

AGENTI ARTIFICIALI INTELLIGENTI



Misura delle prestazioni (definite dall'esterno), in base a

- Conoscenza → può anche essere appresa a runtime
- azioni
- percezioni

Esempio : aspirapolvere

posizione, sporcozida

↖
A, B

↖

sporco, pulito

PERCEZIONE: [A, sporco]

- Azioni :
- vai sx → Aspira
 - vai dx → Noop

Conoscenza : in questo esempio: vuota

Prestazioni : quanto ha pulito

[quanto consuma? quanto tempo ci mette?]

AGENTE IDEALE

→ compie l'azione che massimizza le prestazioni in base a ciò che ha percepito fino a quel momento e la sua conoscenza progressa.

RAZIONALITÀ ≠ OMNISCIENZA

Agent program → implementa la Agent function

Implementazione → se if ... then ... può funzionare,
ma ogni if ... then sarebbero

TROPPI, anche la memoria scriverebbe
troppe

non può nemmeno essere statica ma dinamica

AMBIENTE OPERATIVO

Performance da ottimizzare

Environment in cui devo agire, fisico o virtuale

Actuators per eseguire azioni nell'ambiente

Sensors (telecamere, gps)

ES. tessuto automobilistico

AMBIENTI

→ accessibile / non all'agente, i sensori rilevano tutto il necessario?

visione → completo → scacchi

→ parziale

→ nulla

→ deterministico / non deterministico / strategico

↳ gli effetti di un'azione

sono noti

effetti

alternativi;

maggiori probabilistici

→ episodico / non episodico: la qualità di un'azione dipende dalle precedenti.

Quello episodico è più facile, a ogni nuova scelta ci si può riferire delle precedenti

→ statico / dinamico / semidinamico → l'ambiente non

↳ mentre prendo la decisione su cosa fare lui. cosa cambia, → taxi driver → cambia ma devo comunque metterci poco tempo

scacchi

→ Discreto o continuo # percezioni, basso o alto?
azioni, basso o alto?

→ singolo agente / multi agente

↳ cooperativo / competitivo

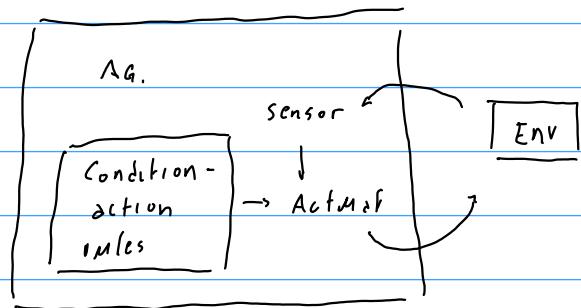
↳ controllo centralizzato /
controllo distribuito

TIPI DI AGENTI

→ Simple reflex

agents

ambiente episodico,
senza memoria



"c'è sporco" → pulisci

"no sporco" → non pulire

non rilevo la posizione, quindi
dovrei spostarmi a caso

Non è in grado di accorgersi che hanno finito.

→ Model-based reflex agents

In più hanno la conoscenza dello stato del mondo,

modelli

↳ stato

del mondo

es qui viene detto all'inizio che c'è nella stanza

↳ proprie

A e in seguito alle sue azioni può

aggiornare il suo stato.

azioni

E' in grado di accorgersi che ha finito.

Il modello è lo stato del mondo + le azioni.

→ Agente basato su gol: non solo considera
presente e passato ma anche futuro.

necessità di un piano, un percorso,

non hanno solo sensori ma anche un Gol

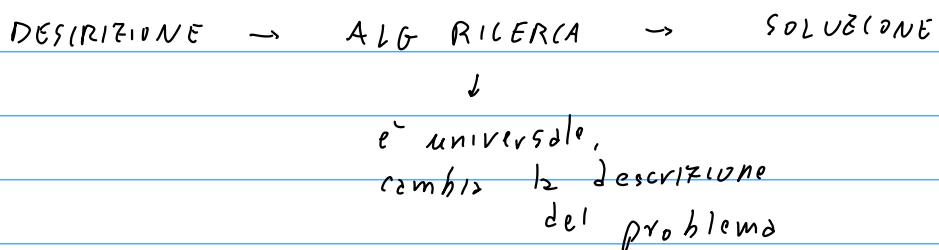
→ Agente basato sull'utilità, oltre a un obiettivo

sanno scegliere quali azioni sono più o meno utili

GOL BASED AGENTS

NON usano un database di regole
hanno un modello, fanno un ragionamento a catena
non eseguono in blocco le azioni ma una alla volta
Eseguono una ricerca in uno spazio di possibilità
Richiedono una specifica del gol, le variabili dello stato del
mondo, devono avere note le azioni e le variabili
(es. la posizione).

Trovare la sequenza di azioni si ripete la percezione
dopo ogni azione eseguita singolarmente.
ogni azione trasforma lo stato corrente.



FORMALIZZAZIONE DEL PROBLEMA

Lingaggio della logica / strutture dati → descrizione
del problema

Azioni

Erat, mizale

Erst- oblettivo o Goc test

ES: PROBLEMA A SISTAU UNICO

Ho le rivelate osservabilità del mondo

L'agenzia Ha dei sensori che gli dicono la presenza di sporco e dove si trova

VARIABILI → POSIZIONE, SPORCIZIA

A, B $\mathcal{E}_1, \mathcal{N}_0$

$$2 \times 2 \rightarrow 4$$

strategic russell.

obiettivo: non c'è spazio da nessuna parte

\Rightarrow 2 possible blocks; $\begin{cases} A \text{ NO} \\ B \text{ NO} \end{cases}$

$\rightarrow \{ A \text{ NO } \text{ or } A \}$

\downarrow { pos A
A no B no
pos B

"stato singolo" = l'oggetto sa sempre che in uno stato preciso

RISOLVERE il problema: trovare un percorso nel grafo degli stati (Automa a stati finiti)

es: rompicapo dell'8

STATI: posizioni dei numeri + casella vuota: 9 variabili

Posso prevedere lo spostamento della

casella vuota,

muovo $\rightarrow N, S, E, W$,

Meno azioni, sono ordini esponenziali
di combinazioni in meno

Stati raggiungibili: 10^{25}

è impossibile elencarli tutti,

non c'è l'algoritmo quindi lo si formalizza come un problema di ricerca

Non c'è una ricerca su grafo, il grafo intero non

lo scriviamo in una volta ma lo costruiamo man mano

Problema delle N regine: capire come è fatto lo stato finale,
se esiste

è possibile che le N regine non si attacchino sulla scacchiera

Divenuta un problema di ricerca

Stato iniziale: nessuna regina sulla scacchiera

N regine, 64 caselle, 64^N ($64 \cdot 63 \cdot \dots \cdot (64-N+1)$)

Si può riformulare in maniera più intelligibile

"una regina per ogni colonna", 8^N

"ogni regina su una riga diversa" $8!$

Problema di ricerca - Automata a stati finiti

↳ stato iniziale,
stato finale,
stati intermedi
raggiungibili,
azioni

↓
può rappresentare un problema
di ricerca a stato singolo

STATO SINGOLO → L'AGENTE HA COMPLETA OSSERVABILITÀ

PROBLEMA A STATI → L'AGENTE HA OSSERVABILITÀ PARZIALE,
MULTIPLI
non sa di essere in uno stato singolo,
particolare perché ha parziale informazione

STATI MULTIPLI

→ completa ignoranza: es. aspirapolvere, non sa in quale
dei possibili stati (8 si trova),
Le azioni restano le sa di trovarsi in un sottoinsieme
stesse, è ciò che di stati, chiamato BELIEF STATE,
percepisce del mondo che è cambiato.

Concettualmente può essere rappresentato con un grafo ma
operativamente no

PROBLEMI CON CONTINGENZE

Parziale osservabilità, es: l'aspirapolvere sa dove sta
e sa se ciò sporco nella stanza corrente ma non sa se nelle
altre c'è sporco o meno

+ INCERTEZZA SULLE NIE AZIONI: se pulisco dove è già pulito
potrei depositare polvere → non determinismo

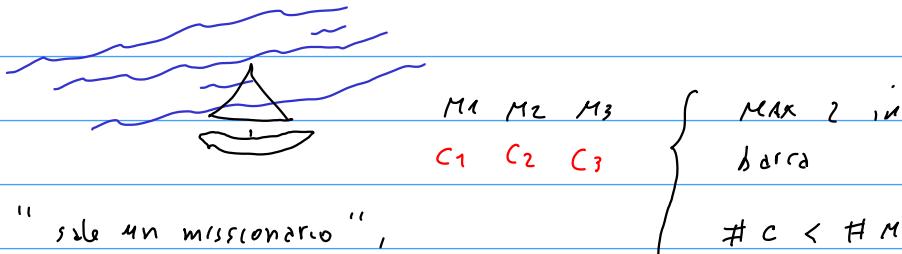
NON posso garantire di risolvere il problema,
quindi si deve equipaggiare l'agente con strumenti in più,

es. rilevare lo sporco,

si genera un albero con un numero di stati superproporzionale
⇒ si usano euristici.

Come si formalizza un problema

Problema dei missionari e dei cannibali (e family crisis)
è importante scegliere bene variabili e azioni.



L'azione è "sala un missionario",
non "sala M_1 ".

↓ ASERAZIONE ...

L'azione è "sposta" Z^A, INC, Z^C
 IN, IC

5 azioni possibili.

DX, SX? non serve distinguere se l'azione è "sposta
verso il lato opposto",
il problema è 2 stato unico.

INFO essenziali: DX: #M, #C, c'è la barca sulla riva DX

SX: #M, #C, " " " " " SX

~> DX: #M, #C, B(arca), sulla SX non scrivono nifo, si
ricevono dalla dx.

Nel diagramma degli stati non ci sono anche gli stati
potenziali ma solo quelli leciti

3 variabili e 5 azioni

Stato iniziale: $X=3, Y=3, Z=1$

Stato finale: $X=0, Y=0, Z=0$

FAMILY CRISIS

risorse limitate: tempo, soldi,

qual è tempo:

stato del problema

[P₁, P₂, P₃, P₄, P₅, L, T]

Osservabilità totale,

J J J J J J ↳ o...30
--- TF --- T,F

Le azioni impiegano sempre

il tempo rischioso, è deterministico, si può scrivere

determinare l'elemento non deterministico e il problema diventerebbe
parzialmente osservabile.

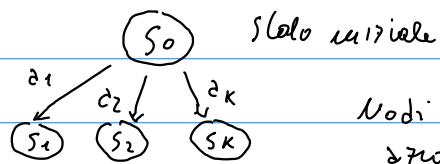
Stato finale: deve soddisfare il gol test P_{A,L} = True,
non importa T.

ALGORITMI DI RICERCA

sono generali, ma con elementi di specializzazione
perche' allora non si usa sempre? perche' un algoritmo
specializzato e' meglio, questi ci usano quando non
ci sono algoritmi classici o se ci sono non va bene
perche' il grafo sarebbe troppo grande.

