Introduzione all'Intelligenza Artificiale

Federico Matteoni

A.A. 2019/20

Indice

Introduzione

Alessio Micheli, Maria Simi
`elearning.di.unipi.it/course/view.php?id=174`
Intelligenza Artificiale si occupa della **comprensione** e della **riproduzione** del comportamento *intelligente*.
Psicologia cognitiva: obiettivo comprensione intelligenza umana, costruendo modelli computazionali e verifica sperimentale.
Approccio costruttivo: costruire entità dotate di intelligenze e **razionalità**. Questo tramite codifica del pensiero razionale per risolvere problemi che richiedono intelligenza non necessariamente facendolo come lo fa l'uomo.
Definizioni di IA: pensiero-azione, umanamente-razionalmente.
Costruire macchine intelligenti sia che operino come l'uomo che diversamente.
formalizzaz conoscenze e meccanizzazione ragionemtno in tutti i settori dell'uomo
comprensione tramite modelli comp della psicologia e comportamente di uomini, animali ecc
rendere il lavoro con il calcolatore altrettanto facile e utile che del lavoro con persone capaci, abili e disponibili.

Poniamo definizione di IA: arte di creare macchine che svolgono funzioni che richiedono intelligenza quando svolte da esseri umani. Non definisce "Intelligenza", cosa significa "intelligente"?

Capitolo 1

Agenti Intelligenti

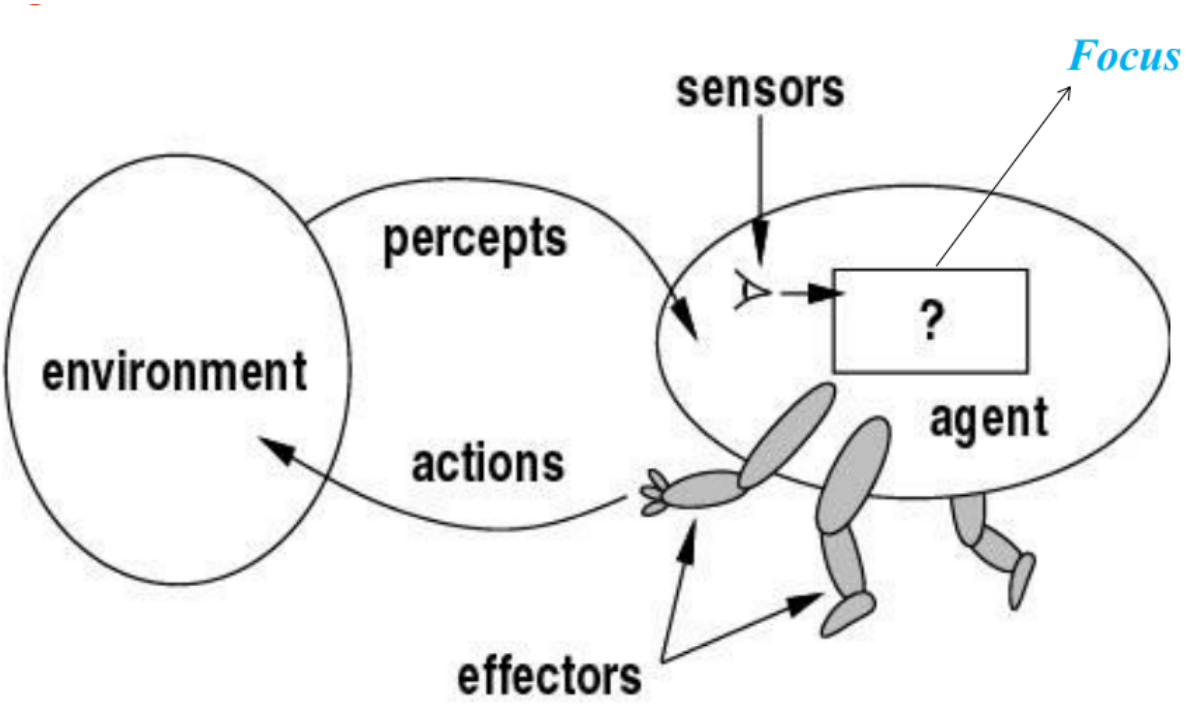
1.1 Intelligenza

L'intelligenza è vista come l'avere diverse capacità, durante il progresso nell'area di ricerca: buon senso, interazione con un ambiente, acquisizione di esperienza, comunicazione, ragionamento logico...

Considerazioni L'intelligenza quindi non è una collezione di tecniche per risolvere problemi **specifici**, ma per l'informatica consiste nel **fornire metodologie sistematiche per dotare le macchine di comportamenti intelligenti/*razionali* su problemi generali *difficili***.

