi:
- Costo reale minimo restante = 10
- $h(n) \leq 10 \rightarrow$ euristica ammissibile
- $h(n) > 10 \rightarrow$ euristica non ammissibile

L'idea è che un'euristica ammissibile può essere ottimista, ma non deve mai essere troppo pessimista rispetto al costo minimo reale.

Le euristiche sono quindi uno degli strumenti fondamentali attraverso cui l'IA classica riesce a rendere affrontabili spazi di ricerca troppo grandi per essere esplorati completamente.

Il concetto non è però scomparso con l'IA moderna.
Nei sistemi contemporanei, una funzione che svolge un ruolo simile a quello di un'euristica o di una funzione di valutazione può essere appresa dai dati o dall'esperienza, invece di essere definita interamente dal progettista.
Nei sistemi per il gioco, per esempio, una rete neurale può imparare a valutare quanto una determinata posizione sia favorevole e utilizzare questa informazione per guidare la ricerca verso le mosse più promettenti.

Un esempio noto è AlphaZero, che combina reti neurali e ricerca nell'albero di gioco: le reti apprendono a stimare il valore delle posizioni e la probabilità che determinate mosse siano promettenti, mentre la ricerca utilizza queste informazioni per decidere quali possibilità esplorare maggiormente (Silver et al., 2018).

Si può quindi osservare una continuità tra IA classica e moderna:
Nell'IA classica la guida della ricerca è spesso progettata esplicitamente.
- Nell'IA moderna, una funzione con ruolo analogo può anche essere appresa attraverso l'esperienza.

A*

Un esempio eccellente di uso dell'euristica è l'algoritmo A*.
L'algoritmo A* (A star) è uno dei più noti algoritmi di ricerca informata dell'IA classica (Hart et al., 1968).

Il suo obiettivo è trovare un percorso da uno stato iniziale a uno stato obiettivo, cercando di minimizzarne il costo.
Per orientare la ricerca utilizza un'euristica, che permette di assegnare una priorità agli stati che sembrano più promettenti.

A* combina infatti due informazioni: il costo già sostenuto per raggiungere uno stato e una stima del costo ancora necessario per arrivare all'obiettivo.

Per ogni stato *n*, A* calcola:
$$f(n) = g(n) + h(n)$$
Dove:

- *n* → Lo stato del problema da valutare
- *g(n)* → È il costo reale accumulato dallo stato iniziale fino allo stato *n*, per esempio distanza percorsa, tempo impiegato o numero di mosse.
- *h(n)* → È la funzione euristica, cioè una stima del costo minimo ancora necessario per raggiungere l'obiettivo a partire dallo stato *n*.
- *f(n)* → È la stima del costo complessivo di una soluzione che passa attraverso lo stato *n*.

A* mantiene una frontiera, cioè l'insieme degli stati già generati ma non ancora espansi, e seleziona di volta in volta quello con il valore di *f(n)* più basso (Hart et al., 1968).
In questo modo l'algoritmo non considera soltanto quanto uno stato sembri vicino all'obiettivo, ma tiene conto anche di quanto sia costato raggiungerlo.

Possiamo interpretare le due componenti in questo modo:

- g(n): quanto mi è costato arrivare fin qui?
- h(n): quanto penso che mi costerà arrivare da qui all'obiettivo?

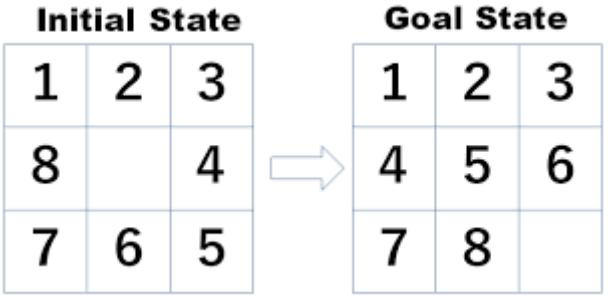
La loro somma permette ad A* di bilanciare il percorso già effettuato con la stima del percorso ancora da compiere.

Questo distingue A* da una ricerca puramente greedy (golosa), che sceglierebbe gli stati considerando soltanto *h(n)*, cioè quelli che sembrano più vicini all'obiettivo, senza considerare il costo già sostenuto (Russell & Norvig, 2021).

Una buona euristica può permettere ad A* di esplorare molti meno stati rispetto a una ricerca non informata. L'efficienza dipende però dalla qualità dell'euristica: una stima poco informativa può ridurre notevolmente questo vantaggio.
A* può garantire una soluzione ottimale con un'euristica ammissibile e opportune condizioni sui costi. Nella ricerca su grafo occorre inoltre gestire correttamente i percorsi migliori verso

stati già visitati, riaprendoli quando necessario; un'euristica consistente consente di evitare queste riaperture (Russell & Norvig, 2021).