COSTI → ONLINE → eseguire le azioni
↳ OFFLINE → trovare le azioni

Albero di ricerca:



Nodi foglia: nessun
azione e' piu' eseguibile,
TCO (family crisis)

Profondità = profondità

maggioria dell'albero

Dopo al massimo molto non
si puo' piu' fare niente

Branching factor = #max figli;

fattore di divisione
massimo

Nel caso dei cammini l'albero
puo' essere finito.

Albero ≠ diagramma. (spazio)

L'algoritmo genera l'albero navigando lo spazio degli stati.

Sg = stato gol, una volta trovato si smette di costruire l'albero

Lo stato finale non e' solo nei problemi delle N regine.

Ci sono algoritmi che trovano la sol ottima e altri che trovano
una sol sub ottima.

Espandersi in modo significa simulare tutte le possibili azioni
possibili in un modo.

ALGORITMO GENERALE

- 1) $L \leftarrow$ stato (nodo) iniziale
- 2) IF L empty THEN return fail
- ELSE :

 - $n \leftarrow \underline{\text{choose}}(L)$
 - 3) IF GOALTEST(n) return n
 - 4) remove n from L and add to L n's children
 - 5) GOTO 2)

$L \rightarrow$ frontiera, contiene le foglie dell'albero di ricerca esaminato finora.

choose e add children sono le feature specifiche

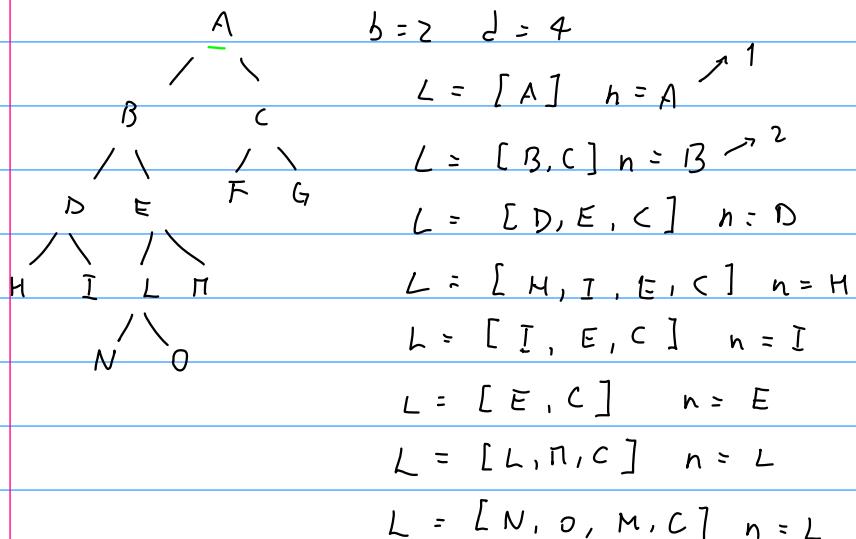
SELECTA : potrebbe essere il primo nodo della lista

e i figli vengono aggiunti in testa : LIFO

oppure in coda : FIFO ;

profondità

\hookrightarrow doppiezza



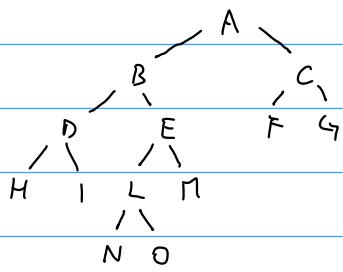
I nodi sono strutture dati contenenti

\hookrightarrow le variabili che descrivono lo stato

\hookrightarrow il percorso che mi ha portato lì (z_1, z_2, \dots)

Nel problema delle N regine non importa il percorso ma lo stato finale, nel problema dei cammbaldi è il contrario

FIFO



$$L = [A]$$

$$n = [A]$$

$$L = [B, C] \quad n = B$$

$$L = [C, D, E] \quad n = C$$

$$L = [F, G, D, E] \quad n = F \quad L = [D, E, F, G]$$

$$L = [G, D, E] \quad n = G \quad L = [E, F, G, H, I]$$

$$L = [D, E] \quad n = D \quad L = [F, G, H, I, L, M]$$

$$L = [E, H, I] \quad n = E \quad L = [G, H, I, L, M]$$

$$L = [H, I, L, M] \quad n = H \quad L = [H, I, L, M]$$

$$L = [I, L, M] \quad L = [L, M]$$

$$L = [M, N, O] \quad L = [N, O]$$

Ricerca in completezza

completenza spaziale (quanto
è lungo L)

b^d al massimo

$$\mathcal{O}(b^d)$$

completenza temporale

(quanti modi vengono
visitati) sicuramente

saranno visitati i modi
in fondo, b^d ,
e non solo, anche quelli
primi.

$$1 + b + b^2 + \dots + b^d$$

$$\sum_{i=0}^d b^i \quad \mathcal{O}(b^d)$$

Terminazione

(esecuzione di return),
se entro una sol. la trova,
anche se ci sono altri modi termina,

se non c'è sol. può andare
avanti all'infinito

Ricerca in profondità

completenza spaziale,

lucare in b e d,

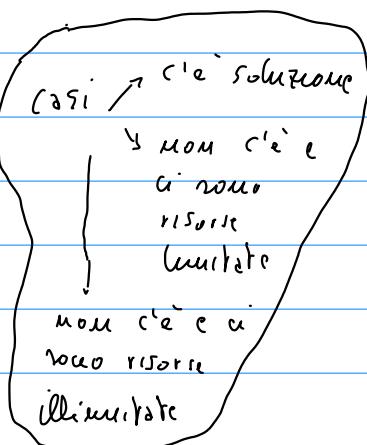
$$\mathcal{O}(b \cdot d)$$

basata

completenza temporale

stesa della ricerca in
completezza

Terminazione: se si mette una lista
di stati e se c'è la sol. allora termina,
senza lista c'è sol. e differenza di
pluri, potrebbe non terminare



però il numero di stati c'è finito
 (la differenza dei modi), quindi
 posso aggiungere una lista di stati visitati, se uno stato è già stato visitato allora non lo si inserisce nella frontiera, quindi anche qui la terminazione è garantita.

Qualità della soluzione:

(profondità del nodo)
 Trova la sol ottima, quando esiste e quando gli archi hanno costo uniforme

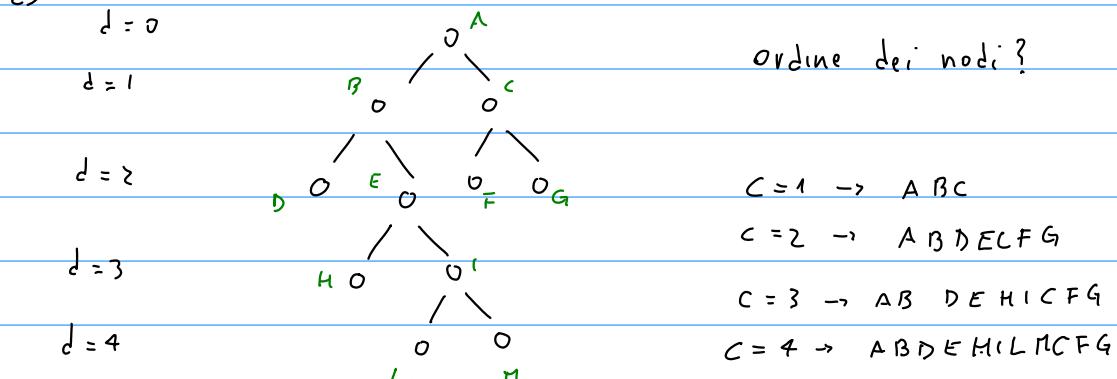
Non garantisce di trovare l'ottimo, per garantirlo si deve lasciare la terminazione solo quando L si svuota.

ITERATIVE DEEPENING

Algoritmo a complessità spaziale $O(b \cdot d)$ e che trova la sol ottima quando esiste.

Se non c'è un vincolo sulla lunghezza della soluzione non termina.

ES:



Supponiamo non ci sia il gol: andrebbe avanti all'infinito

MIGLIORIAMO L' ALGORITMO:

Bisogna farlo terminare, e non si può dire "ritorna quando la lista si svuota" perché la lista si svuota a ogni incremento di c

PSEUDOCODE

```
2: C = 1, stop = TRUE  
3: L = [empty initial]  
   IF L empty  
     IF stop = FALSE  
       C = C + 1  
       stop = TRUE  
       GOTO 2  
     ELSE  
       RETURN "Fail"  
     n = first-node(L)  
     IF GOAL TEST(n)  
       RETURN n  
     ELSE  
       L = L - {n}  
       IF DEPTH(n) < c  
         L = PUSH(CHILDREN(n), L) (e' una pila, DFS)  
         LIFO  
       ELSE IF CHILDREN(n) not EMPTY  
         stop = FALSE  
       GOTO 3
```

Con quest. algoritmo c'è un controllo in più:
si accorgi che costituisce una frontiera sotto la quale non c'è niente.

Prima c' sarebbe stato incrementato anche una volta toccato il fondo.

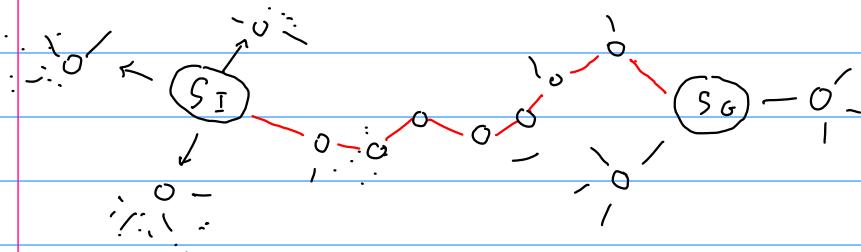
Di default stop c' è TRUE, se si mette a FALSE

Quando sono a profondità massima, se ho almeno un figlio, allora non termino

RICERCA BIDIREZIONALE

Parto dalla fine e vedo a ritroso.

Ricerca in parallelo,
multi-threaded



Criterio di terminazione:

+ Criterio di terminazione: se tra nodi

so c'è intersezione tra

figlio uno è presente nell'altra frontiera
allora ho trovato un percorso

le due frontiere

le due ricerche parallele sono in痘 p(0)rtà =>

=> trovo il percorso se esiste

le due ricerche sono in profondità => non c'è

detto che trovo il percorso e ce lo trovo.
potrebbe essere subottimo

VANTAGGI : ricerca $b^d \rightarrow b^{d/2}$ e' un grande vantaggio
molto più che $\left(\frac{b}{2}\right)^d$

BENCHMARK

Nella pratica Bidirectional Search non porta grandi vantaggi.

Evitare cicli in DFS: memorizzo i nodi che vedo.

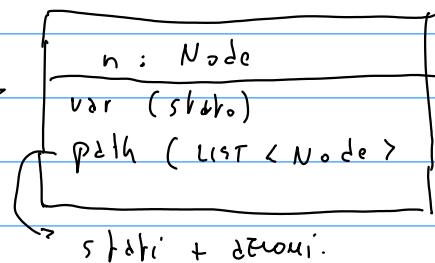
sul percorso dalla radice a n,

h' è una struttura dati con →

Per ogni modo fai

vergono che non sia

presente nel percorso fra la
radice e n.