1.2 Agenti

Iniziamo con inquadrare gli **agenti**. L'approccio moderno dell'IA consiste della costruzione di agenti intelligenti. Questa visione ci offre un quadro di riferimento ed una prospettiva **diversa** all'analisi dei sistemi software.
Il primo obiettivo sarà di costruire agenti per la risoluzione di problemi vista come una **ricerca in uno spazio di stati** (**problem solving**)



Ciclo *percezione- azione*

1.2.1 Caratteristiche

Sono qualcosa di più di un modulo software.

Situati Gli agenti sono **situati in un ambiente** da cui **ricevono percezioni** e su cui **agiscono** mediante **azioni** (**attuatori**).

Sociali Gli agenti hanno **abilità sociali**: comunicano, collaborano e si difendono da altri agenti.

Credenze, obiettivi, intenzioni...

Corpo Gli agenti hanno un **corpo**, sono **embodied** fino a considerare i meccanismi delle emozioni.

1.2.2 Percezioni e Azioni

Percezione Una percezione è un input da sensori.

Sequenza percettiva Storia **completa** delle percezioni
La **scelta delle azioni** è **unicamente determinata dalla sequenza percettiva**.

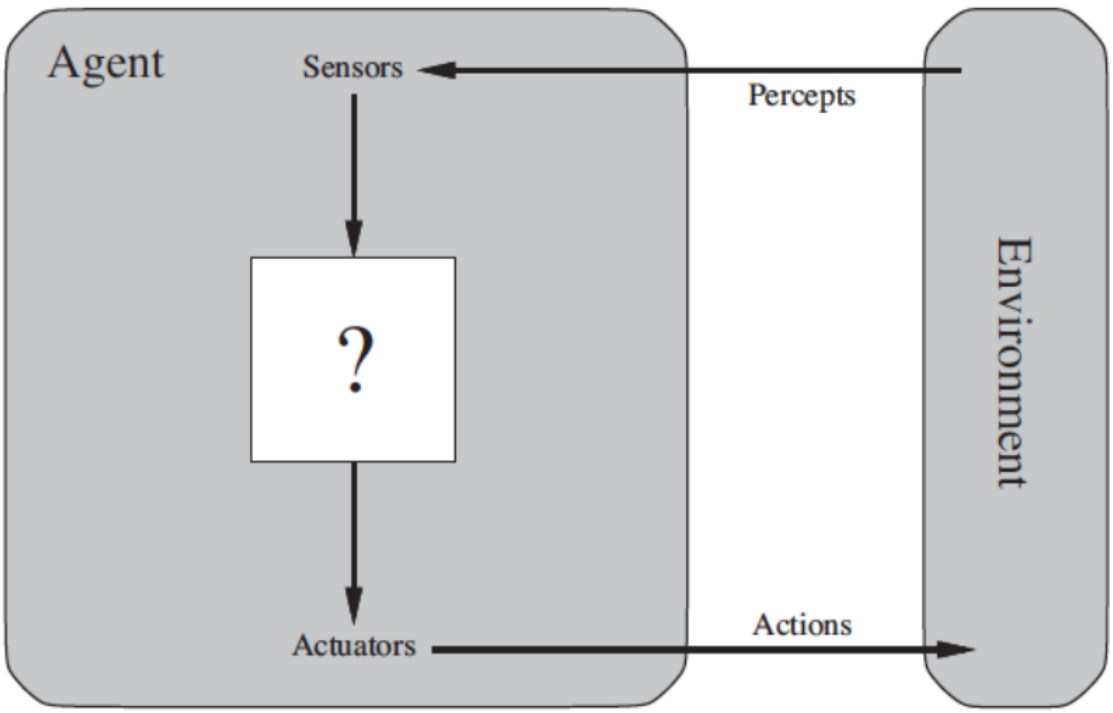
Funzione Agente Definisce l'azione da intraprendere per ogni sequenza percettiva e **descrive completamente l'agente**. Implementata da un **programma agente**.

$$\textbf{Sequenza Percettiva} \longrightarrow^{f} \textbf{Azione}$$

Il compito dell'IA è progettare il programma agente.

1.2.3 Agente e ambiente

Architettura astratta



Esempi

Agente robotico Percepisce con camera, microfoni e sensori. Interagisce con motori, voce...

Agente finanziario Percepisce i tassi, le news. Interagisce con acquisti e scambi.

Agente di gioco Percepisce le mosse dell'avversario. Interagisce tramite le proprie mosse.

Agente diagnostico Percepisce i sintomi e le analisi dei pazienti. Interagisce fornendo la diagnosi.

Agente web Percepisce le query utente e le pagine web. Interagisce fornendo i risultati di ricerca.

1.2.4 Agenti Razionali

Agenti razionali Un agente razionale **interagisce con l'ambiente in maniera efficace**: "*fa la cosa giusta*". L'agente razionale raggiunge l'obiettivo nella maniera più efficiente.
Serve quindi una **misura di prestazione**, di *come vogliamo che il mondo evolva*, a seconda del problema e considerato l'ambiente.

