6/11/2023 | LEZ 8 |

ALGO & INFORMATA: MEMORIA

LEZ PRIMA → TIME OF A* E OTTIMALITÀ TEMPORALE

OGGI → ALGO INFO x ↑ SPACE OCCUPIED

ALGO & A MEMORIA LIMITATA

ALGO BEAN SEARCH

CARATT.: BREST-FIRST, MA LA FRONTIERA È GRANDE FC, KEEP I K NODI & 'PROMETTENTI'.

CODE

Algorithm 1: Beam Search Algorithm

Data: Graph (G), start node (s), goal node (g), beam width (β)
Result: Path with lowest cost
Function beamSearch(G, s, g, β)
  openList ← *s*
  closedList ← *empy list*
  path ← *empy list*
  while *open list is not empty* **do**
    b ← *best node from openList*
    openList.remove(b)
    closedList.add(b)
    if *b is g* **then**
      path.add(b)
      return *path*
    end
    N ← *neighbors(b)*
    for *n in N* **do**
      if *n is in neither closedList nor openList* **then**
        openList.add(n)
      else if *n is in openList* **then**
        if *path with current parent* ≤ *path with old parent* **then**
          Replace parents of n
        end
      else if *n is not in closedList* **then**
        openList.add(n)
      end
    end
    if *number of nodes in openList* > β **then**
      openList ← *best* β *nodes in openList*
    end
  end
  return *path*
end

RISULTATO

- + EFFICIENTE IN MEMORIA
- NON COMPLETO, MA SUBOTTIMALE
- NON BELLISSIMO, MA PUÒ DARE DEI BUONI CON MEMORY RISPARMIATA

IDA*: A* AD APPROFONDIMENTO ITERATIVO

CARATT.:

- ID MA SFRUTTA VANTAGGI DI A*
- IL "CONFINE" NON È α (COME IN ID) MA:

  $F = g + h \Rightarrow$ $f(n)$, n = NODO U + PICCOLO F "SCOPERTO" FUORI DAL CONFINE PRIMA

- A ITERAZIONE, SUI NODI U $F \leq F$ È "SCARTA" I NODI FUORI - CONFINE, FINITO SI RESTRINGE CONFINE CON $F = f$ + PICCOLO TRA NODI SCARTATI.

PRESTAZIONE

- COMPLETEZZA: SI, ASSUMENDO CHE:
  (CORRETTEZZA)
  - IF $\forall$COST(AZIONE) $= k$ E INCREMENTO $= K$
  - IF " " $=$ VARIABILE $\wedge$ INCR $\geq \varepsilon$
  - IF F = COME DEFINITO IN CARATT. (C. MINIMO AZIONE).

- COMPLESSITÀ SPAZIALE $= O(b \cdot d)$ → COME DF.

BEST-FIRST RICORSIVO

CARATTERISTICHE

- DFS RICORSIVA, MA COL F-LIMITE COME CONFINE:
- X NODO CHE VOGLIO ESPANDERE, TENGO CONTO DI F-LIMITE = F DI PERCORSO MIGLIORE ALTERNATIVO.

CODE

```
function Ricerca-Best-First-Ricorsiva(problema)
    returns soluzione oppure fallimento
    return RBFS(problema, CreaNodo(problema.Stato-iniziale), ∞)        // all'inizio f-limite è un valore molto grande

function RBFS (problema, nodo, f-limite)
    returns soluzione oppure fallimento e un nuovo limite all' f-costo      // restituisce due valori
    if problema.TestObiettivo(nodo.Stato) then return Soluzione(nodo)
    successori = [ ]
    for each azione in problema.Azioni(nodo.Stato) do
            aggiungi Nodo-Figlio(problema, nodo, azione) a successori     // genera i successori
    if successori è vuoto then return fallimento, ∞
    for each s in successori do        // valuta i successori
            s.f = max(s.g + s.h, nodo.f)
    loop do
            migliore = il nodo con f minimo tra i successori
            if migliore.f > f_limite then return fallimento, migliore.f
            alternativa = il secondo nodo con f minimo tra i successori
            risultato, migliore.f = RBFS(problema, migliore, min(f_limite, alternativa))
            if risultato ≠ fallimento then return risultato
```

- IF NODO RIC. HA F > F_LIM. RITORNO INDIETRO E RESPIDO ALTRI NODI, "FORGET" SUCC. E NODO SCARTATO.