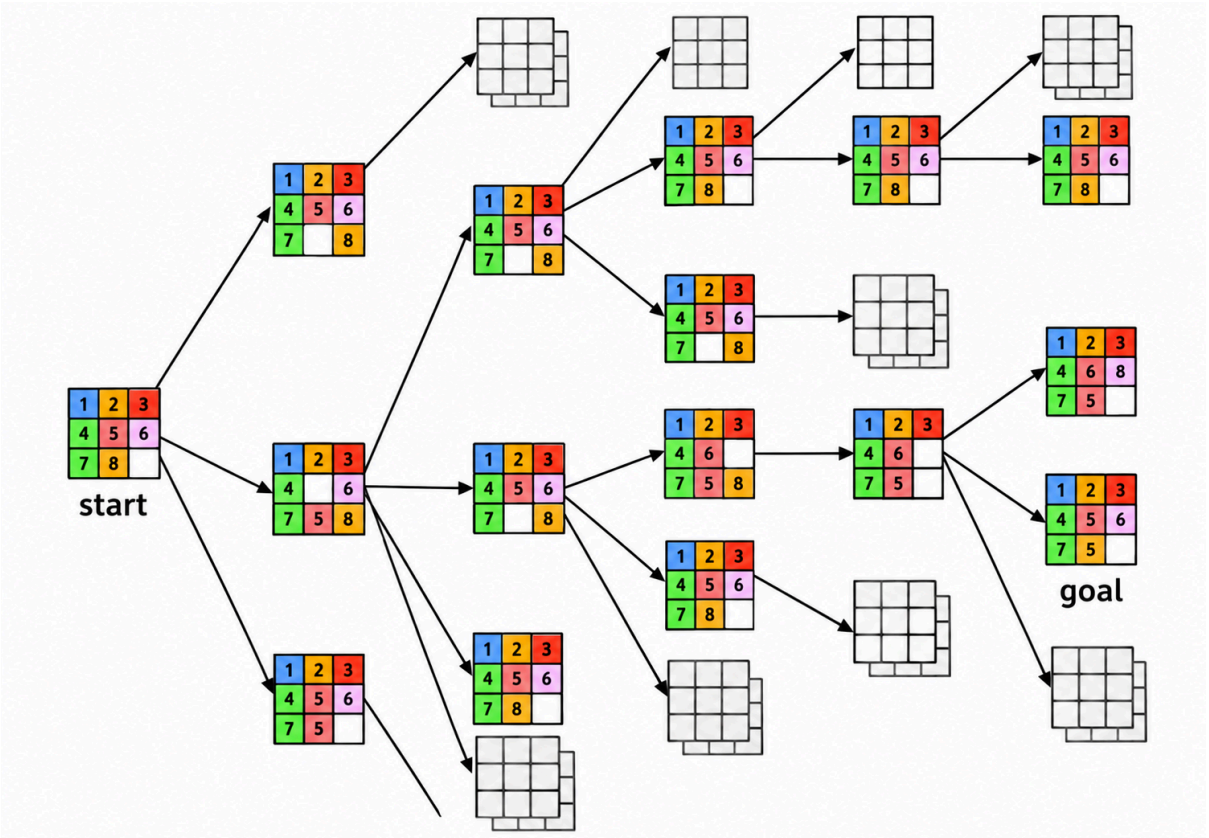
Un classico problema utilizzato per comprendere il funzionamento di A* è l'8-puzzle. L'8-puzzle è un rompicapo formato da una griglia 3×3, contenente otto tessere numerate e una casella vuota.



Una mossa consiste nello spostare una tessera adiacente nella casella vuota.

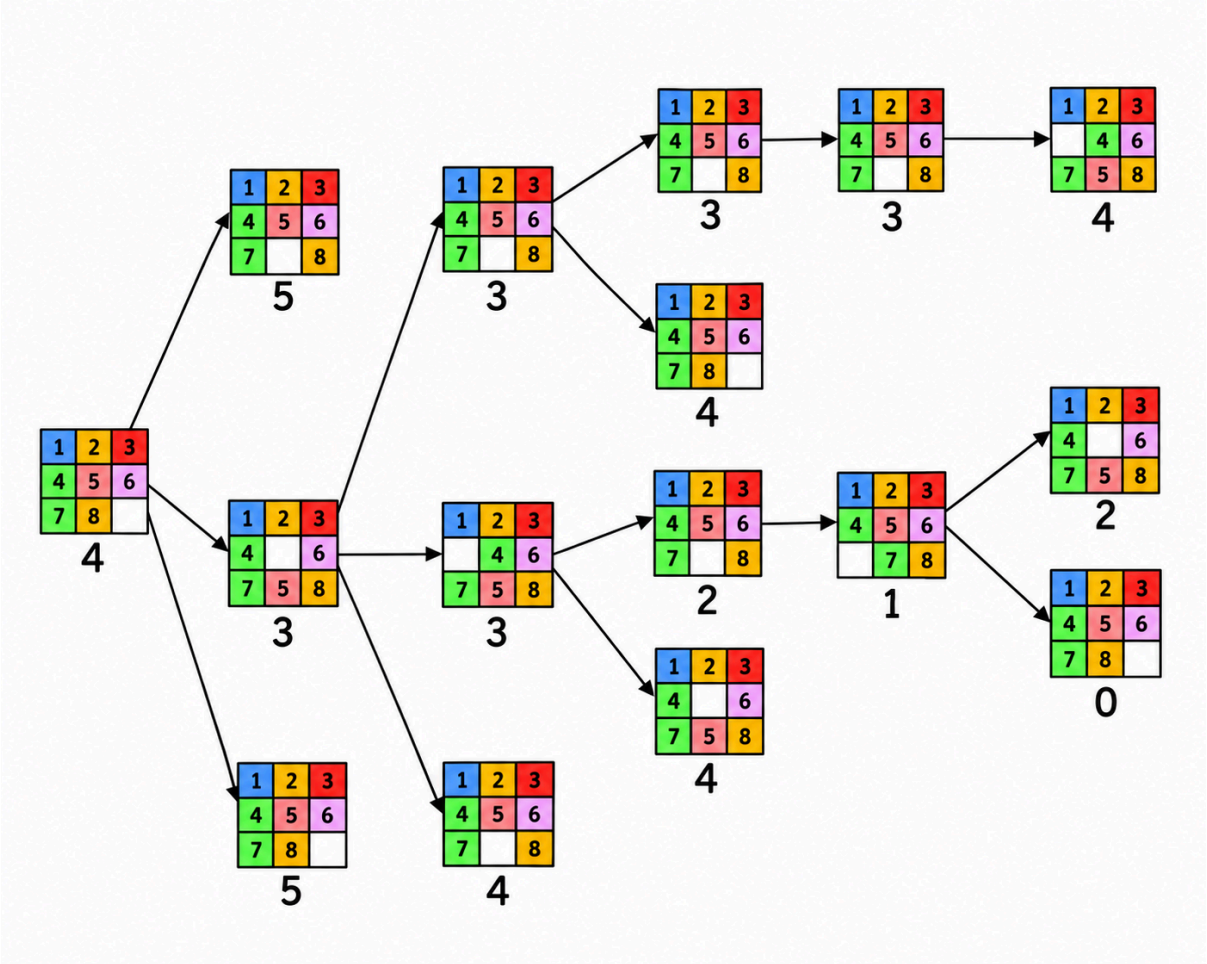
L'obiettivo è trasformare una configurazione iniziale in una configurazione obiettivo, per esempio disponendo le tessere in ordine numerico e lasciando la casella vuota nell'ultima posizione.

