



# MagicSolver

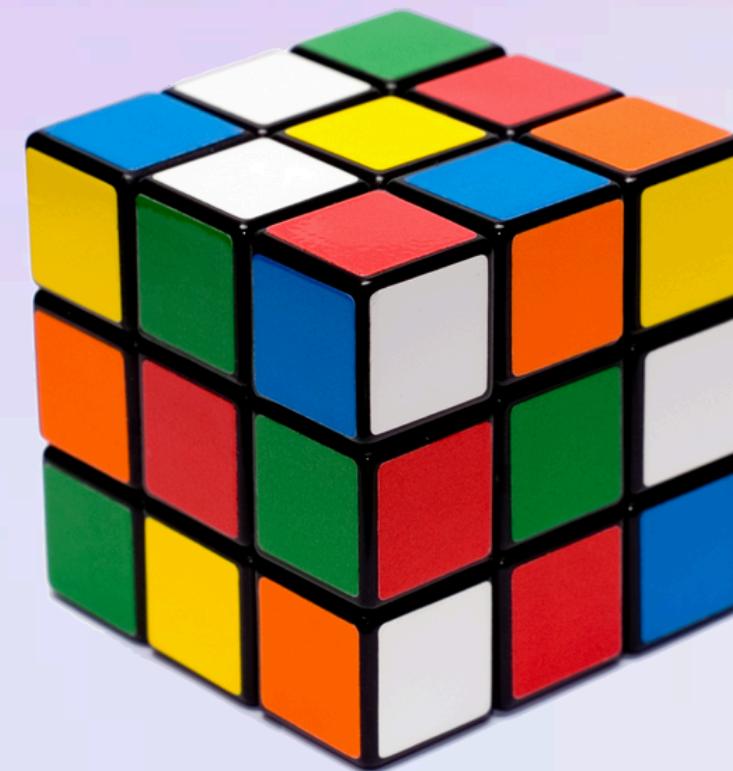
**Non è magia, è intelligenza artificiale**

# INTRODUZIONE

Nato nel 1974 il cubo di Rubik è uno dei rompicapi più affascinanti ancora oggi grazie all'enorme spazio degli stati, prendere scelte in maniera casuale rende la sua risoluzione molto improbabile se non impossibile...



$4,3 \times 10^{19}$

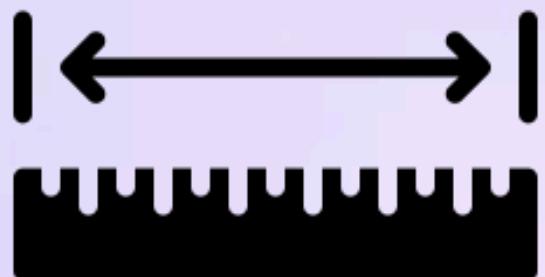


Obiettivo di questo progetto è far risolvere il celebre rompicapo all'intelligenza artificiale, ricercando la soluzione più rapida e veloce

# SPECIFICA P.E.A.S

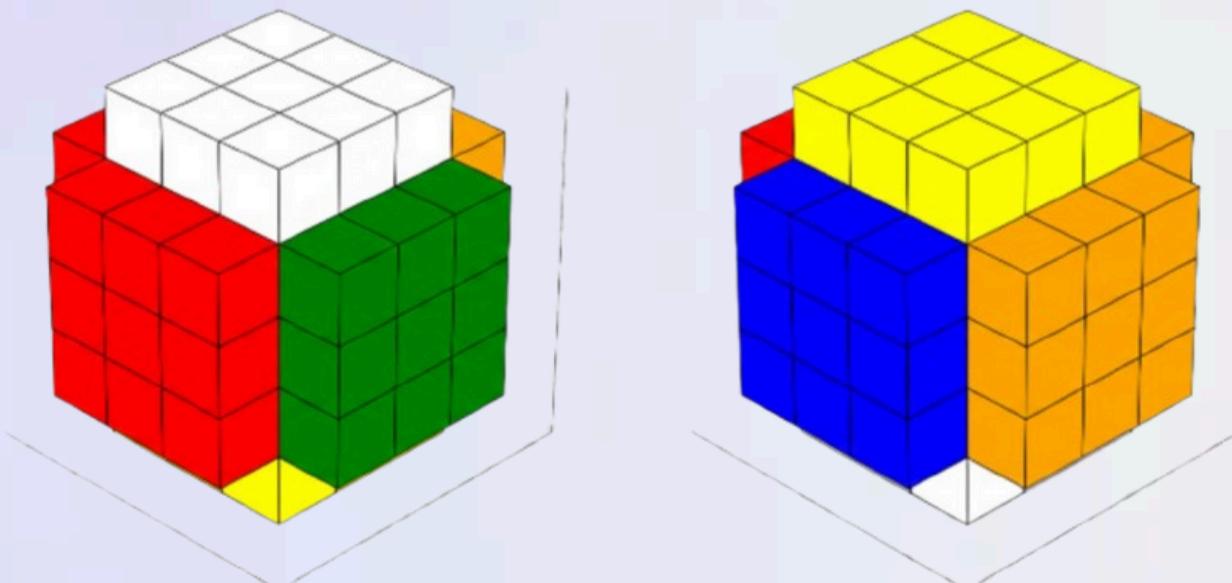
## Performance:

L'agente viene valutato in base alla lunghezza della soluzione



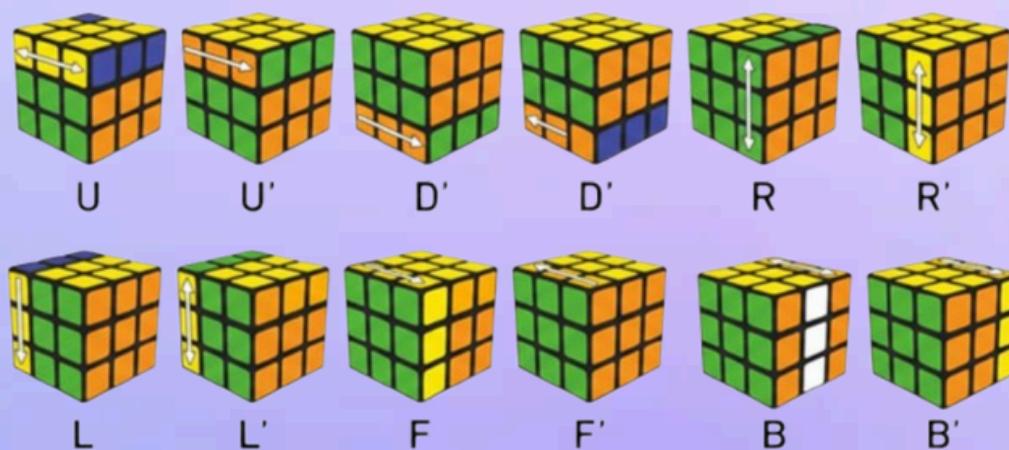
## Environment:

Tensore 5x5x5 che rappresenta il cubo



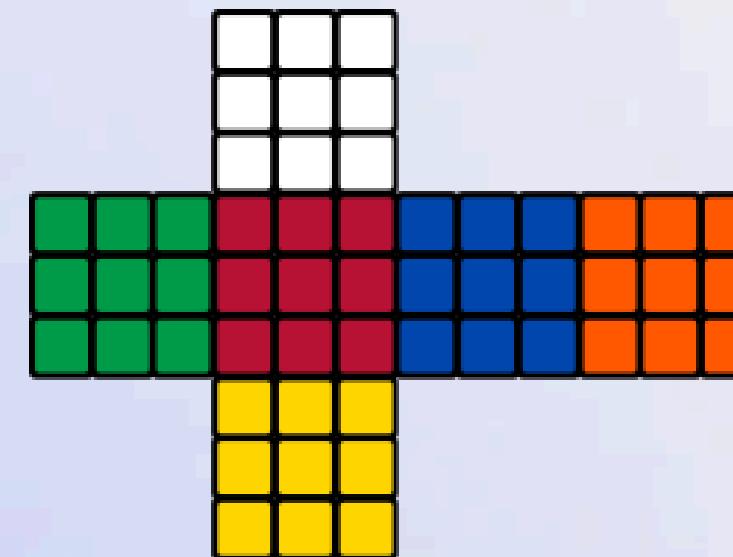
## Actuators:

Insieme delle mosse ammesse



## Sensors:

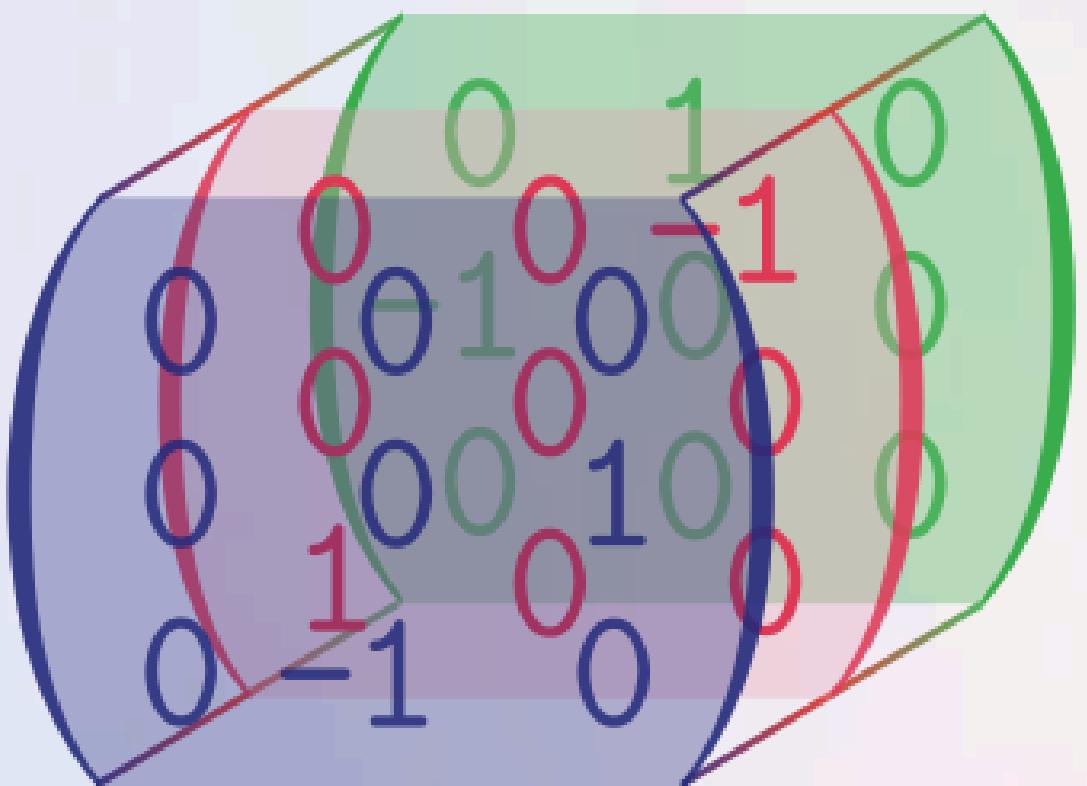
L'utente inserisce le mosse in un cubo 2D



# PERCHÈ IL TENSORE?

Mentre una matrice standard  $3 \times 3 \times 3$  si limita a mappare i colori superficiali, la nostra struttura  $5 \times 5 \times 5$  agisce come uno "scheletro geometrico". Inserendo degli spazi vuoti strategici, separiamo fisicamente i componenti: angoli, spigoli e centri diventano entità distinte con coordinate univoche. Questo elimina le ambiguità durante le rotazioni: non stiamo solo spostando indici, stiamo simulando la meccanica reale del cubo, preservando l'integrità dei pezzi interni che altrimenti andrebbero persi nel calcolo.