RICERCA A COSTO UNIFORME :

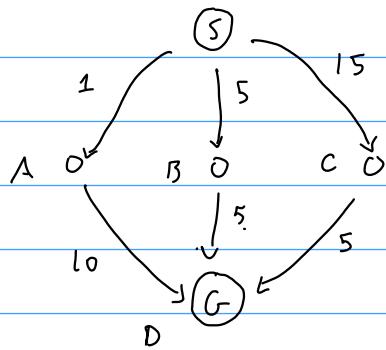
le svolte non hanno costo uniforme.

vincolo : $\text{cost}(i) > 0$

Si memorizza il costo del percorso dalla radice al nodo

L'ordina secondo il costo. Si preleva sempre il modo a
costo più basso.

ES:



Ordine di visita:

S

$$L = [S]$$

$$L = [A, B, C] \quad C = [S]$$

$$L = [B, C] \quad D(11) \quad C = [S, A]$$

$$L = [D, C] \quad D(10) \quad C = [S, A, B]$$

Dobbiamo valutare i nodi non solo in base alla profondità,

$$f(n) = g(n) + h(n)$$

~~n~~ → Goal

So → n

può essere
anche infinito.

Serve per arrivare primo al nodo goal,

se h finge esiste non sbaglia mai a prelevare n dalla frontiera L

BEST FIRST SEARCH

$$h(n) = 0 \rightarrow \text{ampliata}$$

$$h(n) = 0 \text{ e } g(n) = \text{costo da radice a } n$$

Se le heuristiche non sbagliano

per eccesso troverà la sol ottima

↓
costo uniforme

A^* = best first search con h ammissibile.

Ottimalità di A^* : il fattore di discriminazione deve essere finito!
non avremo neanche la terminazione.

E se ci dovranno costi nulli o negativi (guadagno)?

no garanzia di ottimalità, anche con costi uguali a zero.

L'heuristica rende la ricerca informata.

es. rompicapi dell'8 → heuristica: #caselle malposte.

distanza di manhattan

→ Progettare euristiche:

→ problema rilassato (con meno vicini)

→ sotto problema → sottoproblema dell'8 → sottoinsieme:

↓

Se voglio che
l'euroistica rimanga
ammisibile:

soltimma del sottoproblema

mettiamo il

cotro della

soltimma del

*	2	*
8	*	4
*	6	*

sottoproblema come euroistica
del problema originario.

→ Confrontare euristiche

n nodi visitati dall'algoritmo.

branching factor effettivo

$N+1$ nodi, profondità:

$$N+1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

b^* medio, ottenuto dalla sol con A^* con h_1, h_2, \dots (euroistiche
può b^* è vicino a 1 meglio è, significa che ha da confronto)
visitato meno nodi multilivello.

Una ricerca euristica più costosa fa quadruplicare nel numero
di nodi visitati (di meno) ma costa di più calcolarla
(e viene calcolata tante volte!)

TRUCCO: si possono pre-calcolare i valori euristici
"PATTERN DATABASE" → OFFLINE

→ Combinare euristiche:

→ Prendere il max → 10000 volte meglio di manhattan

→ 1000000 volte più veloce di A^* con Manhattan ...

→ Prendere il min non ha senso

→ Prendere la somma non mantiene l'ammisibilità

PATTERN DATABASE DISGIUNTI

es. sottoproblemi 1, 2, 3

$$h_{pd} : h_{p1} + h_{p2} + h_{p3}$$

come diventa ammmissibile?

ITERATIVE DEEPENING A* / Best First Search

il cui non viene scelto incrementalmente

$$\text{ma in base alla funzione } f = g + h$$

In A* non conta tanto la profondità

$$\text{ma il valore della funzione } f = g + h$$

$$(S_0) \quad f = \tilde{g}^0 + h$$

valore euristico,

c'è il primo taglio.

$$\overline{\dots} L$$

$$0 \xrightarrow{f \leq \tilde{f}} ?$$

$h=0 \rightarrow$ ricerca a costo uniforme

$h=0$ e costi uguali \rightarrow ricerca in cieca

Per risparmiare spazio:

si può rilassare l'ipotesi della ammmissibilità. \rightarrow perdendo la garanzia dell'ottimalità

Il problema delle N regine diventa risolvibile anche con $N = 1000\ 000$

Euristica consistente (o monotona):

$$h(n) \leq c(n, \alpha \rightarrow n') + h(n')$$

consistenza \Rightarrow ammmissibilità, h non decresce mai,

\Rightarrow quando incontro un nodo sono sicuro che prima l'avevo incontrato passando da un percorso più breve, quindi

se lo riincontro posso fare pruning, senza controllarne

A* con $h(n)$ monotona e lista closed \Rightarrow

ottimalità per grafi di ricerca

RICERCA LOCALE

non ci serve più l'ottimo, ci basta una soluzione, meglio se di buona qualità.

(= RICERCA GLOBALE)

HILL CLIMBING (GRADIENT DESCENT)

non ho più la memoria ma il modo corrente

no albero di ricerca, solo nodo corrente, nodi successori.

complessità spaziale nulla

è estremamente veloce ma ha problemi nei minimi locali, spalle, izzane.

HILL CLIMBING CON MOSSE LATERALI

HC STOCHASTIC A aggiungere casualità all'euristica, siccome

↓
genera tutti gli stati successivi! l'euristica a volte sbaglia ogni tanto e disobbediamo
probaabilità maggiore per stati successivi migliori COSTOSO

HC STOCHASTIC B prima scelta migliorativa random

Si per GRANDI VICINATI (per grandi vicinali, meno costoso, non genera tutti i successori ma il primo che migliora)

HC STOCHASTIC C ↓, rischia casuali e mosse laterali

si applica solo dove lo stato iniziale non è

importante e lo posso scegliere io

(N REGINE ✓ 100% problemi risolti con 8 regine)

Rompicapo dell' 8 ✗

SIMULATED ANNEALING :

$\Delta E = \text{next.value} - \text{current.value}$ (next is a random successor of current).

if $\Delta E > 0$

 current \leftarrow next

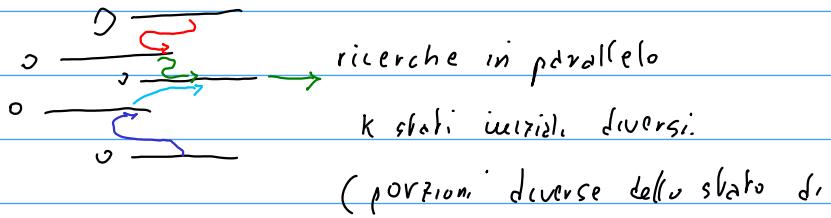
else

 current \leftarrow next with probability $e^{\frac{\Delta E}{T}}$

ΔE : maggiore è il peggioramento ($\Delta E < 0$) minore è la probabilità che il cambiamento sia effettuato.

quando $\Delta E = 0$: potrei dare $p=1$ (mosse laterali).

LOCAL BEAM



Scegliamo i k migliori

successori tra tutte le ricerche

percorri scendenti tendono ad essere abbandonati

Fattore sbilenco per evitare che tutte le k ricerche fluiscano in un'unica area

TABU SEARCH (deterministica)

memoria locale aumentata con una memoria breve,

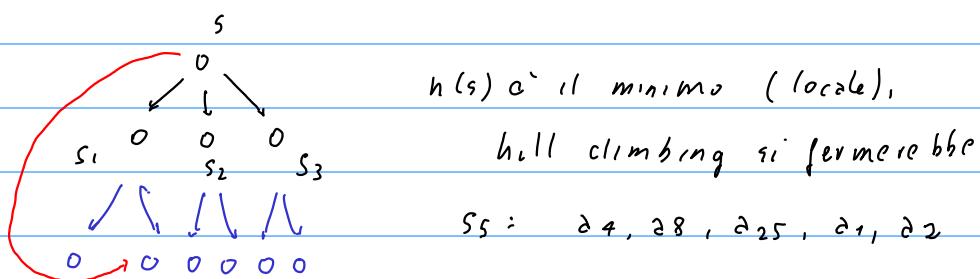
HILL CLIMBING in cui scegliere un successore anche quando peggiora,
ma non prendo gli stati negli ultimi K passi. (stati TABU),

Permette di uscire da massimi e minimi locali.

Se i minimi locali sono molti ma poco profondi taboo list funziona bene

ENFORCED HILL CLIMBING (innesta una ricerca in aspettativa)

VS (stato), se non migliora con una mossa estendi il vicinato a uno o due livelli, e' molto utile nelle creste. Faccio un macromosso.



Problema: memoria!

ALGORITMI GENETICI

stato → stringa (es N regine: sequenze di numeri)

FITNESS FUNCTION → dà una valutazione.

mescolanza tra stati

CROSSOVER

finora le ricerche erano offline

RICERCA ONLINE

alternanza ragionamento - computazione.

conoscenza parziale del mondo, apprendimento tramite esperienza,
memoria, ambiente dinamico

l'agente sa di essere in uno stato o un insieme di possibili
stati e sa quali azioni può eseguire, sa il costo dell'azione e
sa applicare il gol test.

l'agente non sa l'effetto della sua azione,

non sa gli stati successori, sa solo dove può muoversi
(es labirinto, MICEW)

misura delle prestazioni (online)

costo sol online
costo sol offline

può essere infinito

SPAZIO SICURO: → NO DEAD-ENDS

→ Tutte le azioni sono reversibili

Algoritmi

AGENTE — AZIONE, OSSERVAZIONE, AGGIORNAMENTO

Quale algoritmo ben si presta?

Una volta eseguita un'azione non posso tornare indietro
col teletrasporto.

A* fa teletrasporti sui nodi della frontiera.

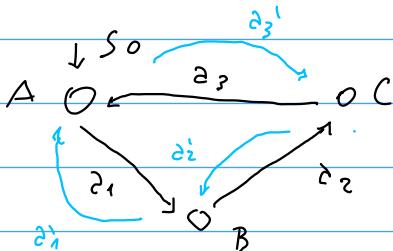
Ampezzo procede per livelli e similmente richiede dei salti

Profondità: compatibile → costruzione online di

↳ Albero di ricerca

↳ Diagramma degli stati

ESERCIZIO : ONLINE DFS ALGORITHM



$$s^1 = \alpha$$

$$s = \text{null}$$

(1) $\text{unex}[A] = a_1$
 $a = a_1$
 $\text{unex}[A] = []$

(2) $s^1 = B \quad s = A \quad \text{unex}[B] = a_2$
 $\text{unback}[B] = A \quad a = a_2$
 $\text{unex}[B] = []$

(3) $s^1 = C \quad s = A$

$$\text{unex}[C] = a_3$$

$$\text{unback}[C] = B$$

$$a = a_3$$

$$\text{unex}[C] = []$$

(4) $s^1 = A \quad s = C$
 $\text{unex}[A] = []$
 $\text{unback}[A] = [C]$
 $a = a_3'$
 $\text{unback}[A] = []$

(5)

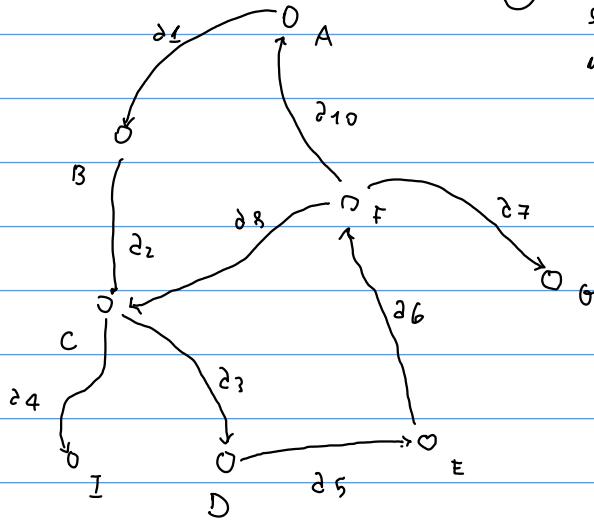
RANDOM WALK

(ricerca locale, è facile renderla online (es hill-climbing)).