Ogni diversa disposizione delle tessere rappresenta uno stato e ogni spostamento possibile rappresenta un'azione che conduce a un nuovo stato.



Nell'immagine è possibile rappresentare una parte dello spazio degli stati attraverso un grafo: partendo dalla configurazione iniziale vengono generate le configurazioni raggiungibili con una mossa, poi quelle raggiungibili con due mosse e così via.

Per applicare A* è necessario definire anche una funzione euristica.
Una possibile euristica per l'8-puzzle è il numero di tessere fuori posto rispetto alla configurazione obiettivo (Russell & Norvig, 2021).
Per esempio, se quattro tessere non si trovano nella posizione corretta: $h(n) = 4$
Lo stato obiettivo avrà invece $h(n) = 0$, questo perché tutte le tessere si trovano nella posizione desiderata.



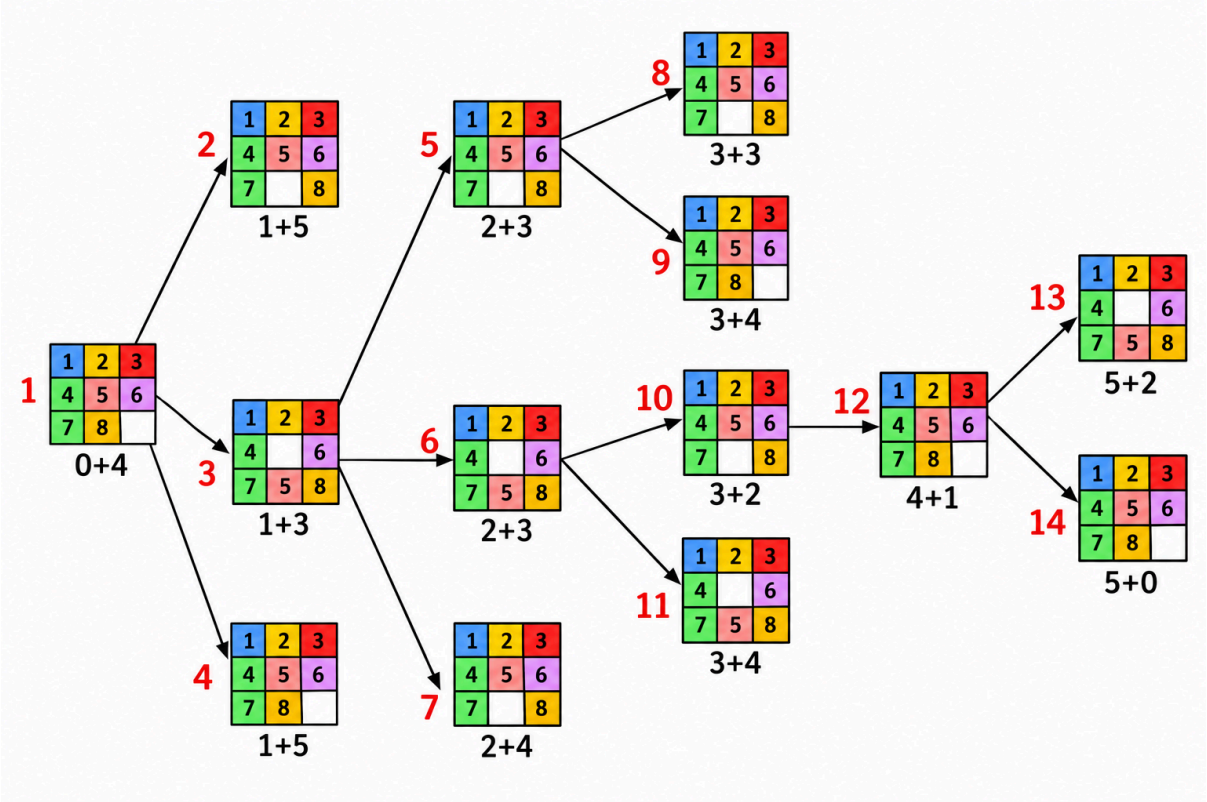
Se ogni mossa ha costo 1, il valore *g(n)* corrisponde semplicemente al numero di mosse effettuate dallo stato iniziale.

Immaginiamo quindi uno stato raggiunto dopo due mosse, con tre tessere ancora fuori posto:

- $g(n) = 2.$
- $h(n) = 3.$

A* calcola: $f(n) = g(n) + h(n) = 2 + 3 = 5.$

L’algoritmo confronta questo valore con quello degli altri stati presenti nella frontiera e seleziona per l’espansione quello con il valore di *f(n)* più basso.



Nel grafico dell’esempio, A* parte dallo stato iniziale e genera i suoi stati successivi.
Tra quelli presenti nella frontiera seleziona quello con il valore minimo di *f(n)* e lo espande, generando nuove possibilità. Il procedimento viene ripetuto aggiornando continuamente la frontiera.
In particolare, per l’esempio dell’immagine, A* funzionerebbe in questo modo:

- A* espande lo stato 1, con $f(n) = 4$, e inserisce nella frontiera gli stati raggiungibili.
- Tra gli stati presenti nella frontiera, lo stato 3 ha il valore minimo, $f(n) = 4$, e viene quindi espanso.
- L’espansione dello stato 3 genera, tra gli altri, gli stati 5 e 6, entrambi con 5.
- Poiché i due stati hanno lo stesso valore, una regola di spareggio può determinare quale venga espanso per primo. Supponiamo che venga scelto lo stato 5.
- Dopo l’espansione dello stato 5, lo stato 6 rimane lo stato con il valore minimo della frontiera e viene quindi espanso.
- Lo stato 6 genera lo stato 10, con $f(n) = 5$, che viene successivamente espanso.
- Lo stato 10 conduce allo stato 12, anch’esso con $f(n) = 5$.
- Espandendo lo stato 12 viene generato lo stato 14, con $f(n) = 5$ e $h(n) = 0$: questo stato soddisfa il test dell’obiettivo.
- Quando lo stato obiettivo viene selezionato dalla frontiera come nodo da espandere, A* può terminare la ricerca e ricostruire il percorso che lo collega allo stato iniziale (Russell & Norvig, 2021).