$$\epsilon_{ijk} =$$

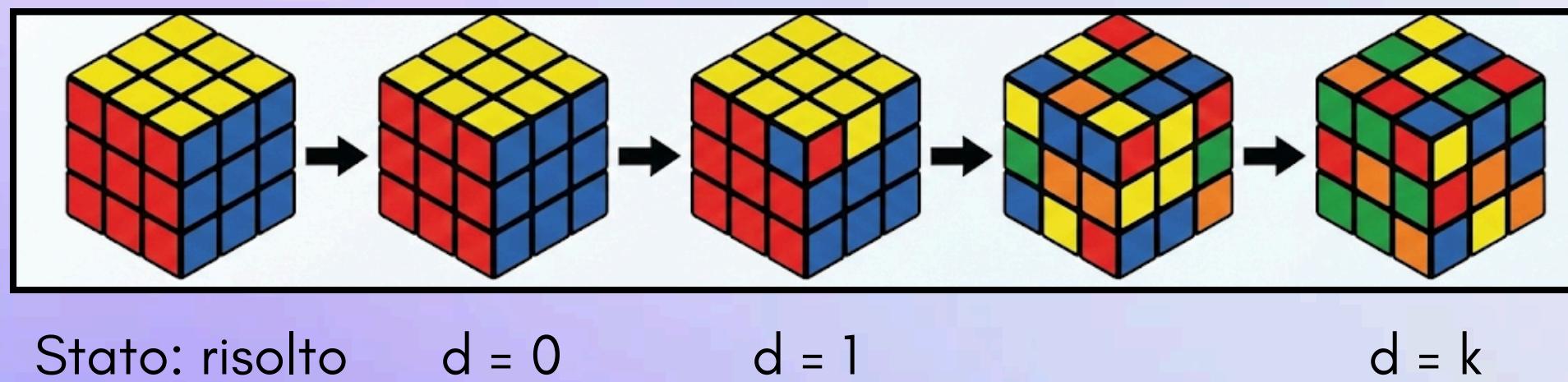


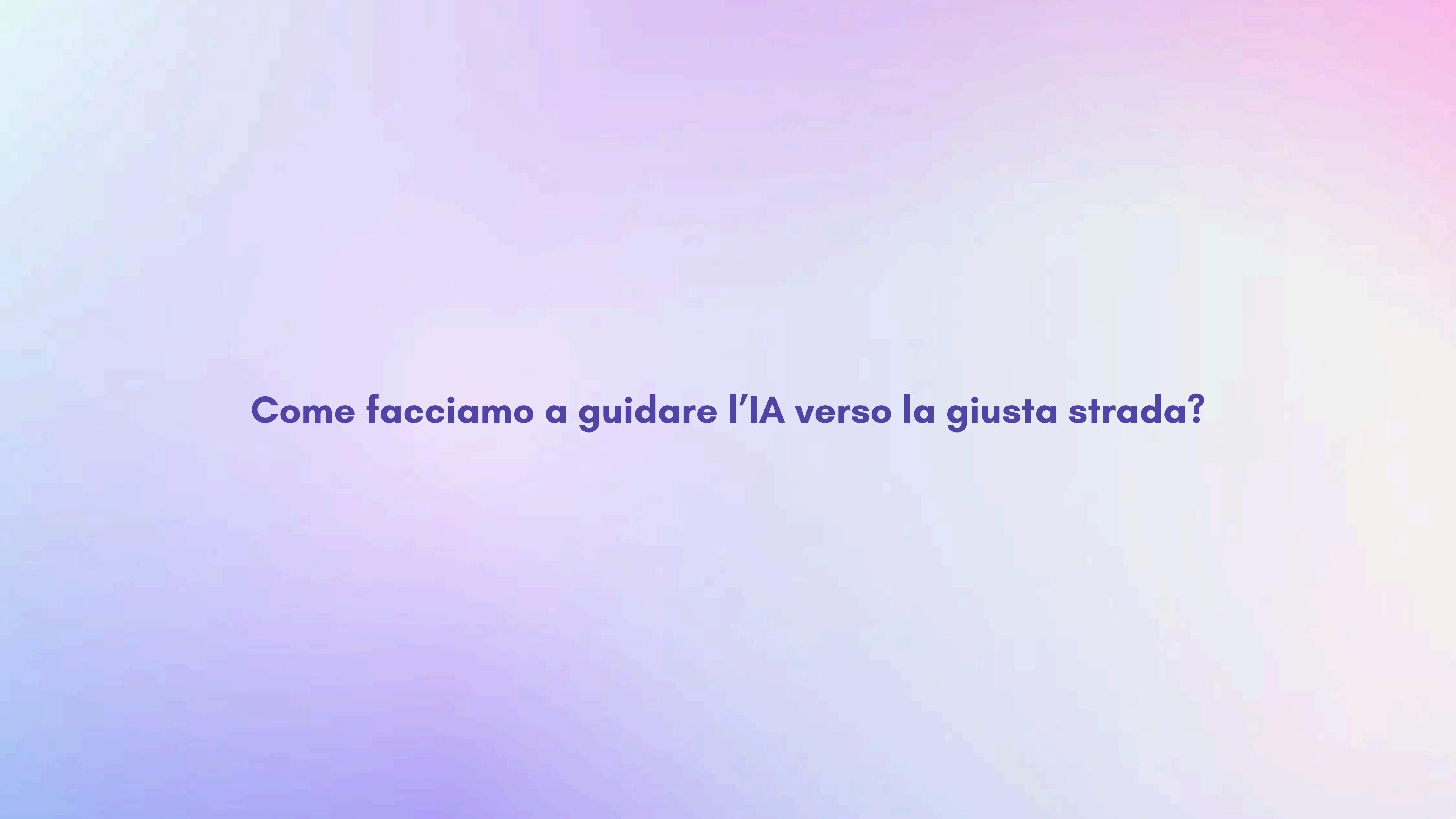
# UN DATASET SINTETICO: PERCHÈ?

