

Soluzione del problema come ricerca in uno spazio degli stati:

- Gran parte dei problemi di Intelligenza Artificiale hanno la ricerca (o controllo) come componente fondamentale.
- I problemi si possono modellare come **Problemi di Ricerca** in uno spazio degli stati (**Strategie di Ricerca**).
- **Spazio degli stati**
 - Lo spazio degli stati è l'insieme di tutti gli stati raggiungibili dallo stato iniziale con una qualunque sequenza di operatori.
- **Lo Spazio degli Stati è Caratterizzato da:**
 - Uno **stato iniziale** in cui l'agente sa di trovarsi (non noto a priori);
 - Un **insieme di azioni** possibili che sono disponibili da parte dell'agente (Operatori che trasformano uno stato in un altro o più formalmente una funzione successore $S(X)$ che riceve in ingresso uno stato e restituisce l'insieme degli stati raggiungibili).
 - Un **cammino** è una sequenza di azioni che conduce da uno stato a un altro.

1

TEST: RAGGIUNGIMENTO DEL GOAL

- La verifica può essere solo l'appartenenza dello stato raggiunto all'insieme dello stato (o degli stati) goal.
- A volte lo stato obiettivo può essere descritto in modo "astratto" attraverso proprietà (si pensi allo stato di scacco matto).
- **Altri obiettivi (non solo raggiungere il goal, ma...):**
 - trovare la sequenza di operatori che arrivano al goal;
 - trovare tutte le soluzioni;
 - trovare una soluzione ottima.
 - In quest'ultimo caso vuol dire che una soluzione può essere preferibile a un'altra.
 - Una funzione **costo di cammino** assegna un costo a un cammino (in gran parte dei casi quale somma del costo delle azioni individuali lungo il cammino).

2

Problem-solving agents

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  static: seq, an action sequence, initially empty
           state, some description of the current world state
           goal, a goal, initially null
           problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then do
    goal ← FORMULATE-GOAL(state)
    problem ← FORMULATE-PROBLEM(state, goal)
    seq ← SEARCH(problem)
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