Se due stati possiedono lo stesso valore di *f(n)*, entrambi risultano ugualmente promettenti secondo questo criterio e l'implementazione può utilizzare una regola aggiuntiva per decidere quale esplorare per primo.
Il processo continua fino a quando viene selezionato uno stato che soddisfa il test dell'obiettivo. A quel punto l'algoritmo può ricostruire la sequenza di azioni che collega lo stato iniziale allo stato obiettivo (Russell & Norvig, 2021).

È importante osservare che durante questa procedura A* non sta facendo realmente "muovere" un agente attraverso gli stati: sta esplorando una rappresentazione del problema e costruendo un possibile percorso. Il percorso risultante potrà eventualmente essere eseguito successivamente da un agente.

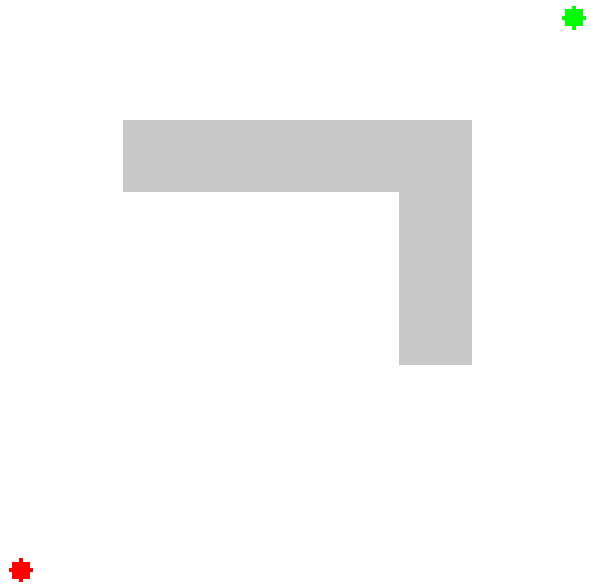
Perché A* possa garantire una soluzione ottimale, è particolarmente importante la proprietà di ammissibilità dell'euristica.
Come già detto, un'euristica è ammissibile quando non sovrastima mai il costo minimo reale necessario per raggiungere l'obiettivo.
Supponiamo, per esempio, che da una determinata configurazione dell'8-puzzle siano realmente necessarie almeno 6 mosse per raggiungere l'obiettivo.

- Se $h(n) = 5$, l'euristica è ammissibile perché sottostima il costo reale.
- Se $h(n) = 6$, è ancora ammissibile perché fornisce una stima esatta.
- Se invece $h(n) = 7$, non è ammissibile perché sovrastima il costo minimo reale.

La sovrastima è importante perché può far apparire un percorso realmente conveniente più costoso di quanto sia, alterando la priorità con cui A* esplora le alternative e facendo venir meno le condizioni che permettono di garantire l'ottimalità.

Una delle applicazioni più comuni di A* è il **pathfinding**, cioè la ricerca di un percorso tra due punti all'interno di un ambiente.
Nel pathfinding ogni posizione può essere rappresentata come uno stato e ogni movimento consentito come un'azione.

A seconda del tipo di movimento consentito, una possibile euristica può essere la distanza euclidea o la distanza Manhattan (Russell & Norvig, 2021).
Questa stima tende a orientare la ricerca verso l'obiettivo, mentre il valore *g(n)* tiene conto del percorso già effettuato.
Quando l'algoritmo incontra un ostacolo, non procede semplicemente in linea retta verso la destinazione: esplora percorsi alternativi, continuando però ad assegnare maggiore priorità agli stati che presentano un buon equilibrio tra costo già sostenuto e costo stimato rimanente.



A* mostra quindi molto chiaramente il ruolo delle euristiche nell'IA classica: non sostituiscono la ricerca, ma la guidano, permettendo all'algoritmo di concentrare l'esplorazione sulle alternative che sembrano più promettenti.

IA bio-ispirata

Quando lo spazio delle possibili soluzioni diventa molto grande, esplorarlo completamente può essere impraticabile.
Anche utilizzando tecniche di ricerca informata come A*, alcuni problemi presentano un numero di possibilità così elevato da rendere troppo costoso analizzarle sistematicamente.

In questi casi possono essere utilizzati algoritmi di ottimizzazione, il cui obiettivo è cercare soluzioni di buona qualità rispetto a uno o più criteri.

In alcuni problemi è possibile individuare matematicamente la soluzione migliore in assoluto. In altri, invece, trovare l'ottimo globale richiederebbe un tempo o una quantità di risorse troppo elevati.
In questi casi può essere più utile trovare una soluzione molto buona senza dover esplorare tutte le possibilità.

Una famiglia particolare di tecniche di ottimizzazione è costituita dagli **algoritmi bio-ispirati**, che prendono spunto da alcuni meccanismi osservabili nei sistemi naturali.

L’IA bio-ispirata comprende infatti tecniche che trasformano principi biologici, evolutivi o comportamentali in procedure computazionali utili per esplorare grandi spazi di soluzioni (de Castro, 2007).

A differenza dell’IA simbolica, questi algoritmi non si basano necessariamente sulla rappresentazione esplicita di conoscenze e regole logiche.
Per questo motivo è più corretto considerarli come un approccio distinto al problem solving e all’ottimizzazione, sviluppato parallelamente ad altre famiglie dell’Intelligenza Artificiale.

I processi naturali hanno prodotto nel tempo meccanismi di adattamento, selezione, cooperazione e auto-organizzazione.
Gli algoritmi bio-ispirati non copiano direttamente la natura, ma utilizzano alcuni di questi principi come modello per progettare strategie di ricerca e ottimizzazione.