Può essere un sottoalgoritmo pur che una prima scelta.

RICERCA LOCALE ONLINE CON APPRENDIMENTO DI $h(n)$

ESERCIZIO : ONLINE DFS ALGORITHM



(1)

$$S' = A$$

$$S = \text{null}$$

$$\text{unex}[A] = [d_1, d_3, d_{10}]$$

$$a = d_1$$

$$\text{unex}[A] = [d_3, d_{10}]$$

$$S \leftarrow A$$

(2)

$$S' = B$$

$$S = A$$

$$\text{unex}[B] = [d_2] \quad \text{unback}[B] = [A]$$

$$a = d_2$$

$$\text{unex}[B] = []$$

$$S = B$$

(3)

$$S' = C$$

$$S = B$$

$$\text{unex}[C] = [d_3, d_4]$$

$$\text{unback}[C] = [B]$$

$$a = d_3$$

$$\text{unex}[C] = [d_4]$$

$$S = C$$

(4)

$$S' = I$$

$$S = C$$

$$\text{unex}[I] = []$$

$$\text{unback}[I] = C$$

$$a = \overline{d_4}$$

AI 2020-10-05 1:01:32 / 2:35:02

COSTRAINT SATISFACTION PROBLEMS

→ BACKTRACKING SEARCH for CSPs (backtracking offline)
soluzione ottenuta senza far l'enumerazione
es. N regine

Gli stati sono assegnamenti di valori a variabili.

Vincoli → Binari → coppie di valori

↳ rappresentazione estensionale

↳ combinazioni di variabili e loro valori.

$$D(x_i) = \{1, \dots, 8\} \quad \text{PROBLEMA DELLE N REGINE}$$
$$i = 1, \dots, 8 \quad x_1, x_2, x_3, \dots, x_8$$
$$x_i \neq x_j \quad \forall i, j$$

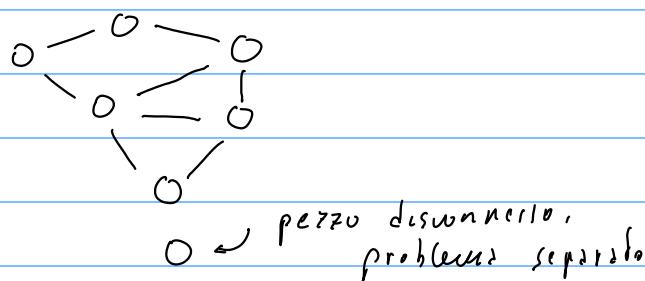
Vincolo in forma estensionale:

$$(x_1, x_2) : \left\{ \begin{array}{l} (1, 3), (1, 4), \dots \\ (2, 4), \dots \end{array} \right\}$$

molto esteso ma possibile

PROBLEMA DEL MAP-COLOURING

Colori limitati $D_i = \{\text{red, green, blue}\}$



Ipergrafi (vincoli più complessi).

CSP → variabili \rightarrow valori discreti \rightarrow domini \rightarrow finiti $\rightarrow d^n \rightarrow NP$
 \rightarrow valori continui \rightarrow spesso più infiniti
 \rightarrow valori misti \rightarrow semplici

\rightarrow VINCOLI \rightarrow variabili
 \rightarrow bindiri
 \rightarrow ordine superiore

es: Cryptaritmetica

Vincolo "ALCIDI-F", e' a 6 variabili

Search formulation:

initial state \rightarrow empty assignment {}

successor function \rightarrow assign value, no conflict

ricerca in profondità: depth n, n variables,
d = cardinalità del dominio,

si assegna un valore a una variabile

reduce: n - d possibilità

$$\text{depth 1: } (n-1) \cdot d \quad \left. \begin{array}{l} \vdots \\ \text{depth } n: 1 \cdot d \end{array} \right\} \boxed{n! \cdot d^n}$$

Invece di DFS \rightarrow DFS profondità limitata

MA: siccome non conosce l'ordine dell'assegnamento delle variabili; per ogni passo di ricerca si fissa l'ordine di assegnamento delle variabili:

1 \rightarrow si assegna la prima variabile

2 \rightarrow ...

:

Scompare l' $n!$, d^n foglie

Scelgo il backtracking come alg di ricerca

BACKTRACKING SEARCH

return RECURSIVE BACKTRACKING ({}, CSP)

RECURSIVE BACKTRACKING (assignment, CSP)

IF Assignment COMPLETE :

return assignment

vars ← select unassigned variables

FOR value in Domain(vars) :

IF value CONSISTENT with assignment, constraints :

ADD { var : value } to Assignment

result ← RECURSIVE BACKTRACKING (assignment, CSP)

IF result ≠ FAILURE :

return result

REMOVE { var : value } from assignment

return FAILURE

Il fallimento avviene quando la sol non esiste proprio

no libe L m ricorsione

si sceglie una variabile da assegnare:

ordinamento → statico *

→ dinamico **

per ciascun valore anche i domini delle var possono

essere ordinati → statico

→ dinamico

Sempre l'ordine delle variabili è molto di aiuto

IMPROVING BACKTRACKING EFFICIENCY

→ which variable next?

→ which values?

→ is it going to fail? (pruning)

} senza $h(s)$

TECNICHE :

Metodi: un solo buonio può essere compilato in una combinazione di buoni.

→ MOST CONSTRAINED VARIABLE

Var con meno valori disponibili per via dei vincoli precari, potrebbe essere una var in posz a fallimento quindi meglio saperlo prima, e colorazione: regole di dominio. (MINIMUM REMAINING VALUES)

→ MOST CONSTRAINED VARIABLE

Per gestire i pareggi eventuali ottenuti dopo la prima strategia:

degree heuristic: guarda le variabili future

sono heuristiche non legate a un problema ma generali, per qualsiasi CSP, domini indipendenti

H → specifico
H → generale
H → calcolato
sul dominio

LEAST CONSTRAINED VARIABLE

valori meno vincolanti, che lascia più libertà (\Rightarrow meno backtracking)

Queste 3 heuristiche rendono possibile il problema delle N regine con

$$N = 1000$$

+ ALTRE DUE TECNICHE che amalgano ancora di più.

FORWARD CHECKING

Vogliono spingere di più sul ragionamento.

"Prima di fare questa scelta... ma poi cosa posso fare?"

Teniamo traccia delle possibili scelte di valori per le future variabili, mi posso accorgere lo passi prima che una scelta a destra mi bloccerà poi

→ GENERALIZZAZIONE: CONSTRAINT PROPAGATION (INFERENCE)

controlli a rilento

X vincolo Y che vincolo Z che vincolo W ...

vengono eliminati gli assegnamenti impossibili

CONSTRAINT PROPAGATION \rightarrow ARC CONSISTENCY

ARC CONSISTENCY

COSTO POL. ED E' EFFICACE:

guarda tutte le coppie (altri guardano le triple).

$\forall x \text{ of } X \quad \exists y \text{ of } Y \text{ s.t. } (x,y) \text{ satisfy constraints}$

Quando x perde un valore, i vicini di x vanno ricontrollati a loro volta

Può dare un fallimento molto preciso,

può essere usato come preprocessing oppure come inferenza
(costo polinomiale quadratico)

ALGORITMO DI BACKTRACKING CON INFERNZA:

ARC-CONSISTENZA

Arc-consistenza:

(gli archi sono i vincoli)

...  ...

Ordinamento

QUEUE: inizialmente

univoci

tutti i vicini

quadrati

(caso peggiore, tutte le coppie)

(\hookrightarrow stessa coppia non può essere

in entrambe le direzioni)

reincerito un numero di volte

pari al dominio di X_i

(a terminazione c'è garanzia del fatto che i domini delle variabili sono discreti e finiti.

$n^2 d^3$

$$d = \max_{i=1, \dots, n} |X_i|$$

$n = \# \text{ variabili.}$

$n^2 - n = \# \text{ tutte le coppie di variabili.}$

ma la stessa coppia può entrare al massimo d volte \Rightarrow
 $\max_{d \times n}: (n^2 - n)d$ elementi estratti dalla coda.

M> I complimenti di REMOVE-INCORRECT-VALUES è $d^2 \Rightarrow$
 $(n^2 - n) d^3$

BACKTRACKING CON ARC CONSISTENCY (INFERENCE)

AC-3 → ARC CONSISTENCY (generalizzato)

RICERCA LOCALE APPLICATA AI CSP

Primo si disegna una sol e poi si cambiano i valori min, mettendo il num. di vincoli violati, è un hill climbing con funzione obiettivo da minimizzare (il numero di vincoli violati).

N-consistency (NAM consistency)

(è 3-consistency) viene usata nei problemi contenuti (dove l'arc consistency non è utilizzabile).

Per coppie deve esistere $\exists \rightarrow Z : C_{xz} \circ C_{yz}$ siano verificati

In mancanza delle 3 consistenze si può dimostrare che un CSP non è risolvibile (es. colorazione della mappa con solo due colori).