Esterna, perché bisogna definirla *prima* di agire. Non si può definire l'obiettivo dopo aver iniziato ad agire, altrimenti non è significativo.
Esempio: la volpe che non arriva all'uva.

Scelta dal progettista a seconda del problema e considerando l'effetto che ha sull'ambiente.

Razionalità La razionalità è relativa/dipende da:

Misura delle prestazioni

Conoscenze pregresse dell'ambiente

Percezioni presenti e passate (sequenza percettiva)

Capacità dell'agente (le azioni possibili)

Definizione Un **agente razionale**, quindi, **esegue l'azione che massimizza il valore atteso della misura delle prestazioni per ogni sequenza di percezioni**, considerando le sue percezioni passate e la sua conoscenza pregressa.
Non si pretende perfezione e conoscenza del futuro, ma massimizzare il risultato *atteso*. Potrebbero essere necessarie azioni di acquisizione di informazioni o esplorative (**non onniscenza**).
Le capacità dell'agente possono essere limitate (**non onnipotenza**).

Razionalità e apprendimento Raramente il programmatore può fornire a priori tutta la conoscenza sull'ambiente. L'agente razionale, quindi, **deve essere in grado di modificare il proprio comportamento con l'esperienza**, cioè con le percezioni passate.
Può migliorarsi esplorando, **apprendendo**, aumentando la propria autonomia per operare in ambienti differenti o mutevoli.

1.2.5 Agenti Autonomi

Un agente è **autonomo quando il suo comportamento dipende dalla sua esperienza**. Se il suo comportamento fosse determinato solo dalla propria conoscenza *built-int* allora sarebbe **non autonomo** e poco flessibile.

1.3 Ambienti

Definire un problema per un agente significa **caratterizzare l'ambiente in cui lavora**, cioè l'**ambiente operativo**. L'agente razionale è la soluzione del problema.

1.3.1 PEAS

Performance, prestazioni

Environment, ambiente

Actuators, attuatori

Sensors, sensori

Esempio Autista di taxi

Prestazione	**Ambiente**	**Attuatori**	**Sensori**
Arrivare alla destinazione, sicuro, veloce, ligio alla legge, confortevole, consumo minimo di benzina, profitti massimi	Strada, altri veicoli, clienti	Sterzo, acceleratore, freni, frecce, clacson	Telecamere, sensori, GPS, contachilometri, accelerometro, sensori del motore...

Formulazione PEAS dei problemi

Problema	**P**	**E**	**A**	**S**
Diagnosi medica	Diagnosi corretta	Pazienti, ospedale	Domande, suggerimenti, test, diagnosi	Sintomi, test clinici, risposte del paziente
Analisi immagini	Numero di immagini/zone correttamente classificate	Collezione di fotografie	Etichettatore di zone nell'immagine	Array di pixel
Robot "selezionatore"	Numero delle parti correttamente classificate	Nastro trasportatore	Raccogliere le parti e metterle nei cestini	Telecamera (pixel di varia intensità)
Giocatore di calcio	Fare più goal dell'avversario	Altri giocatore, campo di calcio, porte	Dare calci al pallone, correre	Locazione del pallone, dei giocatori e delle porte

1.3.2 Simulatore di Ambienti

Uno **strumento software** con il compito di:

Generare gli stimoli per gli agenti

Raccogliere le azioni in risposta

Aggiornare lo stato dell'ambiente

Opzionalmente, attivare altri processi che influenzano l'ambiente

Valutare le prestazioni degli agenti

Gli esperimenti su classi di ambienti (variando le condizioni) sono essenziali per valutare la capacità di generalizzare. La valutazione delle prestazioni è fatta tramite la media su più istanze.

1.3.3 Proprietà dell'Ambiente-Problema

- **Osservabilità**
  Completamente osservabile: l'apparato percettivo è in grado di dare una conoscenza completa dell'ambiente o almeno tutto quello che serve a decidere l'azione.
  Parzialmente osservabile: sono presenti limiti o inaccuratezze nell'apparato sensoriale. (Es. la videocamera di un rover vede solo parte dell'ambiente in un dato istante).

- **Singolo/Multi-Agente**
  Distinzione tra agente e non agente: il mondo può cambiare anche attraverso **eventi**, non necessariamente per le azioni di agenti.
  Multi-Agente Competitivo, come gli scacchi: comportamento randomizzato ma razionale.
  Multi-Agente Cooperativo, o benigno: stesso obiettivo e comunicazione.

- **Predicibilità**
  Deterministico: lo stato successivo è completamente determinato dallo stato corrente e dall'azione.
  Stocastico: esistono elementi di incertezza con probabilità associata. Es: guida, tiro in porta.
  Non deterministico: si tiene traccia di più stati possibili che sono risultato dell'azione, ma non in base ad una probabilità.