Tra le principali famiglie possiamo trovare:

- Algoritmi evolutivi: Prendono ispirazione dall’evoluzione biologica e dalla selezione naturale.
- Swarm intelligence: Si ispira al comportamento collettivo di gruppi come colonie di formiche, sciami o stormi.
- Sistemi immunitari artificiali: Riprendono alcuni principi dei sistemi immunitari biologici, come riconoscimento, adattamento e memoria (de Castro, 2007).

Il funzionamento cambia a seconda della tecnica, ma l’idea generale consiste spesso nel generare o mantenere diverse soluzioni candidate, valutarne la qualità e modificare progressivamente l’insieme delle possibilità esplorate.

Per poter utilizzare questi algoritmi è comunque necessario definire alcuni elementi fondamentali.
È necessario stabilire come rappresentare una soluzione, come valutarne la qualità attraverso una funzione obiettivo o funzione di fitness, quali eventuali vincoli debbano essere rispettati e quali parametri governino il comportamento dell’algoritmo (Eiben & Smith, 2015).

Gli algoritmi bio-ispirati sono particolarmente interessanti quando lo spazio delle soluzioni è molto grande, irregolare o difficile da esplorare attraverso metodi esaustivi. Possono essere utilizzati anche quando non è disponibile una formulazione analitica semplice del problema o quando i metodi di ottimizzazione esatti risultano troppo costosi.

Non garantiscono però necessariamente di trovare la soluzione migliore in assoluto.
Spesso cercano invece di raggiungere una soluzione sufficientemente buona all’interno delle risorse computazionali disponibili. Il risultato può dipendere dai parametri scelti, dalla rappresentazione delle soluzioni e dalla casualità utilizzata durante l’esplorazione (Eiben & Smith, 2015).
Anche il costo computazionale può essere elevato, soprattutto quando è necessario valutare un grande numero di soluzioni candidate o quando la funzione utilizzata per misurarne la qualità è complessa.

L’IA bio-ispirata rappresenta quindi un modo diverso di affrontare il problema della ricerca: invece di esplorare sistematicamente tutti gli stati possibili, utilizza meccanismi ispirati ai sistemi naturali per esplorare, selezionare e migliorare progressivamente le soluzioni.

Tra questi approcci, uno dei più conosciuti è quello degli algoritmi evolutivi, e in particolare degli algoritmi genetici, che prendono ispirazione dai processi di selezione, riproduzione e mutazione presenti nell’evoluzione biologica.

Algoritmi evolutivi

Gli algoritmi evolutivi (Evolutionary Algorithms, EA) sono una famiglia di algoritmi di ottimizzazione bio-ispirati che prendono spunto da alcuni meccanismi dell’evoluzione biologica, come variazione, ereditarietà e selezione (Eiben & Smith, 2015).

Nell’evoluzione biologica gli individui di una popolazione presentano caratteristiche differenti. Alcune di queste differenze sono ereditabili e possono influenzare la capacità degli individui di sopravvivere e, soprattutto, di riprodursi. Se in un determinato ambiente alcune caratteristiche favoriscono un maggiore successo riproduttivo, nel corso delle generazioni tali caratteristiche possono diventare più frequenti nella popolazione.

La selezione naturale non segue un progetto e non cerca intenzionalmente una soluzione ottimale: è il risultato delle differenze nel successo riproduttivo degli individui all’interno di uno specifico ambiente (Eiben & Smith, 2015).

Esempio
Un esempio classico è quello della falena delle betulle (Biston betularia).

Prima dell’industrializzazione, in alcune aree dell’Inghilterra le falene di colore chiaro risultavano meglio mimetizzate sui tronchi chiari ricoperti di licheni, mentre quelle più scure erano generalmente più visibili ai predatori.

Con l’industrializzazione, l’inquinamento modificò l’ambiente e rese in alcune zone i tronchi più scuri. In queste condizioni le falene melaniche risultavano meno visibili e potevano ottenere un vantaggio selettivo.

Quando successivamente l’inquinamento diminuì e l’ambiente cambiò nuovamente, anche questo vantaggio si ridusse (Cook et al., 2012).

L’esempio mostra un principio importante: una caratteristica non è vantaggiosa in assoluto, ma in relazione all’ambiente nel quale l’organismo vive.

Gli algoritmi evolutivi trasformano alcuni di questi principi in procedure computazionali.

Invece di avere organismi biologici, abbiamo una popolazione di soluzioni candidate.
Ogni soluzione viene valutata in base a quanto bene risolve il problema.
Le soluzioni migliori hanno generalmente maggiori probabilità di contribuire alla generazione di nuove soluzioni, che vengono successivamente modificate e nuovamente valutate.

Ripetendo questo processo per molte generazioni, la popolazione può progressivamente spostarsi verso regioni dello spazio delle soluzioni caratterizzate da risultati migliori.

Gli algoritmi evolutivi non devono quindi conoscere in anticipo il percorso che conduce alla soluzione.
Esplorano lo spazio delle possibilità attraverso variazione e selezione, utilizzando una funzione di valutazione per stabilire quali soluzioni siano più promettenti (Eiben & Smith, 2015).

Uno degli algoritmi evolutivi più conosciuti è **l'algoritmo genetico (Genetic Algorithm, GA)**.

Negli algoritmi genetici ogni possibile soluzione viene rappresentata come un individuo.
Una popolazione contiene molti individui, che vengono valutati, selezionati e modificati nel corso delle generazioni.

Questi algoritmi sono particolarmente utili quando lo spazio delle possibili soluzioni è molto grande o complesso, quando una ricerca esaustiva sarebbe troppo costosa oppure quando non è disponibile un metodo semplice per individuare direttamente la soluzione ottimale.

Non è necessario che manchi un algoritmo deterministico: un algoritmo genetico può essere utilizzato anche quando esistono altri metodi, se risulta conveniente esplorare lo spazio delle soluzioni in maniera approssimata e stocastica.

Il loro vantaggio è che possono esplorare grandi spazi di possibilità senza dover analizzare tutte le soluzioni.
Non richiedono necessariamente una funzione matematica semplice o derivabile, ma devono comunque disporre di una rappresentazione delle soluzioni, di un criterio per valutarle e di eventuali meccanismi per gestire i vincoli.