ES: 3 consistenze:

$$\begin{array}{l} x \leq y \quad y < z \quad x \leq z \\ \text{Dom}(x) = \{1, 2\} \\ \text{Dom}(y) = \{1, 3\} \\ \text{Dom}(z) = \{2, 4\} \\ C_{xz} = C_{xz} - \{(2, 2)\} \end{array}$$

N-consistency \Rightarrow ∃ algoritmo polinomiale risolutivo,

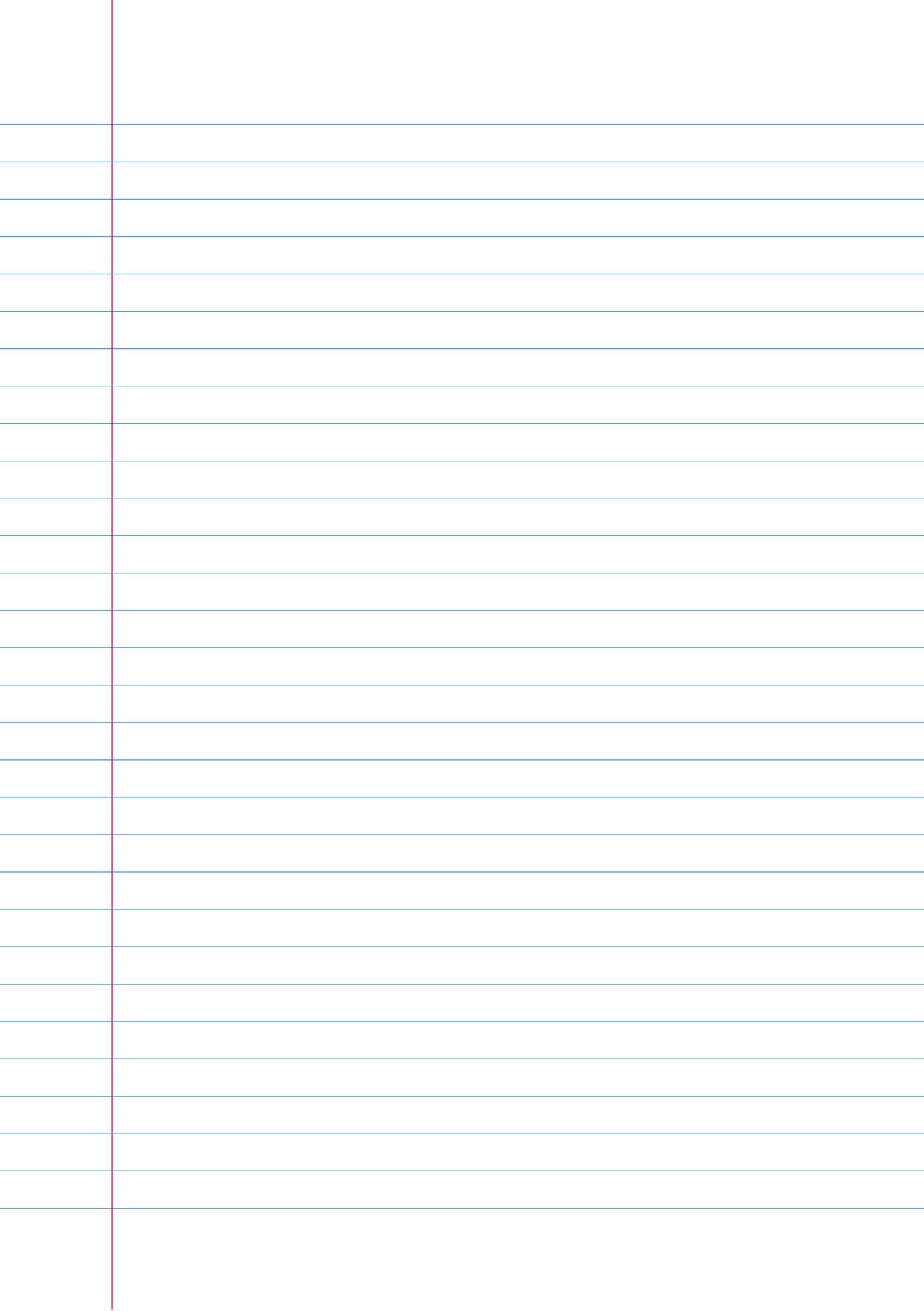
non ho bisogno di usare il backtracking.

(in pratica si usa molto poco, è costoso).

$$x_1 : v_1 \quad x_2 : v_2 \quad x_3 : \quad x_4 :$$



Scelto il valore di una variabile, non viene mai riconiderato.



SEARCHING

Problem → state space → graph
Initial state
goal states
actions applicable in s
transition model ($\text{result}(s, a) \Rightarrow s'$)
action cost function
 $\text{ACTION-COST}(s, a, s')$

SEARCH ALGORITHM

Search tree is superimposed over the state space graph

TREE GRAPH
Node ↔ State Set of states in the world
It can Edge ↔ Action (possibly infinite)
have many and actions
nodes for the same state

BEST FIRST SEARCH

choose n with minimum value of some evaluation function

Tree → Node → STATE
→ PARENT
→ ACTION
→ PATH-COST

Queue ← to store the frontier

Priority queue
FIFO queue
LIFO queue

Graph search → checks for redundant paths ✓

Tree search → ✗

Best first search is a Graph search

removing REACHED if becomes a tree search

Completeness → Finds a solution when there is

Cost optimality → Finds the solution with the lowest path cost

Time complexity

Space complexity → Computer science → $|V|, |E|$
Artificial intelligence → $d(\text{depth}, \text{actions})$
optimal
→ $m(\text{maximum actions})$
→ $b(\text{branching factor}, \text{successors})$

Search strategies ↗ Uninformed
Informed

UNINFORMED

↳ Breadth first search \equiv Best first search with evaluation function:

Optimization:

Best first \rightarrow late goal test

$$f(n) = \text{depth}[n]$$

(num actions to reach n)

Breadth first \rightarrow early goal test \rightarrow COST OPTIMAL but ONLY for problems where ALL ACTIONS have the SAME cost

nodes generated:

$$1 + b + b^2 + \dots + b^d = O(b^d) \leftarrow \text{Space complexity}$$

\nwarrow Time complexity

↳ Dijkstra's Algorithm / Uniform-cost search

Actions have different costs

\equiv Best first with evaluation function

$$f(n) = \text{PATH COST}(n)$$

Goal test is not when generating the node but when expanding lowest cost node

C^* cost of optimal solution

$$\varepsilon \text{ minimum cost of action } O\left(b^{1+\left(\frac{C^*}{\varepsilon}\right)}\right)$$

If all action costs are equal: $O(b^{d+1}) \sim$ Breadth first

↳ Depth first search

\equiv Best first search where the evaluation

function is $f(n) = -d$ but it is implemented

as a tree search, NO table of reached states \times

\times NOT cost optimal

\times can get stuck in a loop if the state space is cyclic.

Can get stuck going an infinite path if state space is infinite

✓ Much smaller needs for memory

REACHED TABLE is not used and the frontier is small.

Time: $O(\# \text{states})$

Space: $O(bm)$ $m \rightarrow$ maximum depth of the tree

~ Variant: backtracking search, only one successor at the time : even less memory usage

↳ Depth limited and iterative deepening search

DFS with limit ℓ on depth

Time: $O(b^\ell)$

Space: $O(b \cdot \ell)$

DFS

→ small memory

$O(bd) / O(b \cdot m)$

BFS

Optimal

when all costs
are the SAME

Complete on finite acyclic
state spaces

(even cyclic if it
checks nodes)

ITERATIVE
DEEPENING SEARCH

Time complexity: $O(b^d) / O(b^m)$

↳ Bidirectional search

simultaneously forward from initial state and
backwards from goal state,

$$b^{d/2} + b^{d/2} \ll b^d$$

INFORMED SEARCH STRATEGIES

↳ GREEDY BEST-FIRST SEARCH

$$f(n) = h(n)$$

Greedy best first graph search is complete only in finite spaces

Time, space complexity: $O(|V|)$, $\sim O(b^m)$ with a good heuristic.

↳ A* best first $f(n) = g(n) + h(n)$

$g(n)$: path cost from initial state to n

A^* is complete, it might also be cost-optimal, it depends on the heuristic

Admissibility: never overestimates the cost to reach the goal

If $h(n)$ is admissible $\Rightarrow A^*$ is cost-optimal

PROOF: suppose the optimal path has cost C^* but a path with cost $C > C^*$ is returned:

\exists n optimal path and n is not expanded

(if it was we would have returned C^* instead of C),

$g^* \leftarrow$ cost of optimal path from the start to n

$h^* \leftarrow$ cost of optimal path from n to nearest goal.

$f(n) > C^*$ (otherwise n would have been expanded)

$$f(n) = g(n) + h(n)$$

$$f(n) = g^*(n) + h(n) \quad (\text{because } n \text{ is an optimal path})$$

$$f(n) \leq g^*(n) + h^*(n) \quad (\text{because } h \text{ is admissible})$$

$$h(n) \leq h^*(n)$$

$$f(n) \leq \underbrace{C^*}_{\text{Contradiction}}$$

□

Contradiction

CONSISTENCY OF h :

$h(n)$ is consistent if for every node n and every successor n' of n generated by an action a , we have

$$h(n) \leq c(n, a, n') + h(n')$$

CONSISTENT \Rightarrow ADMISSIBLE

ADMISSIBLE $\not\Rightarrow$ CONSISTENT

CONSISTENT \Rightarrow A* optimal + the first time we reach a state it will be on the optimal path, no need to update REACHED entries.

A* prunes search tree nodes unnecessary for finding the optimal solution.

A*
→ complete
→ cost optimal
→ optimally efficient

ITERATIVE DEEPENING A* SEARCH

cutoff f-cost ($g + h$)

If the optimal solution has cost C^* there won't be more than C^* iterations



RELAXED PROBLEMS:

A relaxed problem is a problem with fewer restrictions on the actions:

the state space graph is a supergraph of the original state space,
less constraints \Rightarrow more edges.

OPTIMAL SOLUTION of the original problem \Rightarrow
still optimal solution for the relaxed problem,
no other way around \Rightarrow

the cost of an optimal solution to the relaxed problem
is an admissible heuristic for the original problem.
+ it's also consistent.

If a collection of admissible heuristics is available,

We can define $h(n) = \max_{i=1\dots K} \{ h_i(n) \}$

PATTERN DATABASES



cost(optimal solution(subproblem)) < cost(optimal solution(original problem))

Store exact solution costs for all subproblems \rightarrow
compute an admissible heuristic for each state encountered

during a search looking for the corresponding subproblem configuration in the database.

DISJOINT PATTERN DATABASES

SEARCH IN COMPLEX ENVIRONMENTS

LOCAL SEARCH AND OPTIMIZATION PROBLEMS

Local search → No track of paths or states

→ can find the best state according to an objective function (optimization problem)

HILL CLIMBING - GRADIENT DESCENT

"Greedy local search"

Gets stuck because of

- Local maxima
- Ridges → sequence of local maxima
very difficult to navigate
- Plateaus → flat area of the state space

We can add SIDEWAYS MOVE (with a limit to # consecutive moves.)

STOCHASTIC HILL CLIMBING

FIRST CHOICE HILL CLIMBING

RANDOM RESTART HILL CLIMBING

SIMULATED ANNEALING

Accepts ≥ bad move with a decreasing exponential probability.

The worse the move
The + longer the time → The less the probability
that the move will be accepted

LOCAL BEAM SEARCH

Keeps track of k states rather than just one

Initially k states are random

Generates all successors of current k states

chooses the k best successors

≠ running K restarts: because the

local beam search allows communication
among the parallel search threads.

the k states can get clustered in a small region of the state space

⇒ STOCHASTIC BEAM SEARCH

successors chosen with probability proportional to the successor's value

EVOLUTIONARY ALGORITHMS (recombination and selection)

ONLINE SEARCH AGENTS AND UNKNOWN ENVIRONMENTS

An online agent first takes an action, then observes the environment and finally computes the next action

USEFUL FOR:

- dynamic or semidynamic environments
- nondeterministic domains

ONLINE SEARCH PROBLEMS

(deterministic and fully observable environment)

The agent knows:

- ACTIONS(s): the legal actions in state s ✓
 - $c(s, a, s')$: cost of applying action a in [After the action is performed] state s to reach state s' ✓
 - IS-GOAL(s): the goal test ✓
 - MIGHT: Heuristic function
- NO: RESULT(s, a) ✗
ahead of time

Adversary argument

Dead ends cannot be foreseen.