- **Episodico**: l'esperienza dell'agente è divisa in episodi atomici indipendenti. In ambienti episodici non c'è bisogno di pianificare.
  Sequenziale: ogni decisione influenza le succesive.

- **Statico**: il mondo non cambia mentre l'agente decide l'azione.
  Dinamico: l'ambiente cambia nel tempo, va osservata la contingenza. Tardare equivale a non agire.
  Semi-dinamico: l'ambiente non cambia ma la valutazione dell'agente si. Es: scacchi con timer, se non agisco prima dello scadere perdo.

- **Discreto/Continuo**
  Lo stato, il tempo, le percezioni e le azioni sono tutti elementi che possono assumere valori discreti o continui. Combinatoriale (nel discreto) *vs* infinito (nel continuo).

- **Noto/Ignoto**
  Distinzione riferita allo stato di conoscenza dell'agente sulle leggi fisiche dell'ambiente. L'**agente conosce l'ambiente o deve compiere azioni esplorative**?
  Noto $\neq$ **osservabile**: posso giocare a carte coperte, ma con regole note.

Ambienti reali Parzialmente osservabili, stocastici, sequenziali, dinamici, continui, multi-agente e ignoti.

1.4 Struttura di un Agente

Agente = **Architettura** + **Programma**
Ag: P $\longrightarrow$ Az

L'**Ag**ente associa **Az**ioni alle **P**ercezioni. Il **programma dell'agente** implementa la funzione **Ag**.

Programma Agente Pseudocodice del programma agente.

```
function Skeleton-Agent(percept) returns action
        static: memory #la memoria del mondo posseduta dall'agente
        memory <- UpdateMemory(memory, percept)
        action <- Choose-Best-Action(memory) #Cuore dell'IA
        memory <- UpdateMemory(memory, action)
        return action
```

1.4.1 Strutture di Agenti Caratteristici

Agente basato su tabella La scelta dell'azione è un accesso ad una tabella che **associa un'azione ad ogni possibile sequenza di percezioni**.
Vari **problemi**:

Le **dimensioni** possono essere proibitive: per giocare a scacchi, la tabella dovrebbe contenere un numero di righe nell'ordine di $10^{120} >> 10^{80}$ numero di atomi nell'universo osservabile.

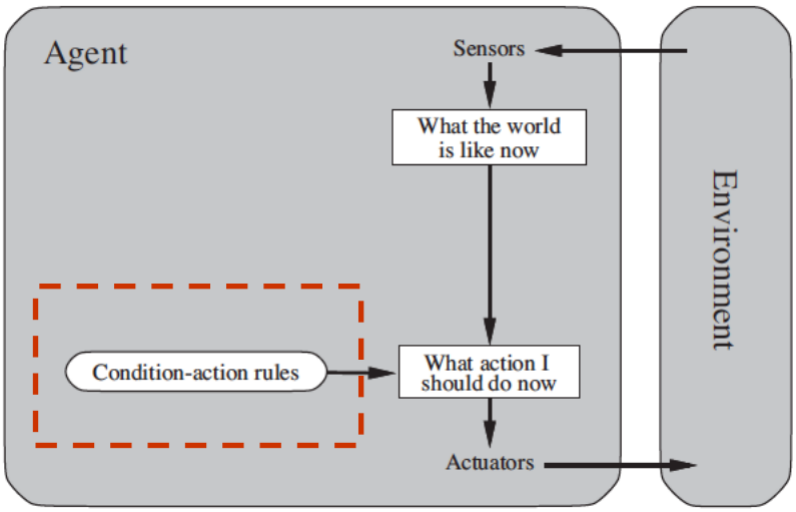
Difficile da costruire

Nessuna autonomia

Difficile da aggiornare, apprendimento complesso.

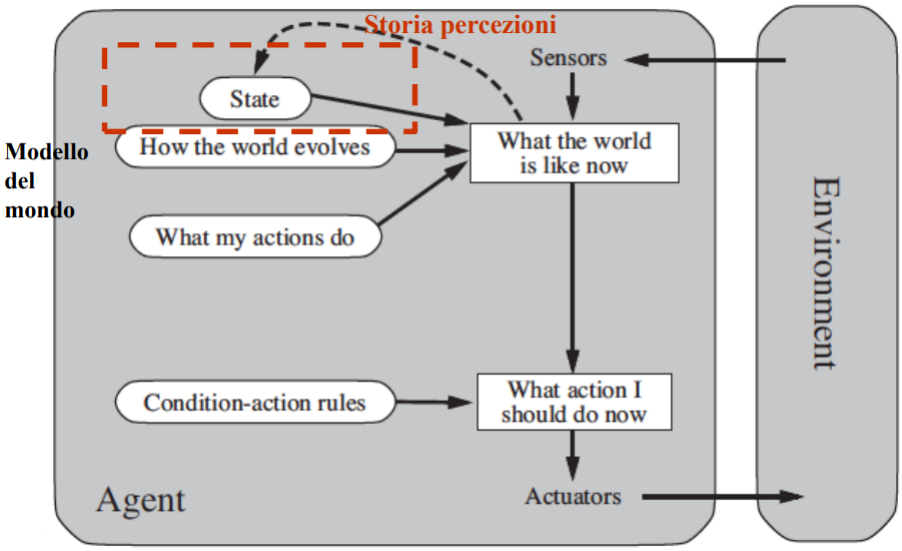
Con le IA vogliamo realizzare **automi razionali con un programma *compatto***.