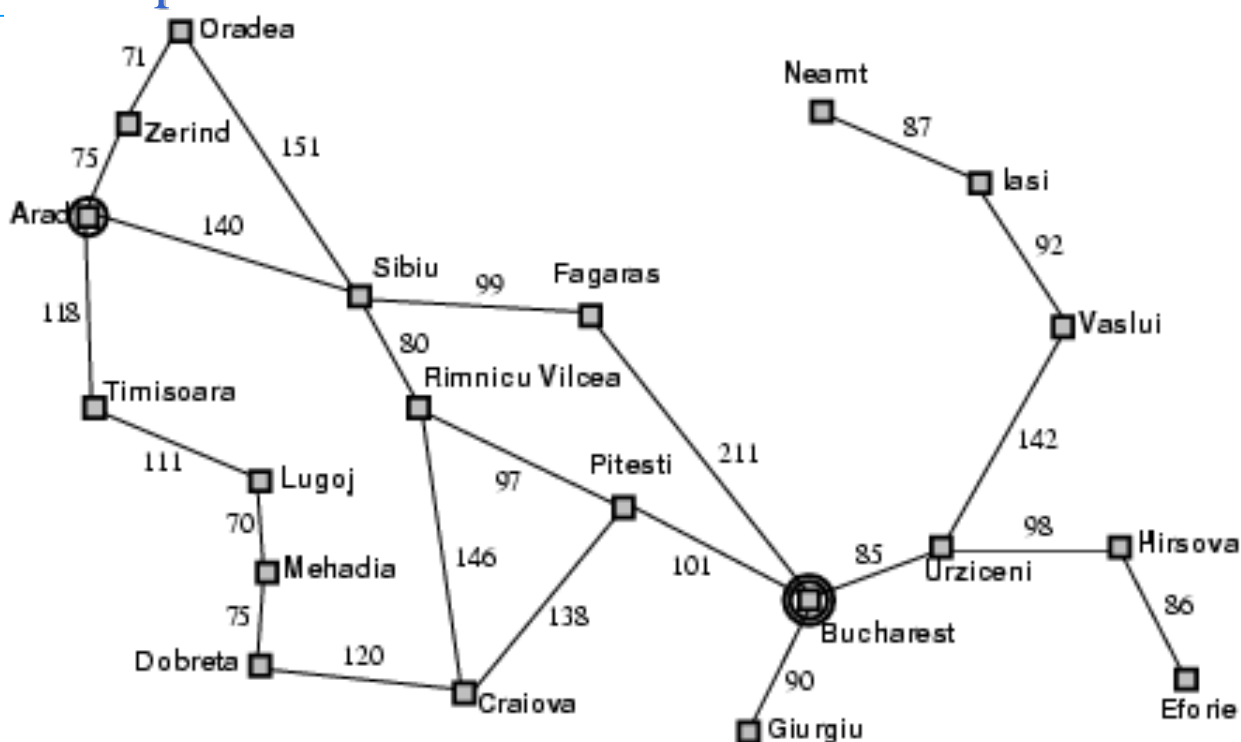
3

Esempio: Romania

- Una vacanza in Romania; attualmente in Arad.
- I voli partono da Bucharest domani
- **goal:**
 - Essere in Bucharest
- **problema:**
 - **stati:** varie città
 - **azioni:** guida fra le città
- **soluzione:**
 - Sequenza di città, ad es., Arad, Sibiu, Fagaras, Bucharest.

4

Esempio: Romania



5

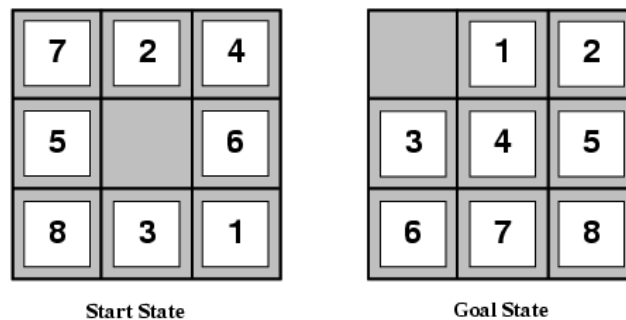
Formulazione del problema

Il **problema** e' definito da quattro punti:

1. **Stato iniziale** es., "at Arad"
 2. **Azioni o funzioni successore** $S(x)$ = insieme di coppie azione-stato
 - es., $S(\text{Arad}) = \{ \langle \text{Arad} \rightarrow \text{Zerind}, \text{Zerind} \rangle, \dots \}$
 3. **goal test**, puo' essere
 - **esplicito**, es., $x = \text{"at Bucharest"}$
 - **implicito**, es., $\text{controllamappa}(x)$
 4. **Costo della strada** es., somma delle distanza, numero di azioni eseguite ecc. $c(x,a,y) \geq 0$
- Una **soluzione** e' una sequenza di azioni che portano dallo stato iniziale al goal.

6

Esempio: The 8-puzzle



stati? Locazioni delle tessere

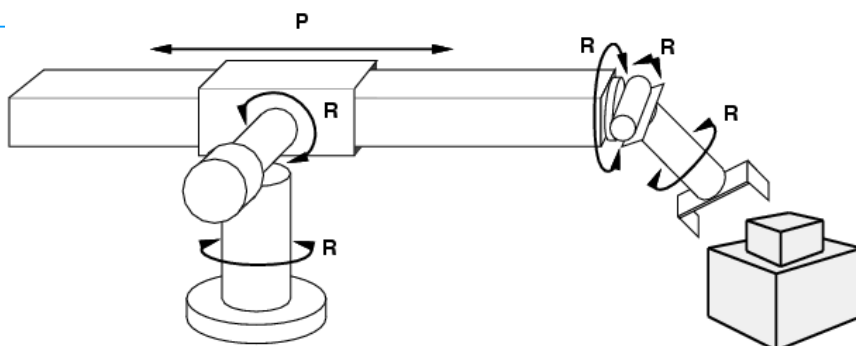
azioni? Muovi la lacuna a sinistra, destra, alto, basso.

goal test? = stato goal (dato)

costo? 1 per mossa.

7

Esempio: assemblaggio con un robot



stati: coordinate dei giunti del robot, parti dell'oggetto da assemblare e posizione.

actions?: movimenti dei giunti del robot.

goal test?: completamente assemblato

costo della strada: tempo di esecuzione.