ONLINE SEARCH AGENTS

Offline algorithms → $\xrightarrow{\text{explore}}$ model of the state space

Online algorithms → $\xrightarrow{\text{explore}}$ real world

An online algorithm can discover successors only for a state that it physically occupies.

Better to expand in local order

⇒ Depth first search is the most local
(apart from when it backtracks)

RESULT[s, a]: table showing the state resulting from action a in state s .

if the current state has unexplored actions, the agent tries one of those actions

if the agent has tried all the actions for a state:

→ BACKTRACK: going back to the state

from which the agent most recently entered the current state

UNBACKTRACKED → table of previous states not yet backtracked

WORST CASE: every link in the state space is travelled exactly twice

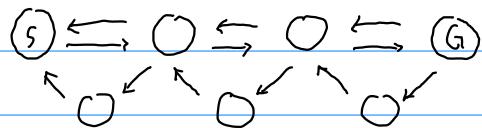
Optimal for exploration.

ONLINE LOCAL SEARCH

Hill climbing is already online, it keeps only one state in memory, it's local in its expansion, but it gets stuck at local maxima.
Random restarts cannot be used because teleport is not admitted in online search.

⇒ RANDOM WALK: randomly selects one of the available actions from the current state.

If space is finite and safely exploratory:
the algorithm terminates
but it can be very slow,
there can be traps



Instead of augmenting hill climbing with stochasticity, we can add memory, storing a current best estimate, $H(s)$, of the cost to get to the goal from s (each state that has been visited)

Initially: $H(s) \leftarrow h(s)$

then it gets updated as the agent gains experience in the state space

LEARNING REALTIME A*

- builds a map of the environment in the result table
- updates the cost estimate for the state it has just left
- chooses the apparently best move according to the cost estimates

Not yet tried actions in a state s are assumed to lead to the goal with the least possible cost ($h(s)$)

A*	LRTA*	
✓	✓	finds the goal in any finite safely explorable environment
✓	✗	Complete for infinite spaces
	$O(n^2)$	exploration in n states environment

OSA¹ learns a map of the environment (the outcome of each action in each state)

OSA acquires more accurate estimates of the cost of each state by using local updating rules

1: OSA = online search agent

A1

ESERCIZIO : RICERCA
IN PROFONDITÀ ONLINE

Diagrammi

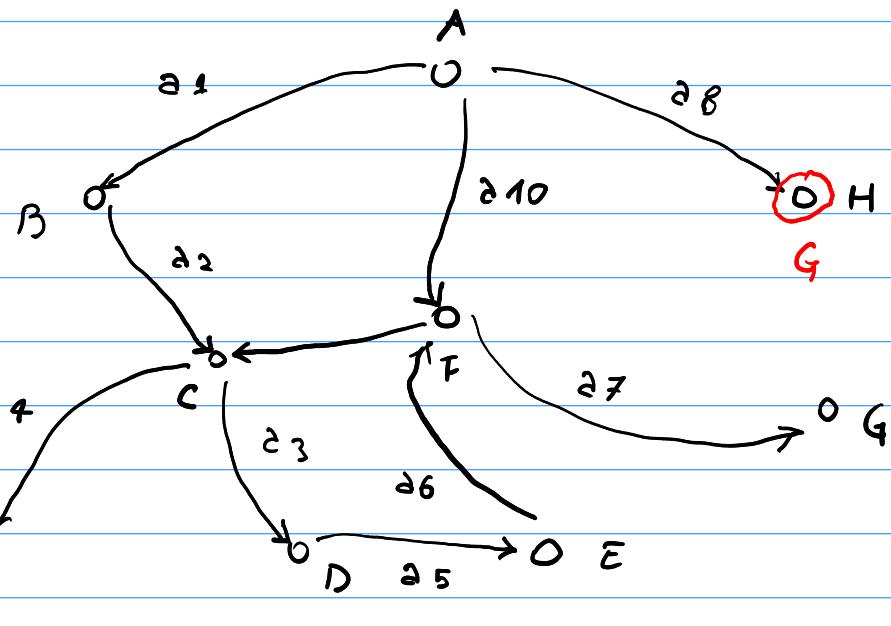
degli
stati.

Non ci
sono le
azioni
di reverse
indicate

ma si

indicono

con $\delta_1, \delta_2 \dots$



Regole di navigazione :

$$B \rightarrow H \rightarrow F$$

$$D \rightarrow I$$

$$G \rightarrow C$$

(1) $s' = A, s = \text{null} \quad \text{unexplored}[A] = [\delta_1, \delta_9, \delta_{10}]$

$$\delta = \delta_1 \quad j = A$$

$j = \text{stato}$
cerchio

(2) $s' = B \quad s = A \quad \text{unexp}[B] = [\delta_2]$

$$\delta = \delta_2 \quad j = B \quad \text{result}[\delta_1, A] = B$$

$$\text{unback}[B] = [A]$$

(3) $s' = C \quad s = B$

$$\text{unexp}[C] = [\delta_3, \delta_4]$$

$$\text{result}[\delta_2, B] = C$$

$$\text{unback}[C] = [B]$$

$$\delta = C$$

Dopo si chiede anche qual c' è il rapporto di competitività ottenuto dall'algoritmo...

④ $S' = D$ $S = C$

...
...

$\alpha_1, \alpha_2, \alpha_3, \alpha_5, \alpha_6, \alpha_7 \dots$

$\alpha_7 \rightarrow$ backtrack, F non nuovo: α_8 non ancora eseguita, C , $\alpha_4 \rightarrow I$, $\bar{\alpha}_4 \rightarrow C$, C non ha più niente da fare

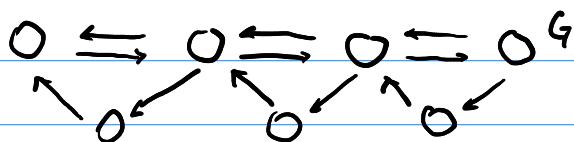
ma unbacktracked $[C] = [B, F]$
sceglie la più recente

↑
HEAD

Ricerca locale resa online:

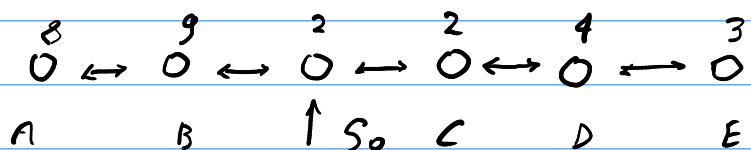
RANDOM WALK

può essere anche molto negativa,
ci sono stati che possono far tornare
indietro.



La possibilità di tornare indietro è doppia.

RICERCA LOCALE ONLINE CON APPRENDIMENTO (la locale non ha memoria)



Rispetto a una normale locale c'è qualcosa in
più: "l'euristica può essere migliorata?"

Aggiorno l'euristica di S₀ con 3 ← 2
Torno in S₀ e aggiorno il 2 di C con 4
Torno a C e aggiorno 3 con 5 in S₀,
poi preferisco D a S₀...

Ovvero sono uscito da un minimo locale,
e all'uscita dall'algoritmo ho imparato b,
alla prossima chiamata dell'algoritmo ho
un'euristica migliorata

LOCAL SEARCH REAL TIME A*

(! / \ non c'è best first, è locale)

di A* c'è solo h ammmissibile

dopo tantissime esecuzioni l'algoritmo troverà subito la soluzione ottima.

Viene usato per migliorare l'euristica, poi

si riusa l'euristica anche con altri algoritmi

LRTA* può essere fermato a piacere, anche senza che

restituisca il GOL, lo si può usare per 5 min

e poi usare A* o Iterative Deepening.

Utilizza una tabella, S, S' , result [], H (euristica modificata)

cost (s, s, s', H) {

h (euristica

aggiornata)

if s' undefined:

return $h(s)$

else :

return $c(s, s, s') + H(s')$

}

Le euristiche devono sempre essere ottimiste.

$H(s)$ viene aggiornato, quando sono in s'

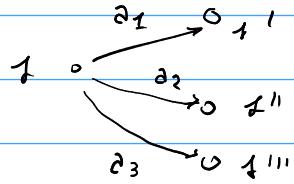
vengono aggiornate la mappa degli stati;

ESEMPIO

$$H[s] = \min_{\text{be Actions}[s]} LRTA^* - \text{cost}(s, a, \text{result}[b, a], H)$$

$$\text{result}[s, a_1] = s'$$

$$s = s'$$



ipotesi: s'' già visitato \Rightarrow

$H[s'']$ è definito

s''' non ancora visitato.

$H[s''']$ nullo

$$\text{Result}[a_2, s] = s''$$

$$\text{Result}[a_3, s] = \text{nullo}$$

Come viene aggiornato $H[s]$?

$$\text{Actions}(s) = [a_1, a_2, a_3]$$

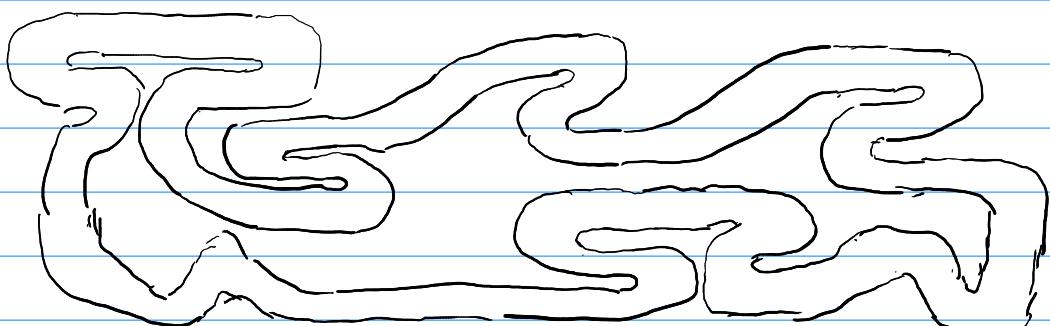
$$H[s] =$$

$$= \min \left\{ \begin{array}{l} LRTA^* - \text{cost}(s, a_1, s', H), \quad \text{result}[a_3, s] = \text{nullo} \\ LRTA^* - \text{cost}(s, a_2, s'', H) \end{array} \right\} \Rightarrow$$

$$= \min \left\{ \begin{array}{l} \text{cost}(s, a_1, s') + H[s'], \quad \text{si userà } h(s) \\ \text{cost}(s, a_2, s'') + H[s''], \\ h(s) \end{array} \right\}$$

= (conosciamo i costi delle azioni, non gli effetti).

affiorante dei run casuali si modifica H , H rimane ammissibile.