Agente Reattivo Semplice



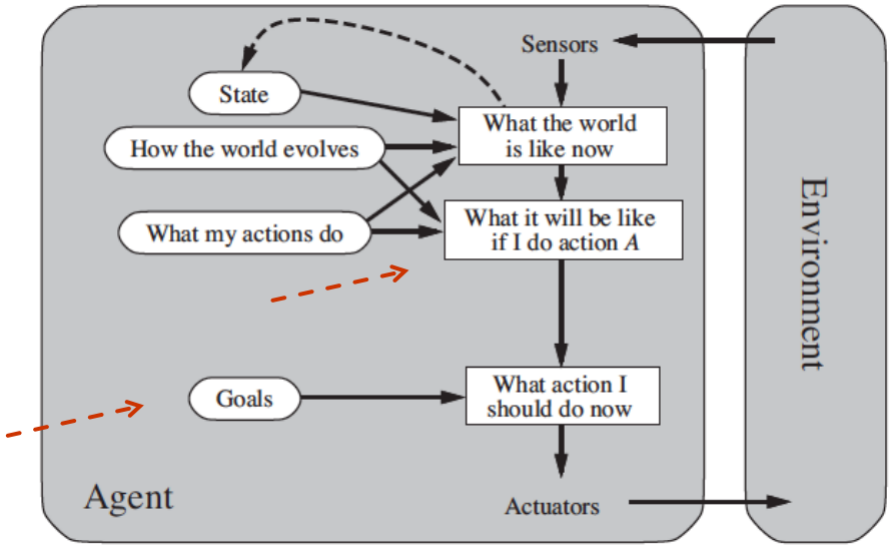

```
function Agente-Reattivo-Semplice(percezione) returns azione
        persistent: regole #insieme di regole condizione-azione (if-then)
        stato <- Interpreta-Input(percezione)
        regola <- Regola-Corrispondente(stato, regole)
        azione <- regola.Azione
        return azione
```

Agenti basati su modello



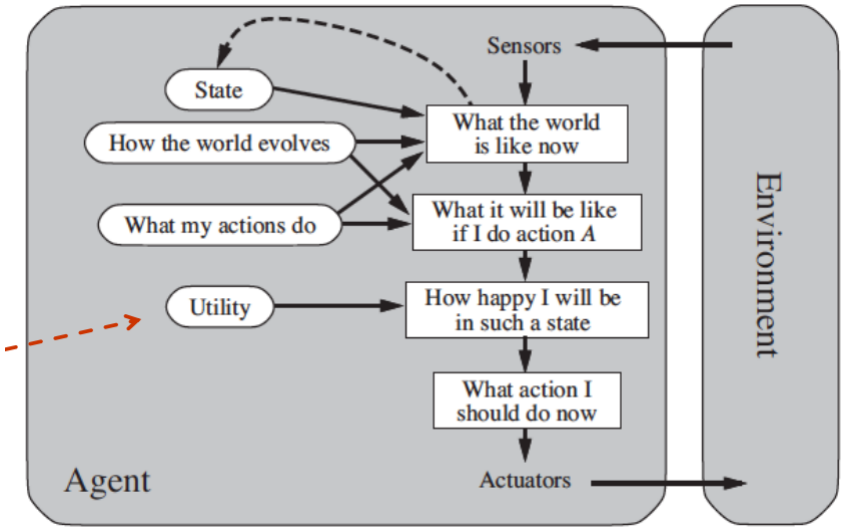

```
function Agente-Basato-su-Modello(percezione) returns azione
        persistent:     stato #descrizione dello stato corrente
                        modello #conoscenza del mondo
                        regole #insieme di regole condizione-azione
                        azione #azione piu recente
        stato <- Aggiorna-Stato(stato, azione, percezione, modello)
        regola <- Regola-Corrispondente(stato, regole)
        azione <- regola.Azione
        return azione
```

Agenti con obiettivo Bisogna pianificare una sequenza di azioni per raggiungere l'obiettivo. (In rosso sono indicate le parti aggiunte)