Non garantiscono però di individuare la soluzione migliore in assoluto.
Il risultato dipende dalla qualità dell'algoritmo, dei suoi parametri e dalla componente casuale utilizzata durante l'esplorazione (Eiben & Smith, 2015).

Per comprendere il funzionamento di un algoritmo genetico possiamo utilizzare il problema dello zaino.

Immaginiamo di avere un insieme di oggetti.
Ogni oggetto i possiede un peso w_i e un valore v_i.
Lo zaino ha una capacità massima C.

L'obiettivo consiste nel selezionare gli oggetti in modo da massimizzare il valore complessivo, senza superare la capacità dello zaino.

Una possibile soluzione può essere rappresentata attraverso una stringa binaria nella quale ogni posizione corrisponde a un oggetto:

- 1 = oggetto selezionato.

- 0 = oggetto non selezionato.

Supponiamo di avere quattro oggetti:
moneta, corona, pipa, felpa.

In questo caso l'individuo (0, 1, 0, 1) rappresenta una soluzione candidata nella quale vengono inserite nello zaino soltanto la corona e la felpa.
È importante notare che questa stringa rappresenta una soluzione candidata, non uno stato attraversato durante un percorso come avveniva negli algoritmi di ricerca come A*.

Il funzionamento di un algoritmo genetico può essere descritto attraverso alcune fasi principali:

- Rappresentazione delle soluzioni: si stabilisce come codificare ogni soluzione candidata. Nel problema dello zaino può essere utilizzata una stringa binaria nella quale ogni bit indica se un oggetto è stato selezionato.
- Impostazione dei parametri: si definiscono parametri come dimensione della popolazione, probabilità di crossover (lo vedremo a breve), probabilità di mutazione e numero massimo di generazioni. Non esiste una dimensione della popolazione valida universalmente: dipende dal problema e dal costo della valutazione.
- Inizializzazione della popolazione: si genera una popolazione iniziale di individui, spesso in modo casuale. A seconda dell'algoritmo, possono essere generate soltanto soluzioni valide oppure possono essere accettate anche soluzioni che violano alcuni vincoli e successivamente penalizzate.
- Valutazione della fitness: a ogni individuo viene assegnato un punteggio che indica la qualità della soluzione. Questo valore viene calcolato attraverso una **funzione di fitness**, cioè una particolare funzione di valutazione utilizzata dagli algoritmi evolutivi.
- Selezione dei genitori: alcuni individui vengono scelti per generare nuove soluzioni. Gli individui con fitness migliore vengono generalmente favoriti, senza però eliminare necessariamente ogni possibilità di selezionare individui meno performanti.
- Crossover: due individui selezionati vengono combinati per generare nuovi individui. Nel one-point crossover, per esempio, le stringhe vengono divise in un punto e le loro parti vengono ricombinate.
- Mutazione: alcune caratteristiche dei nuovi individui vengono modificate casualmente. Nelle stringhe binarie una mutazione può trasformare un bit da 0 a 1 o da 1 a 0. La mutazione introduce nuova variabilità e permette di esplorare regioni dello spazio delle soluzioni che potrebbero non essere raggiunte attraverso il solo crossover.
- Sostituzione: I nuovi individui vengono inseriti nella popolazione della generazione successiva, sostituendo parte o tutta la popolazione precedente.
- Controllo di arresto: Il processo viene ripetuto finché viene soddisfatto un criterio di arresto, come il raggiungimento di una qualità ritenuta sufficiente rispetto alla funzione obiettivo, di un numero massimo di generazioni o di un limite di tempo (Eiben & Smith, 2015).

Nel problema dello zaino, una possibile funzione obiettivo consiste nel massimizzare il valore totale degli oggetti selezionati.
La funzione di fitness può utilizzare questo valore, ma deve anche considerare il vincolo sul peso.

Per esempio, se una soluzione non supera la capacità C, la fitness può dipendere direttamente dal valore totale degli oggetti selezionati.
Se invece il peso supera la capacità, alla soluzione può essere applicata una penalità che ne riduca la fitness.

La fitness quindi non deve necessariamente coincidere con la funzione obiettivo: può derivare da essa e incorporare vincoli o penalità necessarie a guidare correttamente l'evoluzione della popolazione (Eiben & Smith, 2015).

Supponiamo ora che vengano selezionati due individui:
- Genitore A: (0, 1, 0, 1)
- Genitore B: (0, 0, 1, 0)

Con un one-point crossover, dividendo le stringhe dopo il secondo bit, possiamo ottenere il figlio equivalente a (0, 1, 1, 0).
Successivamente può intervenire una mutazione.
Se viene modificato il quarto bit:
- Prima della mutazione: (0, 1, 1, 0)
- Dopo la mutazione: (0, 1, 1, 1)

La nuova soluzione viene nuovamente valutata attraverso la funzione di fitness e può partecipare alla generazione successiva.

Ripetendo ciclicamente valutazione, selezione, crossover e mutazione, la popolazione tende a esplorare lo spazio delle soluzioni privilegiando progressivamente le regioni che producono risultati migliori (cfr. Eiben & Smith, 2015).

L'algoritmo genetico mostra quindi una strategia molto diversa da quella utilizzata da un algoritmo come A*: A* esplora uno spazio di stati cercando un percorso verso un obiettivo; un algoritmo genetico fa evolvere contemporaneamente una popolazione di soluzioni candidate cercando di migliorarne progressivamente la qualità.

Materiale di supporto

Esercizi

Rappresentare un problema come spazio degli stati
Un robot lavora in un magazzino formato da diverse zone collegate tra loro.
Deve prelevare un pacco dalla zona D e portarlo nella zona H.

Le zone sono collegate così:
A → B → D → E
↓ ↓
C → F → G → H

Il robot parte dalla zona A. Ogni spostamento tra due zone collegate costa 1 unità di energia.