CONSTRAINT SATISFACTION PROBLEMS

$X \rightarrow \text{set of variables } \{x_1, \dots, x_n\}$

$D \rightarrow \text{set of domains } \{D_1, D_2, \dots, D_n\}$

$C \rightarrow \text{set of constraints}$

$D_i = \{v_1, \dots, v_k\} \text{ for } x_i \quad \text{es. boolean: } \{\text{True, False}\}$

$C_j = \langle \text{scope}, \text{rel} \rangle \rightarrow \text{values that the variables can take}$

$\downarrow \langle v_1, \dots, v_p \rangle \qquad \downarrow$

set of all tuples of values
that satisfy the constraint

$$\text{e.g. } X_1 = \{1, 2, 3\}$$

$$X_2 = \{1, 2, 3\}$$

$$C: x_1 > x_2 : \begin{cases} \rightarrow \langle (x_1, x_2), \{(3, 1) (3, 2), (2, 1)\} \rangle \\ \rightarrow \langle (x_1, x_2), x_1 > x_2 \rangle \end{cases}$$

CSP \rightarrow assignments of values to variables

$$\{x_i = v_i, x_j = v_j, \dots\}$$

Assignment

$\left\{ \begin{array}{l} \rightarrow \text{CONSISTENT} \rightarrow \text{doesn't violate any constraint} \\ \rightarrow \text{COMPLETE} \rightarrow \text{every variable is assigned a value} \\ \rightarrow \text{SOLUTION} \rightarrow \text{consistent \& complete} \end{array} \right.$

EG: map coloring

$$X = \{\text{WA, NT, Q, NSW, V, SA, T}\}$$

$$D_i = \{\text{red, green, blue}\}$$

$$C = \{ SA \neq WA, SA \neq NT, SA \neq Q, \dots \}$$

Constraint graph:

NODES → variables

EDGES → constraints

E.G.: Job-shop scheduling

CONSTRAINTS:

unary constraint → e.g. $\langle (SA), SA \neq \text{green} \rangle$

binary constraint → e.g. $SA \neq NSW$

[BINARY CSP → only unary and binary constraints

↳ representable as a constraint graph]

ternary constraint → e.g. Between(X, Y, Z)

$\langle (X, Y, Z), X < Y < Z \text{ or } X > Y > Z \rangle$

global constraint → arbitrary number of variables.

CONSTRAINT PROPAGATION - INFERENCE IN CSP

State space search algorithm

→ visits successors of a node

CSP algorithm

→ chooses variables assignments

→ constraint propagation

more constraints ⇒

⇒ less choices

CPS → constraint propagation → local search

LOCAL CONSISTENCY → NODE CONSISTENCY

↳ ARC CONSISTENCY

↳ PATH CONSISTENCY

↳ K-CONSISTENCY

NO DE CONSISTENCY

$x_i : \forall v \in D_i \quad v \text{ soddisfa il vincolo unario} \rightarrow x_i \text{ is node consistent}$

$\forall x \in G \quad x \text{ is node consistent} \rightarrow G \text{ is node consistent}$

ARC CONSISTENCY

x_i is arc consistent: $\forall v \in D_i \quad v \text{ soddisfa i vincoli binari su } x_i$

x_i è arc-consistente rispetto a x_j se

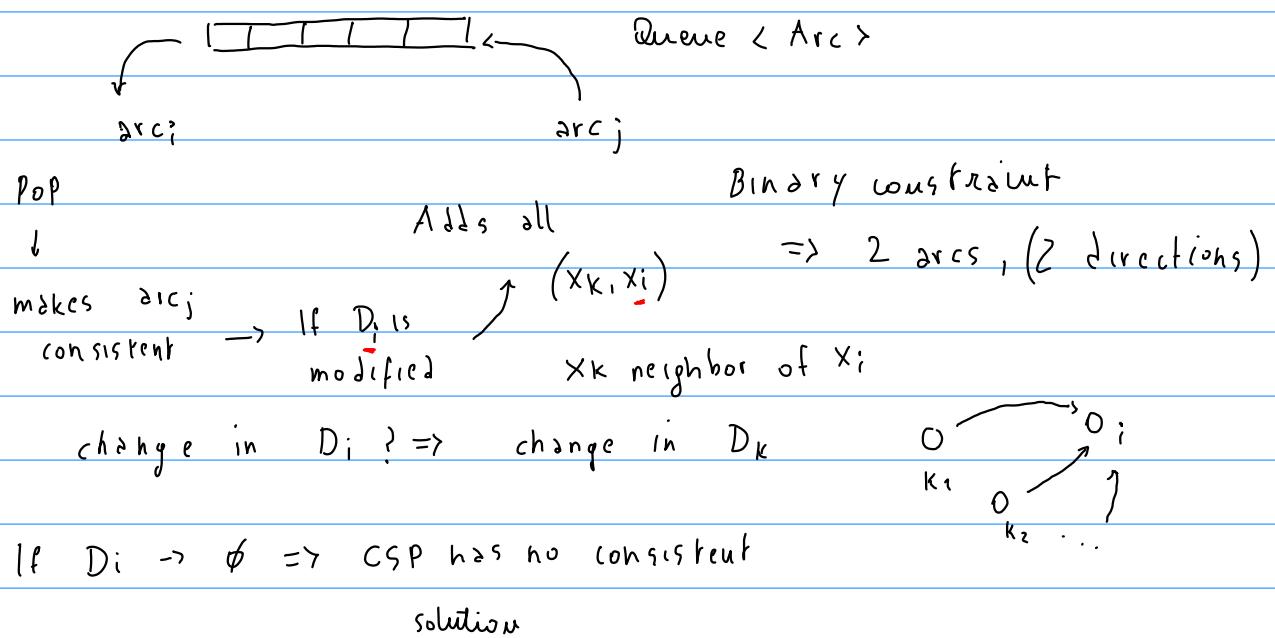
$\forall v \in D_i \quad \exists u \in D_j$ s.t.

il vincolo binario sull'arco (x_i, x_j)
è rispettato

G è arco consistente se $\forall x_i, x_j \quad i \neq j, \quad x_i$ è arco consistente
rispetto a x_j

AC-3 algorithm \rightarrow enforces arc consistency.

Makes every variable arc consistent



$CSP \rightarrow AC-3 \rightarrow CSP' \cong CSP$, same solution, but

CSP' is faster to search because its domain is smaller

AC-3 complexity:

$$n \text{ variables } x_1, \dots, x_n \quad D_1, \dots, D_n$$
$$\max_{i=1, \dots, n} |D_i| = d$$

$$\# \text{ binary constraints} = c$$

$\forall \text{ arc, arc} = (x_k, x_i)$, arc can be enqueued
at most d times

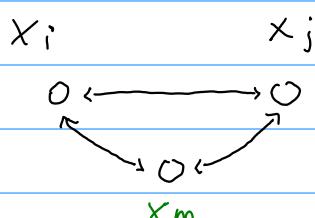
Check consistency of an arc $\rightarrow O(d^2)$

$$\Rightarrow O(c d^3) \text{ but } c \leq n^2 - n \sim n^2 \Rightarrow O(n^2 d^3)$$

PATH CONSISTENCY

Triples of variables

A two variables set $\{x_i, x_j\}$ is path consistent with respect to a third variable x_m if, for every assignment $\{x_i = a, x_j = b\}$ (consistent with the constraints on $\{x_i, x_j\}$) there is an assignment to x_m s.t. the constraints on $\{x_i, x_m\}$ and $\{x_m, x_j\}$ are satisfied.



BACKTRACKING SEARCH FOR CSP

Finished the constraint propagation we have to search...

STATE \rightarrow partial assignment

ACTION \rightarrow assignment extension

On variables, d possible values

$\begin{array}{c} / \quad \backslash \\ o \dots o \end{array}$ n $d \rightarrow$ branching factor

$$(n-1)d$$

$$(n-2)d$$

$$o \quad o \quad \dots \quad o \quad \text{LEAVES} \rightarrow n! d^n$$

BUT the possible assignments
are only d^n

Thanks to commutativity we get back the $n!$ factor.

LEAVES (no care for the order) : d^n

Elena

AI PASSO

Searching → unknown environment - local search
 → known environment

Search problem → state space ($\{s\}$)
 → initial state
 → $\{g\}$ goal states
 → $\{a\}$ actions → forms a path (solution when joining s_0 and G)
 → transition model ($\text{result}(s, a)$)
 → action-cost function $\text{ACTIONCOST}(s, a, s')$
 can be the optimal solution

Search algorithms:

✓ search tree REDUNDANCY!

state space graph

↳ remember all reached

states →

redundant ✓

cycles ✓

↳ specific problem formulation

make redundancy
NOT LIKELY

$$\text{Dijkstra: } d \leftarrow \left[\frac{C^*}{\epsilon} \right]$$

$O(b^{1+d})$

complete and cost-optimal

↳ chain of "parent" pointers:

BFS

complete and cost optimal

no additionally memory.

cycles ✓

redundant paths ✗

reached table

BFS

✓

UCS

✓

BestFS

✓

DFS

✗

DFS: non cost optimal

↳ Finite trees: complete

↳ Acyclic graphs: complete, with redundancy

DFS

:-)

no reached table
very small frontier

TIME: $O(b^d)$

SPACE: $O(b \cdot d)$

↳ Cyclic: stuck in infinite loops

↳ Infinite: INCOMPLETE
(whereas BFS is complete)

DFS → generates all successors (b^d)
 and chooses one to go deeper through
 $m = \text{actions}$

BACKTRACKING → generates one successor → $O(b^m)$ $O(m)$
 check for cycles in $O(1)$ time instead of $O(m)$
 ↳ efficient data structure.

Depth limited search
 Iterative deepening $\leftarrow O(b^d), O(b^m)$ DFS
 Optimal when costs
 are the same and
 complete on finite
 acyclic spaces
 Complete on finite
 spaces when we
 check for
 cycles going
 up to the root.

Greedy Best first search (heuristic $h(n)$)

$f(n) = h(n)$ finite space → complete
 infinite space → incomplete

$O(|V|)$ → can be improved → $O(b^m)$

A* (Best first search with $f(n) = g(n) + h(n)$)

↪ complete ✓

↪ cost optimal ○ → h admissible ✓ → h consistent → never re-add ✓

PROOF: suppose h admissible and
 A^* non cost-optimal:

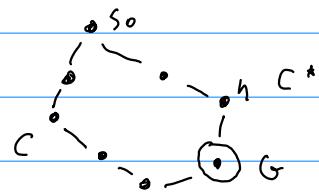
then $\exists C^*$ cost of optimal solution
 but a path with cost $C > C^*$ is
 returned ⇒

$\exists n$ on the optimal path that is
 not expanded

and n is s.t. $f(n) > C^*$

(otherwise n would have been
 expanded)

$$\begin{aligned} \text{but } f(n) &= g(n) + h(n) = \\ &= g^*(n) + h(n) \leq \\ &\leq g^*(n) + h^*(n) = C^* \end{aligned}$$



h is consistent if: $\forall n, n' \quad (n' \text{ successor of } n)$ generated with action $a: \quad h(n) \leq c(n, a, n') + h(n')$

ITERATIVE DEEPENING A*

doesn't keep in memory all reached states \Rightarrow
 \Rightarrow can visit some states more than once.

The cutoff is $f(g+h)$ at each iteration

smallest f cost of any node that exceeds the previous cutoff.