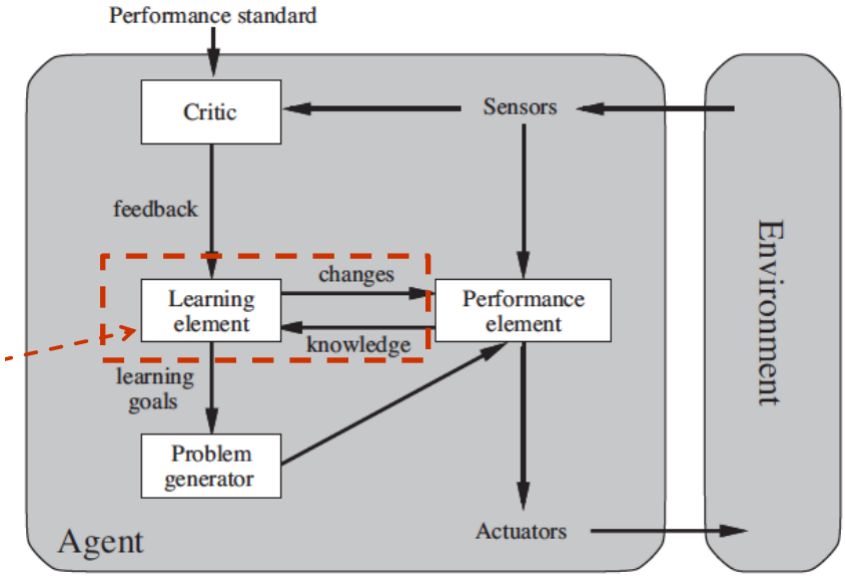
Sono **guidati da un obiettivo nella scelta che intraprendono**, è stato fornito un **goal esplicito**: per esempio una città da raggiungere.
A volte l'azione migliore dipende dall'obiettivo da raggiungere (*da che parte devo girare?*)
Devo **pianificare una sequenza di azioni** per raggiungere l'obiettivo. Sono meno efficienti ma **più flessibili** rispetto ad un agente reattivo. L'obiettivo può cambiare, non è codificato nelle regole.
Esempio classico: ricerca della sequenza di azioni per raggiungere una data destinazione.

Agenti con valutazione di utilità



Obiettivi alternativi, o più modi per raggiungerlo: l'agente deve decidere verso quali muoversi, quindi è **necessaria una funzione di utilità** che associa ad uno stato obiettivo un numero reale.
Obiettivi più facilmente raggiungibili di altri: la funzione di utilità **tiene conto della probabilità di successo** e/o di ciascun risultato (**utilità attesa** o media)

Agenti che apprendono



Componente di apprendimento: produce cambiamenti al programma agente. Migliora le prestazioni, adattando i suoi componenti ed apprendendo dall'ambiente

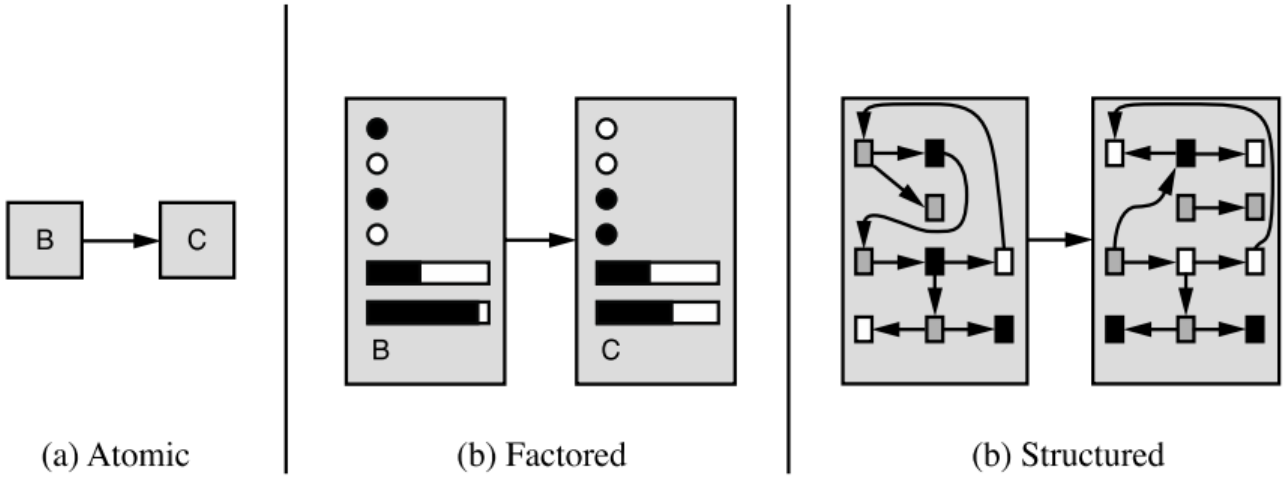
Elemento esecutivo: il programma agente

Elemento critico: osserva e dà feedback sul comportamento

Generatore di problemi: suggerisce nuove situazioni da esplorare

1.4.2 Tipi di rappresentazione

Stati e transizioni



Rappresentazione atomica (stati)

Rappresntazione fattorizzata (+ variabili e attributi)

Rappresentazione strutturata (+ relazioni)

Capitolo 2

Problem Solving

2.1 Agenti Risolutori di Problemi

Problem Solving Questi agenti adottano il paradigma della **risoluzione di problemi come ricerca in uno spazio di stati** (**problemi solving**). Sono **agenti con modello** (storia, percezioni) che **adottano una rappresentazione atomica dello stato**. Sono **particolari agenti con obiettivo** che **pianificano l'intera sequenza di azioni** prima di agire.