- Individua lo stato iniziale.
- Individua lo stato obiettivo.
- Definisci che cosa rappresenta uno stato in questo problema.
- Individua le possibili azioni o operatori.
- Descrivi lo spazio degli stati.
- Trasforma il problema in un grafo orientato indicando nodi e archi.
- Formula il goal test.
- Individua almeno due possibili percorsi che permettono di raggiungere H.
- Spiega la differenza tra percorso e soluzione.
- Calcola il costo dei percorsi individuati.
- Se esistono più soluzioni, spiega quale sceglieresti e secondo quale criterio.
- Immagina ora che il collegamento F → G richieda 5 unità di energia anziché 1. Spiega perché il percorso con meno spostamenti potrebbe non essere più quello migliore.
- Classifica il problema come problema di ricerca, ottimizzazione o pianificazione, motivando la risposta.
- Spiega infine perché lo stato non deve rappresentare tutta la realtà del magazzino, ma soltanto le informazioni rilevanti per risolvere il problema.

Progettare un piccolo sistema esperto
Una scuola vuole costruire un sistema che suggerisca quale laboratorio assegnare a una lezione.
Sono disponibili tre laboratori:

- Lab A: 20 computer e nessuna attrezzatura audiovisiva.
- Lab B: 30 computer e videoproiettore.
- Lab C: 15 computer, telecamere e attrezzatura audio.

Il sistema riceve informazioni come:

- Numero di studenti.
- Necessità di utilizzare computer.
- Necessità di registrare video.

- Necessità di utilizzare attrezzatura audio.
- Disponibilità dei laboratori.

Progetta un semplice sistema esperto.
- Individua quali informazioni devono entrare nella base di conoscenza.
- Scrivi almeno cinque regole nella forma: SE condizione → ALLORA conclusione.
- Inserisci almeno una regola che utilizzi due condizioni contemporaneamente.
- Inserisci almeno una situazione nella quale nessun laboratorio sia adatto.
- Spiega il ruolo del motore inferenziale.
- Descrivi il ruolo dell'interfaccia utente.
- Spiega che cosa dovrebbe mostrare un eventuale modulo di spiegazione.
- Considera questo caso: "24 studenti, necessità di computer, nessuna necessità di registrazione audiovisiva, tutti i laboratori disponibili". Applica le regole attraverso forward chaining e ricostruisci il percorso che porta alla conclusione.
- Parti invece dall'ipotesi: «Il Lab C è adatto». Utilizza il backward chaining per stabilire quali condizioni devono essere verificate.
- Aggiungi successivamente una nuova regola: «Le attività audiovisive con più di 12 studenti richiedono anche il Lab B come spazio di supporto». Individua eventuali problemi o conflitti introdotti nella base di conoscenza.
- Spiega perché, se la scuola acquistasse nuove attrezzature, il sistema non aggiornerebbe automaticamente le proprie conoscenze.
- Collega questo problema al knowledge acquisition bottleneck.
- Concludi spiegando perché in un sistema esperto conoscenza e meccanismo inferenziale sono componenti distinte.

Guidare la ricerca con un'euristica e A*
Un personaggio di un videogioco deve raggiungere una torre.
Il sistema ha già generato quattro possibili stati intermedi:

Stato	Costo già sostenuto *g(n)*	Stima del costo restante *h(n)*
A	2	7
B	5	3
C	3	4
D	6	1

Utilizza il funzionamento dell'algoritmo A*.
- Calcola per ogni stato: f(n) = g(n) + h(n).
- Ordina gli stati dal più promettente al meno promettente secondo A*.
- Indica quale stato verrebbe espanso per primo.
- Spiega il significato di:
  - *g(n)*.
  - *h(n)*.
  - *f(n)*.

- Spiega che cosa si intende per frontiera.
- Una ricerca greedy sceglierebbe necessariamente lo stesso stato di A*? Calcola quale sceglierebbe considerando soltanto h(n).
- Spiega perché prendere in considerazione soltanto la distanza stimata dall'obiettivo può essere rischioso.
- Immagina che per lo Stato D il costo minimo reale restante sia 4, mentre l'euristica stima $H(D) = 1$. L'euristica è ammissibile? Motiva.
- Immagina invece che stimi $H(D) = 6$. È ancora ammissibile?
- Spiega perché un'euristica che sovrastima può compromettere la garanzia di trovare una soluzione ottimale.
- Proponi una possibile euristica per un videogioco 2D nel quale il personaggio può muoversi soltanto in quattro direzioni.
- Proponi una seconda euristica più informativa ma anche più costosa da calcolare.
- Spiega perché una buona euristica deve trovare un equilibrio tra qualità della stima e costo computazionale.
- Concludi spiegando questa frase: «L'euristica non sostituisce la ricerca: la orienta.»

Progettare un algoritmo genetico

Un festival deve costruire il programma di una giornata scegliendo tra 10 possibili attività. Ogni attività possiede:

- Una durata.
- Un costo.
- Un valore di interesse stimato dal pubblico.

Il festival dispone di:

- Massimo 8 ore.
- Budget massimo di 5.000 €.

L'obiettivo è creare un programma che massimizzi l'interesse complessivo rispettando tempo e budget.

Progetta concettualmente un algoritmo genetico per affrontare il problema.

- Proponi una rappresentazione binaria di una soluzione candidata.
- Spiega che cosa rappresenterebbe, per esempio: 1 0 1 1 0 0 1 0 1 0.
- Distingui individuo e popolazione.
- Definisci una possibile funzione obiettivo.
- Progetta una funzione di fitness che tenga conto anche dei vincoli di tempo e budget.
- Spiega come gestiresti una soluzione che supera leggermente il budget.
- Descrivi come potrebbe essere creata la popolazione iniziale.
- Spiega secondo quale criterio selezioneresti i genitori.
- Prendi questi due individui ed esegui un one-point crossover dopo il quinto bit e scrivi due possibili figli.
    - A = 1 0 1 1 0 0 1 0 0 1.
    - B = 0 1 0 0 1 1 0 1 1 0.
- Applica successivamente una mutazione a uno dei figli modificando un singolo bit.
- Spiega perché la mutazione può essere utile anche quando apparentemente peggiora una soluzione.
- Indica almeno tre possibili criteri di arresto.