Uno dei motivi principali per cui abbiamo optato per un dataset sintetico è la mancanza di dataset pubblici che associno uno stato del cubo alla sua distanza dalla soluzione.

Abbiamo quindi pensato a una tecnica di Reverse Scrambling, ovvero partire da un cubo risolto e applicare mosse casuali a ritroso, così la distanza viene associata per costruzione, evitando uso di algoritmi che avrebbero reso il processo lento e costoso.

Inoltre questa tecnica ci permette di avere un dataset equamente distribuito per ogni profondità (da 1 a 20 mosse)





**Come facciamo a guidare l'IA verso la giusta strada?**



**Random Forest**

**Beam Search**

# LA SCELTA DEL RANDOM FOREST È DOVUTA DA DIVERSI FATTORI

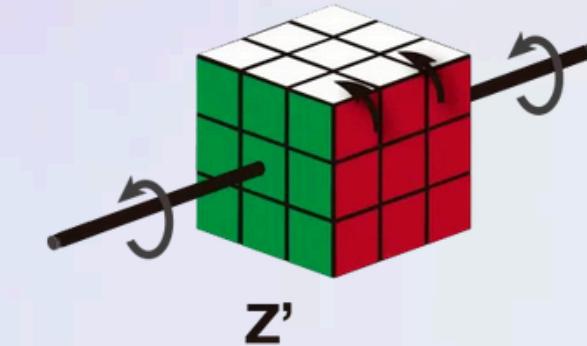
## Veloce per l'algoritmo di ricerca

Il beam search esplora un elevato numero di nodi per ogni livello. l'albero usa semplici if-else rendendo la sua esplorazione molto più veloce



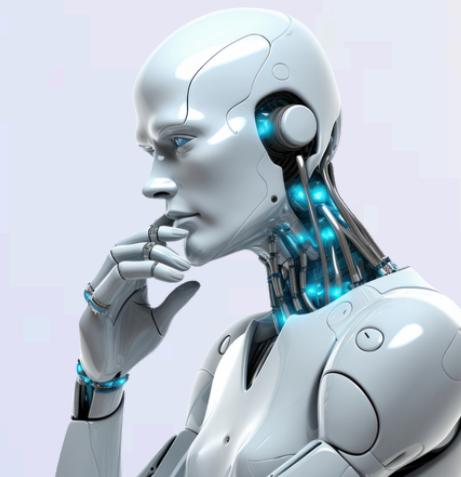
## Ottimalità su dati tabellari

I dati che rappresentano il cubo sono strutturati attraversi dei confronti l'algoritmo riesce a comprendere pattern e interazioni



## Non linearità dei dati

Gli stati del cubo cambiano drasticamente la loro rappresentazione rendendo i modelli lineari inefficienti