2.1.1 Processo di risoluzione

Passi da seguire

1. **Determinazioni dell'obiettivo**: un insieme di stati dove l'obiettivo è soddisfatto.
2. **Formulazione del problema**: rappresentazione degli stati e delle azioni.
   Fa parte del design "umano".
3. **Determinazione della soluzione** mediante ricerca: un piano d'azione
4. **Esecuzione del piano**
   Soluzione algoritmica.

La determinazione dell'obiettivo e la formulazione del problema richiede **tanta intelligenza**, che in fase di design è **spostata sull'umano**. Gli algoritmi sono ancora "*stupidi*".

Assunzioni sull'ambiente **Statico**, **osservabile** (so dove sono, es: *viaggio con la mappa*), **discreto** (insieme finito di azioni possibili), **deterministico** (una azione ⇒ un risultato. L'agente può eseguire il piano "*ad occhi chiusi*", niente può andare storto)

Formulazione del problema Un problema può essere **definito formalmente** mediante cinque componenti:

1. **Stato iniziale**
2. **Azioni possibili** nello stato `s`: `Azioni(s)`
3. **Modello di transizione**
   `Risultato`: stato × azione ⟶ stato
   `Risultato(s, a)`: `s'`, uno stato **successore**
4. **Test obiettivo**: un insieme di stati obiettivo
   `Goal-Test`: stato ⟶ {`true`, `false`}
5. **Costo del cammino**: somma dei costi delle azioni (costo dei passi).
   Costo di un passo: `c(s, a, s')`, mai negativo.

1, 2 e 3 **definiscono *implicitamente* lo spazio degli stati**. Definirlo esplicitamente può essere molto oneroso, come in quasi tutti i problemi di IA.

2.2 Algoritmi di Ricerca

Il processo che cerca una sequenza di azioni che raggiunge l'obiettivo è detto ***ricerca****.*

Algoritmi Gli algoritmi di ricerca prendono in **input un problema** e **restituiscono un cammino soluzione**, un cammino che porta dallo stato iniziale allo stato goal.

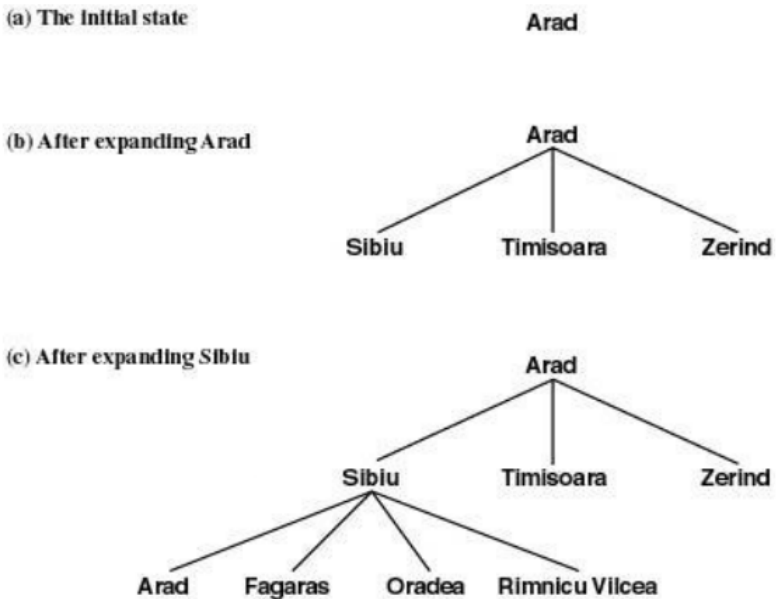
Misura delle prestazioni Trova una soluzione? Quanto costa trovarla? Quanto è efficiente la soluzione?

$$\text{Costo Totale} = \text{Costo della Ricerca} + \text{Costo del Cammino Soluzione}$$

Valuteremo algoritmi sul primo, ottimizzando il secondo.

2.2.1 Ricerca ad Albero

Generazione di un **albero di ricerca sovrapposto allo spazio degli stati**. Ricerca significa **approfondire l'opzione**, mettendo da parte le altre che verranno riprese se non trovo la soluzione.
Quindi l'albero viene generato esplorando i vari nodi partendo dallo stato iniziale. Il nodo è diverso dallo stato: per esempio, in un grafo rappresentante le città, se parto da città A ed esploro l'opzione nodo B, il nodo B avrà come figlio anche città A perché posso tornarci.