- Spiega perché il risultato migliore ottenuto non è necessariamente la soluzione ottima in assoluto.
- Confronta questa strategia con A*:
    - Che cosa rappresentano gli elementi esplorati nei due casi?
    - A* cerca un percorso oppure fa evolvere soluzioni?
    - L'algoritmo genetico mantiene una singola soluzione o una popolazione?
- Spiega perché un algoritmo genetico può essere utile quando lo spazio delle soluzioni è troppo grande per essere esplorato esaustivamente.
- Concludi ricostruendo il ciclo: Rappresentazione → popolazione → fitness → selezione → crossover → mutazione → nuova popolazione → arresto.

Bibliografia

Buchanan, B. G., & Shortliffe, E. H. (a cura di). (1984). *Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project*. Addison-Wesley. https://www.shortliffe.net/Buchanan-Shortliffe-1984/MYCIN%20Book.htm

Cook, L. M., Grant, B. S., Saccheri, I. J., & Mallet, J. (2012). Selective bird predation on the peppered moth: The last experiment of Michael Majerus. *Biology Letters, 8*(4), 609–612. https://doi.org/10.1098/rsbl.2011.1136

de Castro, L. N. (2007). Fundamentals of natural computing: An overview. *Physics of Life Reviews, 4*(1), 1–36. https://doi.org/10.1016/j.plrev.2006.10.002

Eiben, A. E., & Smith, J. E. (2015). *Introduction to evolutionary computing* (2ª ed.). Springer. https://doi.org/10.1007/978-3-662-44874-8

Garcez, A. d'A., & Lamb, L. C. (2020). *Neurosymbolic AI: The 3rd wave* [Preprint]. arXiv. https://doi.org/10.48550/arXiv.2012.05876

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics, 4*(2), 100–107. https://doi.org/10.1109/TSSC.1968.300136

Russell, S., & Norvig, P. (2021). *Artificial intelligence: A modern approach* (4ª ed.). Pearson. https://aima.cs.berkeley.edu/

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science, 362*(6419), 1140–1144. https://doi.org/10.1126/science.aar6404

Approfondimenti

- Stuart Russell e Peter Norvig, Intelligenza artificiale. Un approccio moderno, 4ª ed., Pearson, 2021 — È il riferimento più completo per la parte dedicata alla risoluzione dei problemi attraverso ricerca, spazi degli stati, euristiche, giochi, rappresentazione della conoscenza e inferenza. Anche se l'edizione più recente dedica molto spazio al machine learning, conserva una trattazione molto solida dell'IA classica e permette di collocarla nel quadro più ampio dell'intelligenza artificiale contemporanea.
- Nils J. Nilsson, Intelligenza artificiale, edizione italiana a cura di Salvatore Gaglio, Apogeo, 2002 — È un testo particolarmente adatto per approfondire la tradizione

dell'IA classica e simbolica. Nilsson è uno degli autori storicamente più importanti nel campo della ricerca euristica e dell'intelligenza artificiale; il volume presenta in maniera unitaria agenti, ricerca, rappresentazione della conoscenza, ragionamento e problem solving. Pur essendo datato, proprio per questo capitolo rimane molto pertinente.

- Rishal Hurbans, Intelligenza Artificiale spiegata in modo facile. Guida illustrata per programmatori curiosi, Apogeo, 2023 — È probabilmente il testo più adatto agli studenti. Attraverso diagrammi ed esempi affronta algoritmi di ricerca, ricerca informata, algoritmi genetici, intelligenza di sciame e machine learning. È particolarmente vicino alla struttura della tua dispensa perché passa dalla ricerca classica agli approcci evolutivi e bio-ispirati mantenendo un livello accessibile.
- Roberto Marmo, Algoritmi per l'intelligenza artificiale. Progettazione, Machine Learning, Neural Network, Deep Learning, ChatGPT, Python, 2ª ed., Hoepli, 2024 — Il volume dedica capitoli specifici a agente intelligente, evolutionary algorithm, expert system e fuzzy logic, oltre agli approcci di machine learning. È quindi particolarmente utile per collegare sistemi esperti e algoritmi evolutivi all'interno di una panoramica più ampia delle tecniche di IA.
- Melanie Mitchell, Introduzione agli algoritmi genetici, Apogeo, 1999 — È il riferimento verticale della selezione per la parte dedicata agli algoritmi genetici. Approfondisce rappresentazione delle soluzioni, popolazioni, fitness, selezione, crossover, mutazione e dinamica evolutiva. È un testo storico, ma rimane molto pertinente proprio perché permette di andare oltre l'introduzione presente nella dispensa e studiare in modo specifico la computazione evolutiva.

Nota editoriale

Per la realizzazione di questi materiali ho utilizzato strumenti di intelligenza artificiale generativa come supporto alla scrittura, in particolare per migliorare la forma del testo, riorganizzare i contenuti, correggere la formulazione e velocizzare alcune attività redazionali.

Lavorando autonomamente alla produzione di questi materiali, cerco infatti di automatizzare tutte le attività che possono esserlo, così da poter dedicare più tempo alla ricerca, alla progettazione e allo sviluppo dei contenuti.

L'intelligenza artificiale non determina però i contenuti dell'opera: la scelta degli argomenti, la struttura, le idee, le interpretazioni, gli esempi e l'impostazione didattica sono elaborati da me. L'IA viene quindi utilizzata principalmente come strumento di supporto alla produzione e alla revisione formale, mentre la responsabilità autoriale e progettuale dei contenuti rimane mia.

Salvo diversa indicazione, questo materiale è distribuito con licenza Creative Commons Attribuzione – Non commerciale – Condividi allo stesso modo 4.0 Internazionale (CC BY-NC-SA 4.0).

È quindi possibile condividere, ridistribuire, modificare e creare opere derivate a partire dal materiale, a condizione di attribuirne correttamente la paternità, di non utilizzarlo per scopi commerciali e di distribuire eventuali versioni modificate o derivate con la stessa licenza.