8

ESEMPIO: MISSIONARI E CANNIBALI

ESEMPIO:

- 3 missionari e 3 cannibali devono attraversare un fiume. C'è una sola barca che può contenere al massimo due persone. Per evitare di essere mangiati i missionari non devono mai essere meno dei cannibali sulla stessa sponda (stati di fallimento).
- Stato: sequenza ordinata di tre numeri che rappresentano il numero di missionari, cannibali e barche sulla sponda del fiume da cui sono partiti.
- Perciò lo stato iniziale è: (3,3,1) (nota l'importanza dell'astrazione).

9

ESEMPIO: MISSIONARI E CANNIBALI

- Operatori: gli operatori devono portare in barca
 - 1 missionario, 1 cannibale,
 - 2 missionari,
 - 2 cannibali,
 - 1 missionario
 - 1 cannibale.
- Al più 5 operatori (grazie all'astrazione sullo stato scelta).
- Test Obiettivo: Stato finale (0,0,0)
- Costo di cammino: numero di traversate.

10

ESEMPIO: CRIPTOARITMETICA

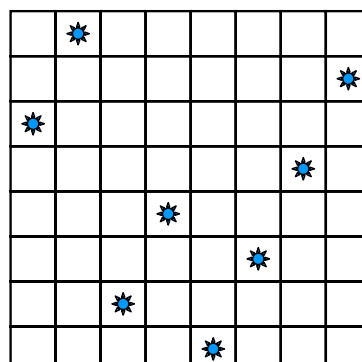
$$\begin{array}{rcccccc} & S & E & N & D & + \\ & M & O & R & E & = \\ \hline M & O & N & E & Y & \end{array}$$

- Operatori: sostituisci tutte le stesse lettere con una cifra che non compare nel rompicapo;
- Test obiettivo: il rompicapo contiene solo cifre e rappresenta una somma corretta;
- Costo di cammino: 0

11

IL PROBLEMA DELLE N REGINE

- Inserire 8 regine su una scacchiera in modo che non si mangino.
- Stati: qualsiasi configurazione da 0 a N regine sulla scacchiera;
- Operatori: aggiungi una regina in un qualsiasi quadrato;
- Test obiettivo: N regine sulla scacchiera, nessuna minacciata;
- Costo di cammino: zero.



12

COME SI GIOCA A SUDOKU

- Alcune caselle sono già fissate, le altre vanno riempite con numeri dall'1 al 9
- la tavola è suddivisa in 9 quadranti di 3x3 caselle
- su ogni quadrante devono essere messi tutti e 9 i numeri, senza ripetizioni
- inoltre, ogni riga orizzontale e ogni riga verticale dell'intera tavola non deve contenere ripetizioni di numeri

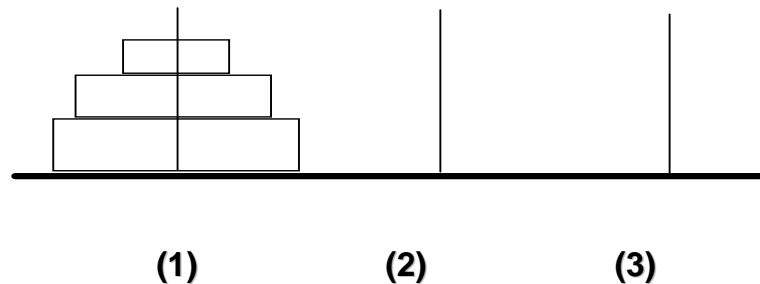
13

SUDOKU: griglia

		9				7		
	4		5		9		1	
3				1				2
	1			6			7	
		2	7		1	8		
	5			4			3	
7				3				4
	8		2		4		6	
		6				5		

14

ESEMPIO: LA TORRE DI HANOI



- Spostare i cilindri concentrici da (1) a (3) nella stessa configurazione di (1) utilizzando eventualmente anche (2). Un cilindro più grande non può essere inserito su un cilindro più piccolo.