] NODO F
] = NODO F ALTERNATIVO
→ NUOVA RICERCA CON
· F_LIM = MIN(F.ALT, F_LIMITE)
IF FAIL, NODO.F = F_MIGLIORE FIGLIO.

PRESTAZIONE

OTTIMO = YES IF h(n) AMMISSIBILE

MEMORY = LINEARE RISPETTO A MAX PROFONDITÀ DI CONT. OTTIMO

TIME = - DIFFICILE A DIRSI, PUÒ ESPLORARE + VOLTE
= CAMBINI (LAVORO INUTILE).

MA* ∧ SMA* ⇒ OTTIMO ×

- STATE SPACE = GRAFO
- COST (ACTION) ≠ UNIFORME
- GESTIR. LOOP = COSTO
  ↑ MANTENERE FRONTIERA

CARATTERISTICHE

- MA* = A* U MEMO. LIMITATA / SMA = MA* SEMPLICE
- SMA* FUNZIONA COME A*, PREGRESSO QUANDO MEMORIA PIENA
- IF ∃ (∅ ESPANSI NODO), ALLORA DELETE IN NODI ESPANSI, NODO U F+ ALTO.

E, IN FATTO MEMORIZZO F OF FIGLIO DIMENTICATO (BFRS), COSÌ SO QUANTO COSTA PASSARE DI LÌ, E RITORNARCI SE GLI ALTRI CAMMINI FANNO CAGARE.

- IF ∀ NODO HA F=, ESPANDO BEST YOUNG NODO ∧ CANC OLDEST WORST NODO.

PRESTAZIONE

- COMPLETO = YES, IF SOL. RAGGIUNGIBILE:
  IF d ↓ D. SOLUZ ≤ MEMORY
- CORRETTO = DIPENDE DA SOLUZIONE:
  - YES → ∃ PATH OTTIMO
  - NO → RETURN PATH GIUSTO + AMMISSIBILE
- SPACE? MEMORIA DISPONIBILE
- TIME? ⇝ HARD FARE VALUTAZIONE

↓

LOTTE PROBLEMI, O LUNGHI TEMPI. SOLO INTRATTABILI

STUDIO FUNZIONI EURISTICHE

utili queste euristiche, ma come possiamo sfruttarle bene? possiamo crearli delle buone euristiche? c'è differenza tra una euristica e un'altra?

RISPONDIAMO CON CALMA

- INNANZI TUTTO, CI SONO EURISTICHE MIGLIORI DI ALTRE, INFATTI:

SIANO h_1, h_2:

DEF!

h_2 È ALMENO INFORMATO (+ EFFIC.) RISPETTO A h_1 SE

$$h_1(n) \leq h_2(n) \Rightarrow \text{DOMINA IF } (\forall n)$$

F N.B.
UNA EURISTICA È AMMISSIBILE IF $\forall n$

$$0 \leq h(n) \leq h^*(n)$$

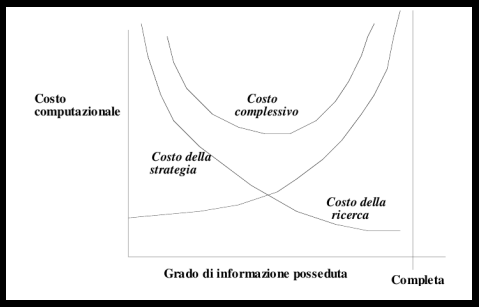
0: POCA INFORMAZIONE — $h^*(n)$: MAX INFO.