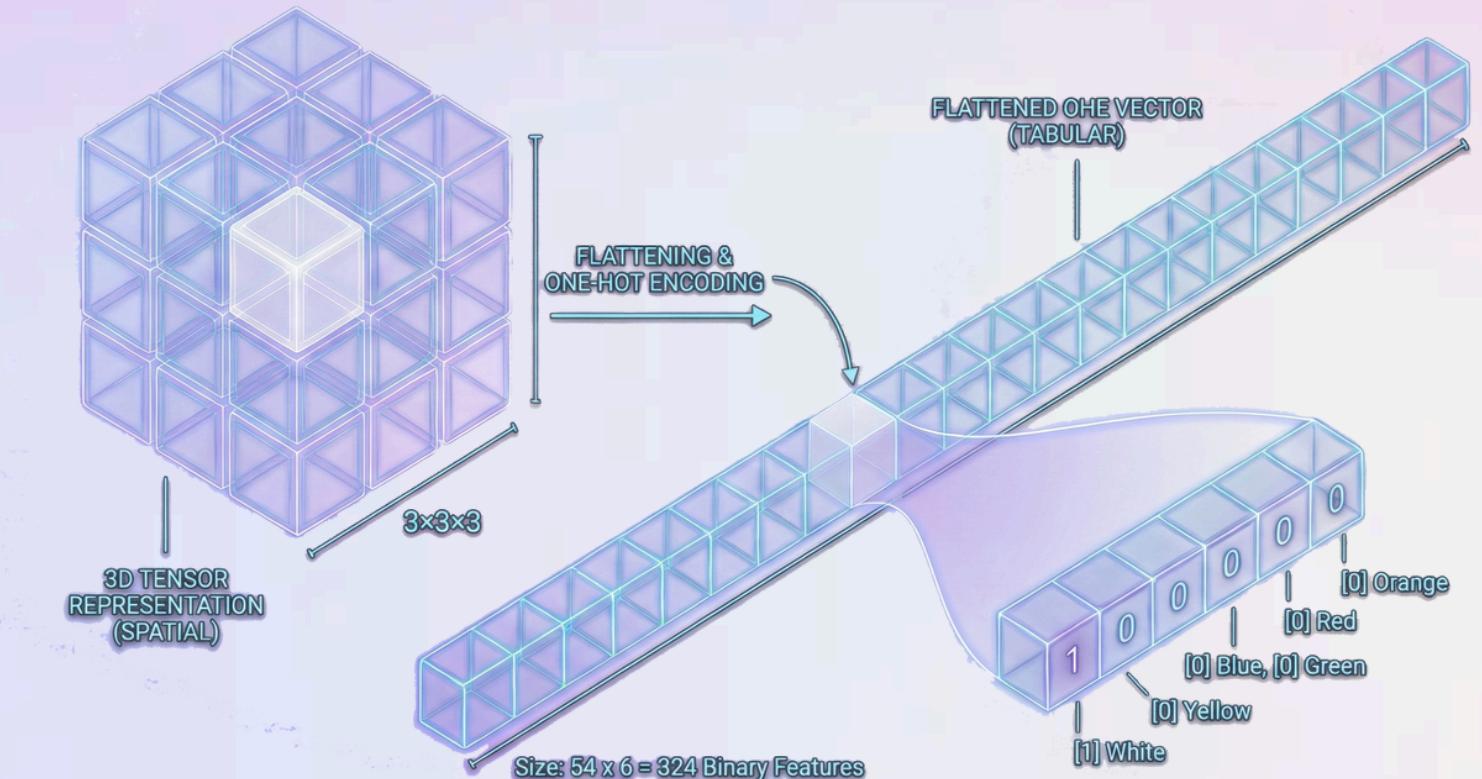
## Robustezza all'overfitting

Ogni albero è formato da dati diversi, in questo modo gli errori di uno vengono ridotti dalla presenza degli altri

# ONE HOT ENCODING (OHE): COME IL MODELLO VEDE IL CUBO

Il modello non riceve il tensore, poichè il Random Forest riceve dati tabellari... Quello che succede è che il tensore viene "appiattito", in modo tale da essere leggibile al modello e conservare le sue informazioni spaziali.

Ma come avviene questo "appiattimento"?



Un cubo è formato da 54 sticker e 6 colori possibili. Invece di rappresentare ogni colore con un numero (bias matematico), usiamo un vettore di 6 elementi per ogni sticker.