Generating heuristics: \rightarrow from relaxed problems

$H \leftarrow$ cost of optimal solution to a relaxed problem

\rightarrow it's admissible
and consistent

\hookrightarrow From subproblems

$H \leftarrow$ solution cost of a subproblem

\rightarrow admissible

$$c(\text{optimal s(sub)}) \leq c(\text{opts(orgp)})$$

PATTERN DATA BASES

the database is constructed \rightarrow stored exact solution costs (comprehensive of actions on non-interesting elements) \leftarrow \forall possible subproblem instance

starting from the goal state going backwards \rightarrow When solving an original problem instance:

$h = \text{database [configuration]} \rightarrow \text{value}$
(cost of solution of subproblem)

DISJOINT PATTERN DATABASES:

create more subproblems, for each of them construct a database and apply pattern database BUT instead of saving the cost of the solution comprehensive of all the moves

(also regarding non interesting elements)

save the cost of the moves on the

interested elements only (smaller cost)

\Rightarrow different subproblems costs can be added,

heuristics can be added and preserve admissibility.

LOCAL SEARCH

no track of paths or reached states \Rightarrow NOT SYSTEMATIC

HILL CLIMBING (local maximum) \sim GRADIENT DESCENT

\hookrightarrow called "greedy local search"

SIMULATED ANNEALING (HILL CLIMBING + RANDOM WALK)

$$T: \infty \rightarrow 0 \quad t: 0 \rightarrow \infty$$

$\Delta E \rightarrow$ worsening

$\Delta E > 0 \Rightarrow$ move accepted

LOCAL BEAM SEARCH:

k states, NOT like random restarts,

RANDOM RESTART \rightarrow independent searches

LOCAL BEAM \rightarrow shared information between
threads

best k successors

among all
successors of

current k states.

unfruitful searches are

abandoned

k states can become too clustered

\Rightarrow STOCHASTIC BEAM SEARCH

chooses successors with probability

proportional to the successor's
value

ONLINE AGENTS

action \rightarrow observation \rightarrow computation
(dynamic and semidynamic environments)

Deterministic fully observable environment

The agent only knows:

- \rightarrow actions in current s
- \rightarrow cost $c(s, a, s')$ after doing a
- \rightarrow GOAL TEST
- \rightarrow might have $h(s)$

ONLINE DEPTH FIRST EXPLORATION AGENT (every action can be undone)

stores & map:
worst case:
traverses every link in the graph
spare exactly twice

result [s, a]	offline	online
unexplored [s]	no more successors	no more successors
unbacktracked [s]	↓ drop from queue	↓ BACKTRACK

IF s' is goal
STOP
IF s' not in untried:
 untried [s'] = actions (s')
IF $s \neq \text{NULL}$:
 result [s, a] = s'
IF untried [s'] is empty:
 IF unbacktracked [s'] is empty,
 return STOP
 ELSE:
 a = action such that result [s', a] = pop (unbacktracked [s'])
 $s' = \text{null}$
ELSE
 $a = \text{POP}(\text{untried}[s'])$
 $s = s'$
return a

k consistency:

$\forall \{vars\}_{k-1}, \forall$ assignment
consistent to vars

$\exists val$ s.t. k th var $\leftarrow val$
is consistent

es 3-consistent:

$\{x_i, x_j\}$ is path consistent with respect to x_m if

$\forall \{x_i = a, x_j = b\}$ arc consistent

$\exists c \rightarrow x_m : \{x_i, x_m\}, \{x_m, x_j\}$ consistent.

CSP strongly consistent (k) : k consistent, $(k-1)$ consistent,
 $(k-2)$ consistent, ..., 1 consistent.

(CSP, n nodes, strongly n -consistent \Rightarrow

BACKTRACKING ON CSP

$$n \cdot d \cdot (n-1) \cdot d \cdot (n-2) \cdot d \cdots 1 \cdot d = n! d^n$$

BUT \rightarrow commutativity

consider a
single variable
for each node.

$$\frac{n! d^n}{n!}$$

$O(n^2 \cdot d)$

Costo RM-
INCONSIST-
VALUES:

$$O(n^2 d^3) \leq$$

uninformed search algorithms \rightarrow improve \rightarrow domain dependent heuristic

CSP

\rightarrow improve \rightarrow domain INDEPENDENT

How to choose next

heuristic

undesignated variable (MRV) Minimum Remaining Values
(or MCV most constrained variable)

LOCAL SEARCH FOR CSP

è incompleto, non serve l'ottimo, è molto più veloce.
a volte è l'unica scelta.

Parte da un assegnamento e lo modifica (cambio le variabili
in conflitto).

HILL CLIMBING guarda la min-conflict.

Funziona bene quando il problema è deuso di soluzioni.

A⁺: Seminar

perchiamo subottimi.

→ Bounded Suboptimal Search: neighbors ← successors
 ↓ best.

L₇ BEG FOCAL SEARCH

(\rightarrow) h admissible

→ Open List

→ Final list → estratto da qui il
prossimo nodo migliore.

11 HgJ_0 J₀

$$f \leq B \cdot f_{\min}$$

espendere è preso da FOCAL

for n in neighbours($h[st]$)

n → OPEN

if $f(n) \leq B \cdot f_{\min}$
 $n \rightarrow \text{FOCAL}$

↳ Weighted A*

prima è nodi con $f_w(n)$ minima

$$f_w(n) = g(n) + Wh(n)$$

$$W = 1 \Rightarrow WA^* = A^*$$

$W \rightarrow \infty \Rightarrow WA^* = \text{Greedy Best first Search}$

WAT è un caso special di TOTAL, la sol trovata

\hat{p} is w -admissible.

→ Bounded Cost Search:

Ls Potential Search

C → può essere in input o può essere calcolato dinamicamente.

probabilità che i nodi siano parte di un
pino avere costo $\leq c$

$$\text{Potenziale : } u(n) = \frac{C - g(n)}{h(n)}$$

simplizieren nach

$m \in \text{scr}(M, \mathbb{F}_1) = \mu(n)$

Gestione dello stime (soft bound)

Abbiamo una stima del costo C della soluzione.

Se C è maggiore, trovare lo sul e' più facile

Obiettivo: velocizzare la ricerca concentrandosi sui nodi vicini a C .

$g(n)$, $h_{\text{search}}(n)$ non ammissibile
 $h_{\delta}(n)$ ammissibile
 C (stima)

Schemi: weighted A* :

$$f_w(n) = g(n) + w h_{\text{bound}}(n)$$

dove: $h_{\text{bound}}(n) = h_{\text{search}}(n) \cdot \left\{ \frac{C}{g(n) + h_{\delta}(n)} \right\}$

Fattore moltiplicativo: penalizza e premia i nodi:

Se $g(n) + h_{\delta}(n) \ll C \rightarrow n$ penalizzato

Se $g(n) + h_{\delta}(n) \geq C \rightarrow h_{\text{search}}$ viene rimpicciolita dal fattore moltiplicativo

Se $C \rightarrow C_{\text{opt}}$: seguimo knoppo A*

(ovvero facciamo knoppo fatica)



metria del penalty rate:

$$\text{prate} = \frac{\{\# \text{exp. nodes s.t. } g(n) + h_{\delta}(n) > C\}}{\#\text{expanded nodes}}$$

Più nodi superano C , più $1 - \text{prate} \rightarrow 0$ e

* quando molti nodi superano C
 (ovvero siamo lontani)

il fattore moltiplicativo

$$\left(\frac{C}{g(n) + h_{\delta}(n)} \right)^{1 - \text{prate}}$$

si avvicina a 1 e si fa retta

a $h_{\text{search}}(n)$, se non fosse per l'esponente

$1 - \text{prate}$ $h_{\text{search}}(n)$ verrebbe severamente penalizzata rallentando la ricerca ma così facendo h_{search} viene ascoltata *

Agenti artificiali intelligenti

Ambiente $\xrightarrow{\curvearrowleft}$ Agente razionalità \rightarrow sulla base della conoscenza acquisita.

$$f: P^* \rightarrow A$$

percezioni \rightarrow azioni.

Ambiente

\hookrightarrow Accessibile / Non accessibile

Osservabilità \rightarrow completa \rightarrow scacchi
 \rightarrow parziali \rightarrow strada
 \rightarrow nulla

\hookrightarrow Deterministico / Non deterministico / Strategico
effetti delle azioni noti,
effetti probabilistici.

\hookrightarrow Episodico / Non episodico

episodico \rightarrow robot classificatore

\hookrightarrow puoi farle, ogni azione c'è se stante

non episodico \rightarrow qualità dipende anche dalle azioni precedenti.

\hookrightarrow Statico / Dinamico / Semidinamico

il mondo non cambia mentre l'agente decide
(l'algoritmo può prendersi il tempo)

Dinamico \rightarrow taxi driver

Semidinamico \rightarrow sfida ma il tempo conta comunque.

\hookrightarrow Discreto / Continuo

percezioni \rightarrow basso

\rightarrow altissimo (ogni step continuo).

\hookrightarrow Singolo Agente / Multiagente

\hookrightarrow competitivo \rightarrow scacchi.

\hookrightarrow collaborativo. \rightarrow

camion e pacchi

Tipi di Agenti

- ~ Simple reflex agents → Database condition-action.
per ambienti episodici (if-then)
non sa quando ha finito.
- ~ Model based reflex agents → Database + ulteriore conoscenza che viene aggiornata, es, se dove si trova.
Si rende conto di quando ha finito. Riconosce salvo stato del mondo
- ~ Goal Based agent: ragiona sul futuro oltre i fare quello che faceva Model Based Agent.
possono avere goal diversi, quindi devono tenere conto dei goal
→ Utility Based agent → oltre ad avere il goal tengono conto di altre metriche.
ogni azione ha un suo costo,

Problemi con Contingenze

- ↑ Sequenza di azioni → Albero di azioni.
ci sono già i belief states (integrazione di azioni sensoriali che hanno effetti imprevedibili, sono etlumi non deterministico)
- Tecniche euristiche.

BFS FOCAL h ammissibile

local < frontier

BFS WEIGHTED A* h ammissibile

$$f_w \text{ minimo: } f_w(n) = g(n) + h(n) \cdot w$$

BFS Explicit Estimation Search

h_1 ammissibile \rightarrow garantisce la w -ammissibilità della soluzione.
 h_2 non ammissibile \rightarrow guarda la ricerca verso il goal

h_2 non ammissibile (basata sugli step, mentre le altre sui costi)

come se tutti i costi fossero 1

se il miglior nodo di h_2 è β ammissibile \rightarrow quello

altrimenti se il migliore di

h_2 è β ammissibile \rightarrow quello

altrimenti scegli quello che raggierebbe A^*_ϵ
(FOCAL)

Gestire stime (soft bound)

Come sfruttare un preventivo.

Le stime priori possono essere anche inaccurate, quindi vogliamo usarle solo come suggerimento.

$$f_w = g(n) + h_{\text{bound}}(n)$$

$$h_{\text{bound}} = h_{\text{search}} \cdot \frac{c}{g + h_0}$$

Percorso se il costo

$\underbrace{g + h_0}_{\text{lower bound}} \longrightarrow$ ammissibile

è troppo sotto stimato.

node troppo piccoli \rightarrow buttati

node che eccedono \rightarrow accelerati.

$\hookrightarrow p_{\text{rate}} = \frac{C \gg}{C}$
 $\hookrightarrow p_{\text{rate}} \approx 1 \Rightarrow$ non usare troppo la frazione!

$\hookrightarrow \dots \Rightarrow$ abbiam sofferto $C \Rightarrow$ usiamo la matrice.

PREPROCESSING, prima di lanciare l'algoritmo!