PRENDIAMO COME ESEMPIO A^*:

OGNI NODO ESPANSO DA A^* CON h_2 È UN SOTTOINSIEME DEI NODI ESPANSI CON h_1

↓

+ PRECISO QUINDI, MA PUÒ ESSERE + COMPLICATO DA CALCOLARE



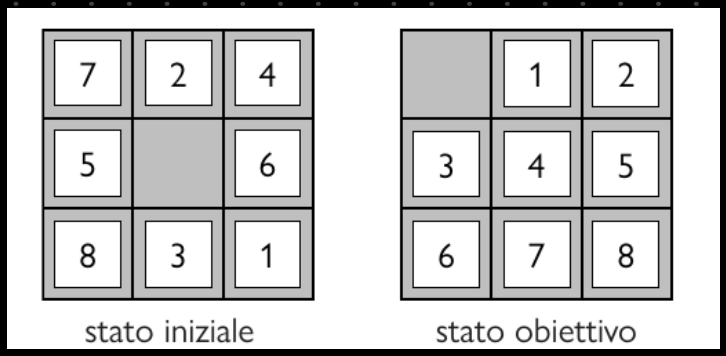
RAPPORTO EFFICIENZA/PRESTAZIONI

ESEMPIO PRATICO: 8 GAME

h_1 = # CASELLE FUORI POSTO

h_2 = SUM DIST. MANHATTAN OF CASELLE FUORI POSTO

↳ SUL # SPOSTAMENTI 1 a – x METTERE A POSTO LA CASELLA.



$h_1 = 8$ → TUTTI FUORI POSTO

≈ $h_2 = 3+1+2+2+3+2+2+3 = 18$

3 MOSSE × MATTONE A POSTO '7'

PRECISIONE ∧ EFFICIENZA

× CAPIRE ACCURATEZZA EURISTICA:

FATTORE DI RAMIFICAZIONE (b^*) ≈ N. DI FIGLI GENERATI AD OGNI ITERAZIONE

ESEMPIO CON A*:

TOT. NODI GENERATI = N

PROFONDITÀ d

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

NODI GENERATI

+ b^* VICINO A 1 → − NODI GENERATI → + P SOL. EFFICIENTE

MA EFFICIENTE ≠ POLINOMIALE.

INFATTI RIMANGONO COMUNQUE COSTI ESP, MA EFFETT. RIDUCONO IL COSTO DI P, CIOÈ LA PROFONDITÀ EFFETTIVA OF ALBERO DI P, DI UN FATTORE k_h

QUINDI U BUONA $h \rightarrow c(P) = O(b^{d-k^h})$

X CAPIRE

	Costo di ricerca (nodi generati)			Fattore di ramificazione effettivo		
d	RIA	$A^*(h_1)$	$A^*(h_2)$	RIA	$A^*(h_1)$	$A^*(h_2)$
6	128	24	19	2,01	1,42	1,34
8	368	48	31	1,91	1,40	1,30
10	1033	116	48	1,85	1,43	1,27
12	2672	279	84	1,80	1,45	1,28
14	6783	678	174	1,77	1,47	1,31
16	17270	1683	364	1,74	1,48	1,32
18	41558	4102	751	1,72	1,49	1,34
20	91493	9905	1318	1,69	1,50	1,34
22	175921	22955	2548	1,66	1,50	1,34
24	290082	53039	5733	1,62	1,50	1,36
26	395355	110372	10080	1,58	1,50	1,35
28	463234	202565	22055	1,53	1,49	1,36

(*) RIA = = P LOW INFORMATA

HOW TO CREATE EURISTICHE

CI SONO VARIE TECNICHE, TUTTE VALIDE:

1] PROBLEMI RILASSATI

CALCOLO h DI UN PROBLEMA "RILASSANDO", CIOÈ TOGLIENDO QUALCHE VINCOLO

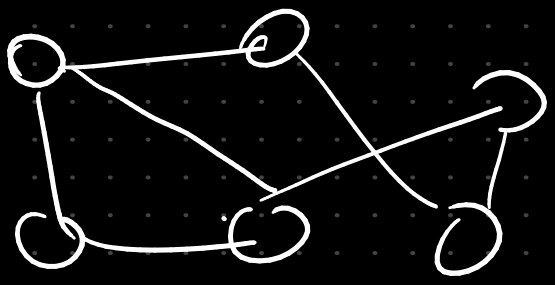
GAME_RILASSATO

→ (1) A CASELLA POSSO MUOVERLA IN QUALSIASI POSTO → GENERO h_2

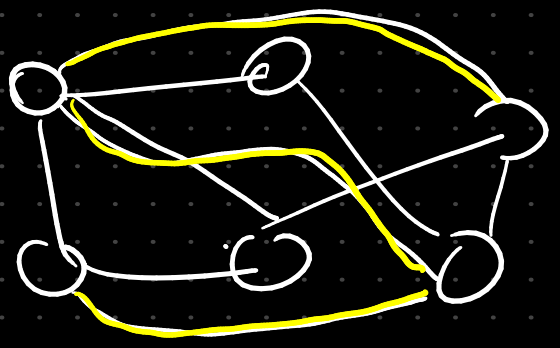
→ (2) A CASELLA SOPRA MUOVE → O ↑, A PRESCI= USCIRE SE LE CASELLE SONO OCC. ONO GENERO h_2