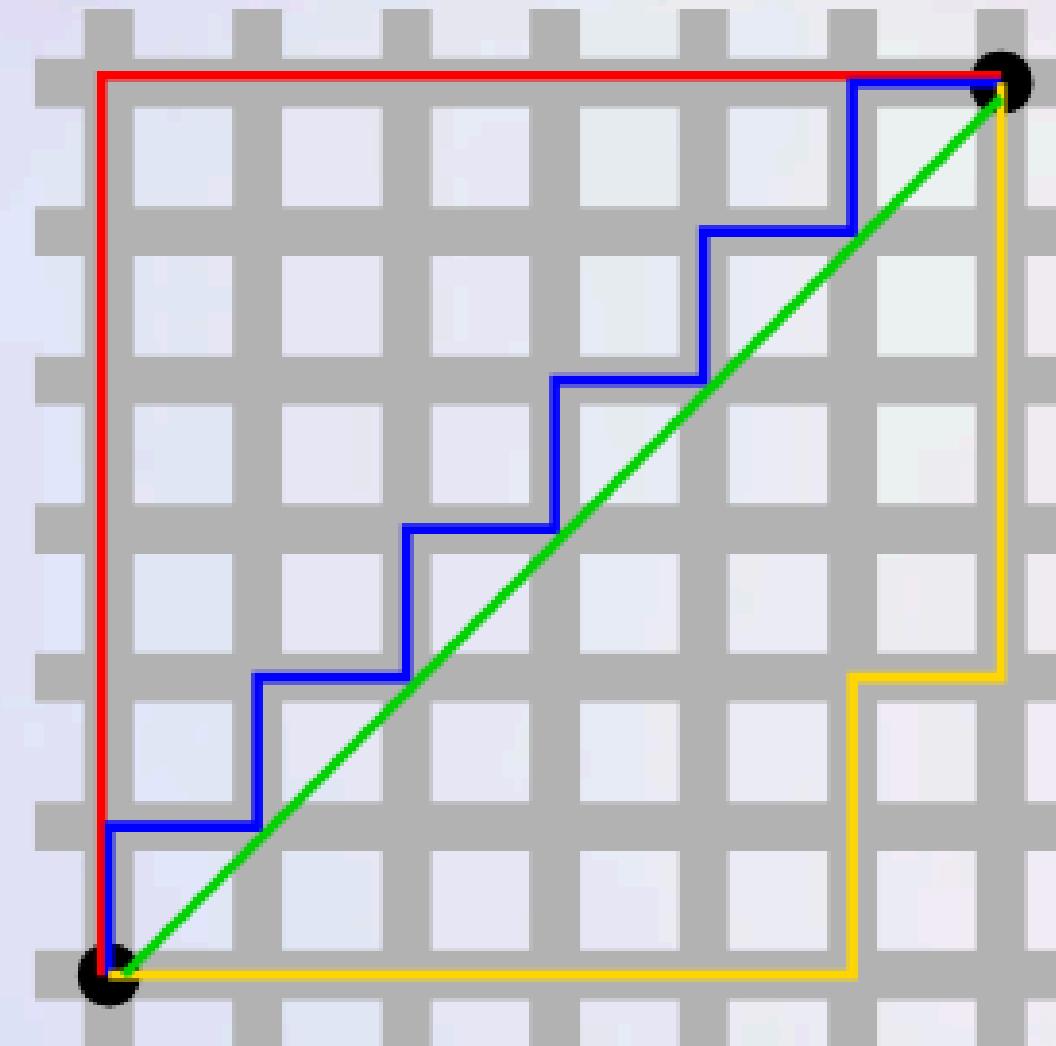
In questo modo avremo 324 feature indipendenti in totale. Il Random Forest impara autonomamente la geometria del cubo senza assunzioni nascoste sui colori.

## DISTANZA DI MANHATTAN: UN'ALTRA STRATEGIA

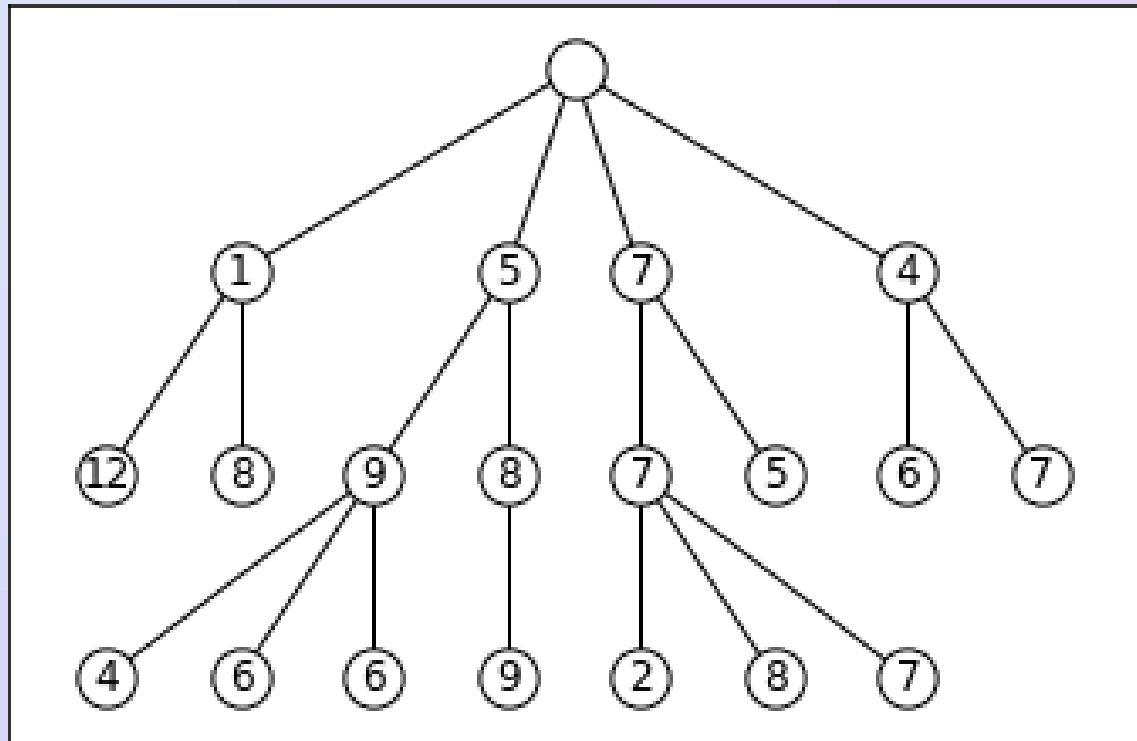
Un'altra tecnica di feature engineering utilizzata è stata la distanza di Manhattan.