15

ALTRI ESEMPI

- Il problema del commesso viaggiatore:
 - Un commesso viaggiatore ha una lista di città che deve visitare tutte una sola volta. Vi sono strade dirette fra ogni coppia di città. Si trovi la strada più breve che il commesso deve seguire per compiere un viaggio completo che inizi e termini in una qualsiasi delle città.
 - (esplosione combinatoria, per 10 città è 10!)
- Il problema della scimmia e la banana:
 - Afferrare una banana appesa al soffitto (avendo a disposizione una sedia e bastone).

16

ESEMPIO: CAPRA, LUPO E CAVOLO

- Portare capra, lupo e cavolo dall'altra parte senza che si mangino (si mangiano se rimangono assieme senza il conducente della barca). Nella barca se ne può trasportare uno solo alla volta.
- Soluzione:
 - Porta la capra sull'altra sponda;
 - Torna indietro
 - Porta il cavolo sull'altra sponda
 - Porta indietro la capra
 - Porta il lupo sull'altra sponda
 - Torna indietro
 - Porta la capra sull'altra sponda

17

SPAZIO DI RICERCA MOLTO AMPIO

- Giochi: scacchi
- Le parole crociate

T	U	O
R		R
E	R	A

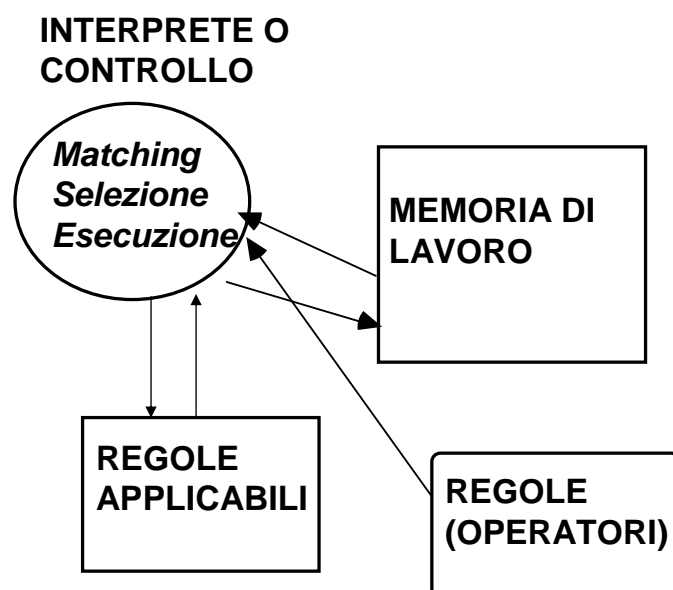
18

Schema architetturale: SISTEMA DI PRODUZIONI

- Insieme di Operatori (regole);
- Uno o più database (Memorie di lavoro);
- Strategia di Controllo.
- MODULARITÀ - FLESSIBILITÀ
- Operatori:
 - `IF <pattern> THEN <body>`
 - non si chiamano per nome, ma si attivano in base al pattern-matching

19

ARCHITETTURA GENERALE:



20

DUE MODALITÀ DI "RAGIONAMENTO"

- FORWARD O DATA-DRIVEN:
 - La memoria di lavoro nella sua configurazione iniziale contiene la conoscenza iniziale sul problema, cioè i fatti noti.
 - Le regole di produzione applicabili sono quelle il cui antecedente può fare *matching* con la memoria di lavoro (F-rules).
 - Ogni volta che una regola viene selezionata ed eseguita nuovi fatti dimostrati vengono inseriti nella memoria di lavoro.
 - Il procedimento termina con successo quando nella memoria di lavoro viene inserito anche il goal da dimostrare (condizione di terminazione).

21

DUE MODALITÀ DI "RAGIONAMENTO"

- BACKWARD O GOAL-DRIVEN:
 - La memoria di lavoro iniziale contiene il goal (o i goal) del problema.
 - Le regole di produzione applicabili sono quelle il cui conseguente può fare *matching* con la memoria di lavoro (B-rules).
 - Ogni volta che una regola viene selezionata ed eseguita, nuovi subgoals da dimostrare vengono inseriti nella memoria di lavoro.
 - Il procedimento termina con successo quando nella memoria di lavoro vengono inseriti fatti noti (CONDIZIONE DI TERMINAZIONE).

22

QUANDO APPLICARE BACKWARD E QUANDO FORWARD ?