OGNI h GENERATA COSÌ È AMMISSIBILE

FF CONSIDERO UNO SPAZIO DI STATI DI UN PROBLEMA



RILASSANDO IL PROBLEMA DIVENTÒ UN SUPERGRAFO



QUINDI:

- SOL DI G È SOL. DI G'
- SOL G' (DA CUI CALCOLO h_R) HA PERCORSI MIGLIORI (PATH CON COSTI SCORRETTI)
  ↓
  h_R NON SOVRASTIMA SOL OF G E RISPETTA Δ TRIANGOLARE

QUINDI h_R AMMISSIBILE $\wedge$ CONSISTENTE

MASSIMIZZARE EURISTICHE

SE HO UNA SERIE DI $h_s = \{h_1, h_2, \dots, h_k\}$ CHE NON DOMINANO FRA DI LORO, RICAVO H:

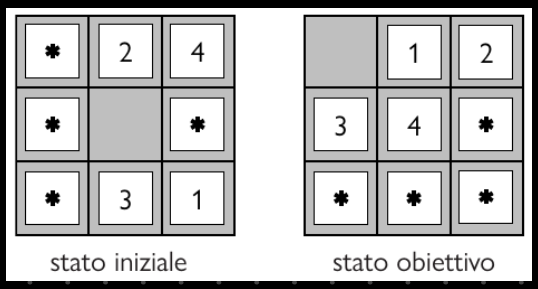
$$H = \max\{h_1(n), h_2(n), \dots, h_k(n)\} \quad \forall n$$

→ CALCOLO + ALTO PERÒ

AMMISSIBILE E DOMINA SU TUTTE

EURISTICHE DI SOTTO PROBLEMI: DATABASE PATTERN

SIA G-CORE UN SOTTO PROBLEMA DI 8-G, ESEMPIO



RILASSARE SOLO 4 PEZZI, IGNORANDO IL RESTO

DA QUI CREO UN DB:

DB = $\forall$ COSTI OF SOLUZ. DI OGNI ISTANZA POSS. DI G-GAME $\Rightarrow$ # ISTANZE = $9 \times 8 \times 7 \times 6 \times 5 = 15120$

↳ RICAVO h_{DB} ↝ SOTTOP. DI 8-GAME, AMMISSIBILE

- DA CUI, IN 8-GAME, DURANTE $\mathcal{P}$:
  - AD OGNI ITERAZIONE (CON CONFIGURAZIONE) CONTROLLO
  - POSSO SFRUTTARE ALTRI SOTTO PROBLEMI (TIPO 5-6-7-P). RICAVARE h'_{DB} E SFRUTTARE MASSIMO
  - POCO UTILE (VELOCIZZA 15-GAME DI POCO) MA COSTRUIRE DB COSTOSO DI TEMPO E SPAZIO.

SOTTO PROBLEMI DISGIUNTI

MA QUESTE EURISTICHE DI SOTTOP., POSSONO ESSERE SOMMATE? NO. ↳ ALCUNE ISTANZE SI SOVRAPPONGONO.

CIOÈ HANNO PARTI DI SOLUZIONE CHE COINCIDONO.

COME FARE?

X 8-GAME:

NUOVO SOTTO PROBLEMA:

4-GAME ~> RIORDINARE '_' - 1-2-3-4 (OK LE ALTRE CASELLE "VUOTE", E VICEVERSA CON 5-6-7-8-'_',

ELIMINO QUINDI, NEL CONTEGGIO, LE MOSSE CHE INFLUENZANO GLI ALTRI SOTTOPROBLEMI

COSÌ POSSO SOMMARE I 2 COSTI E EURISTICHE