Non passiamo i dati grezzi al Random Forest, ma la distanza di ogni cubetto rispetto alla sua giusta posizione.

Il vantaggio è che l'algoritmo in questo modo riceve dati già strutturati invece di dover intuire la struttura da dati grezzi.



# BEAM SEARCH



La scelta di questo algoritmo è stata dettata dalla sua velocità a discapito della completezza. Per ogni livello selezioniamo solo un numero limitato di nodi tra quelli che hanno punteggio migliore.

## Modifiche apportate:

Per aumentare ulteriormente la velocità dell'algoritmo abbiamo apportato le seguenti modifiche:

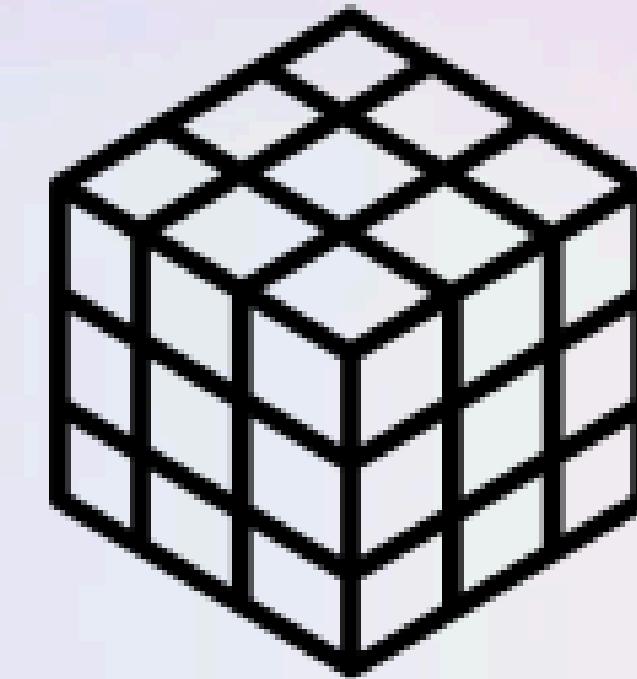
- Tabella hash per salvare i nodi già visitati
- Profondità e larghezza di selezione adattiva
- Riavvio se si trova in una situazione di stallo



# VALUTAZIONI FINALI

OHE è stato preferito poiché presenta un valore di MAE (errore medio assoluto) leggermente inferiore rispetto a Manhattan.

Un'altra osservazione significativa riguarda il Random Forest, il modello infatti attraverso OHE ha appreso autonomamente la struttura geometrica del cubo.



Nonostante il sistema non riesca a risolvere tutti i cubi possibili, riesce comunque ad avere un success rate considerevole, che ci rende soddisfatti del risultato finale

# CONCLUSIONI

## COSA CI HA INSEGNATO MAGICsolver?

- **Obiettivo Raggiunto:** Ci ha dimostrato che il Machine Learning Supervisionato può sostituire efficacemente le complesse euristiche matematiche tradizionali.
- **L'Intuizione Artificiale:** Non serve "insegnare la geometria": con abbastanza dati, l'IA intuisce la distanza dalla soluzione guardando solo sequenze di 0 e 1.
- **Il Compromesso (Trade-off):** Abbiamo sacrificato la perfezione matematica (soluzione ottima) per ottenere un sistema rapido, flessibile e semplice da utilizzare