- Esistono più Stati Iniziali o più Goals?
- Quale è il numero medio di rami generati da un singolo nodo?
- Quale è la modalità di ragionamento più naturale? (spiegazione all'utente)
- **BIDIREZIONALE O MISTO:**
 - È la combinazione dei metodi descritti precedentemente;
 - La memoria di lavoro viene suddivisa in due parti l'una contenente i fatti e l'altra i goals o subgoals;
 - Si applicano simultaneamente F-rules e B-rules alle due parti di memoria di lavoro e si termina il procedimento con successo quando la parte di memoria di lavoro ricavata mediante backward chaining è uguale o un sottoinsieme di quella ricavata mediante (CONDIZIONE DI TERMINAZIONE).

23

BIDIREZIONALE O MISTO

- L'idea è quella di mantenere ed estendere un insieme di sequenze soluzioni parziali.
 - Un agente con diverse opzioni immediate di esito sconosciuto può decidere cosa fare esaminando prima le differenti sequenze possibili di azioni che conducono a stati di esito conosciuto scegliendo, poi, quella migliore.
 - Il processo di cercare tale sequenza è chiamato **RICERCA**.
 - È utile pensare al processo di ricerca come la costruzione di un albero di ricerca i cui nodi sono stati e i cui rami sono operatori.
- Un algoritmo di ricerca prende come input un problema e restituisce una soluzione nella forma di una sequenza di azioni.
- Una volta che viene trovata la soluzione, le azioni suggerite possono essere realizzate.
 - Questa fase è chiamata **ESECUZIONE**.

24

ALTRE CARATTERISTICHE DEL PROBLEMA

- IL SISTEMA È DECOMPONIBILE?
 - Esempio:
 - $\int X + 3X + \sin X * \cos X dX$
- corrisponde alla somma degli integrali:
- $\int X dX + \int 3X dX + \int \sin X * \cos X dX$
- NON TUTTI I SISTEMI SONO DECOMPONIBILI:
 - Si possono avere sotto-problemi interagenti,

25

ESEMPIO: II MONDO A BLOCCHI (planning)

- Stato Iniziale:
`handempty. clear(b). clear(c).
on(c,a). ontable(a). ontable(b).`
- Operatori:
`clear(X) → ontable(X)
clear(X) and clear(Y) → on(X,Y)`
- Goal:
`on(b,c) and on(a,b)`
- I due sottogoal NON POSSONO ESSERE RISOLTI SEPARATAMENTE perché interagiscono.

26

ALTRE DOMANDE

- Esiste interazione con l'utente (spiegazione, conoscenza incompleta)?
- Si possono ignorare dei passi? (come nella dimostrazione di teoremi)
- Si possono annullare dei passi (8-puzzle) o il sistema è *irricoverabile* (scacchi)?
- SISTEMI DI PRODUZIONE MONOTONI:
 - L'applicazione di una regola R non invalida la possibilità di applicare regole applicabili all'atto della selezione di R.
- NON C'È NECESSITÀ DI BACKTRCKING

27

ESEMPIO: LOGICA DIMOSTRATORE DI TEOREMI

- Fatto:
 - a.
 - $a \rightarrow e.$
 - $a \rightarrow d.$
 - $d \rightarrow f.$
- Goal: $f.$
- Non è il caso del mondo a blocchi
- Nota:
 - Il formalismo dei sistemi di produzioni è generale (può rappresentare tutte le funzioni computabili).
 - È possibile trasformare un sistema di produzione non-monotono in uno monotono (formulazione di Green; formulazione di Kowalski). La trattazione, però, si complica notevolmente.

28

ALTRI PROBLEMI

- Problema a stati singoli:
 - Lo stato è sempre accessibile.
 - L'agente conosce esattamente che cosa produce ciascuna delle sue azioni e può calcolare esattamente in quale stato sarà dopo qualunque sequenza di azioni.
- Problema a stati multipli:
 - Lo stato non è completamente accessibile. L'agente deve ragionare su possibili stati che potrebbe raggiungere.
 - In più: anche l'effetto delle azioni può essere sconosciuto o imprevisto.
 - Spesso risolvere questi problemi richiede capacità di rilevamento durante la fase di esecuzione → Agire nel mondo reale piuttosto che in un suo modello.
- Tratteremo estesamente solo problemi a stati singoli.