Algoritmo **Ricerca ad albero**, ossia senza controllare se i nodi (**stati**) siano già stati esplorati.

```
function Ricerca-Albero(problema) returns soluzione oppure fallimento
        #Inizializza la frontiera con stato iniziale del problema
        loop do
        if (frontiera vuota)
                return fallimento
        #Scegli* un nodo foglia da espandere e rimuovilo dalla frontiera
        if (nodo contiene uno stato obiettivo)
                return soluzione corrispondente
        #Espandi il nodo e aggiungi i successori alla frontiera
```

$*$ = **strategia**: quale scegliere? I vari algoritmi si differenziano per la strategia di scelta.
Un **nodo** n è una **struttura dati con quattro componenti**

Stato, n.stato

Padre, n.padre

Azione effettuata per generarlo, n.azione

Costo del cammino dal nodo iniziale al nodo, n.costo-cammino
Indicata come g(b) = padre.costo-cammino + costo-passo ultimo)

Frontiera Lista dei **nodi in attesa di essere espansi**, cioè **le foglie** dell'albero di ricerca. Implementata come una coda con operazioni:

Vuota(coda)

Pop(coda) estrae l'ultimo elemento (implementa la strategia)

Inserisci(elemento, coda)

Diversi tipi di coda hanno differenti funzioni di inserimento e **implementano strategie diverse**.

FIFO → BF
Viene estratto l'elemento più vecchio, cioè in attesa da più tempo. Nuovi nodi aggiunti alla fine

LIFO → DF
Viene estratto l'ultimo elemento inserito. Nuovi nodi aggiunti all'inizio

Con priorità → UC, altri...
Viene estratto l'elemento con priorità più alta in base ad una funzione di ordinamento. All'aggiunta di un nuovo nodo si riordina.

Strategie non informate

Ricerca in **ampiezza** (BF)

Ricerca in **profondità** (DF)

Ricerca in **profondità limitata** (DL)

Ricerca con **apprendimento iterativo** (ID)

Ricerca di **costo uniforme** (UC)

Strategie informate Anche dette di **ricerca euristica**: fanno uso di informazioni riguardo la distanza stimata della soluzione.

Valutazione di una strategia

Completezza: se la soluzione esiste viene trovata

Ottimalità (ammissibilità): trova la soluzione migliore, con costo minore

Complessità in tempo: tempo richiesto per trovare la soluzione

Complessità in spazio: memoria richiesta

2.2.2 Breadth-First

Ricerca in ampiezza Esplorare il grafo dello spazio degli stati a livelli progressivi di stessa profondità. Implementata con una coda FIFO. **Algoritmo su albero**:

```
function RicercaAmpiezzaA(problema)      returns soluzione oppure fallimento
        nodo = un nodo con stato = problema.stato-iniziale e costo-di-cammino = 0
        #Stati goal-tested alla generazione: maggior efficienza si ferma appena trova goal
        if (problema.TestObiettivo(nodo.Stato)) return Soluzione(nodo)
        frontiera = una coda FIFO con nodo come unico elemento
        loop do
        if (Vuota(frontiera)) return fallimento
        nodo = Pop(frontiera)
        for each azione in problema.Azioni(nodo.Stato) do #Espansione
                figlio = Nodo-Figlio(problema, nodo, azione) #costruttore: vedi AIMA
                if (Problema.TestObiettivo(figlio.Stato)) return Soluzione(figlio)
                frontiera = Inserisci(figlio, frontiera) #frontiera coda FIFO
```

Algoritmo su grafo evitando di espandere stati già esplorati:

```
function RicercaAmpiezzaG(problema) returns soluzione oppure fallimento
        nodo = un nodo con stato = problema.stato-iniziale e costo-di-cammino = 0
        if (problema.TestObiettivo(nodo.Stato)) return Soluzione(nodo)
        frontiera = una coda FIFO con nodo come unico elemento
        esplorati = insieme vuoto #gestisco stati ripetuti
        loop do
        if (Vuota(frontiera)) return fallimento
        nodo = POP(frontiera) #aggiungi nodo.Stato a esplorati
        for each azione in problema.Azioni(nodo.Stato) do
                figlio = Nodo-Figlio(problema, nodo, azione)
                if (figlio.Stato non in esplorati e non in frontiera)
                        if (Problema.TestObiettivo(figlio.Stato)) return Soluzione(figlio)
                        frontiera = Inserisci(figlio, frontiera) #in coda
```

Python

```
def breadth_first_search(problem): """Ricerca-grafo in ampiezza"""
  explored = [] # insieme degli stati gia' visitati (implementato come una lista)
  node = Node(problem.initial_state) #il costo del cammino e' inizializzato nel costruttore
  if problem.goal_test(node.state):
    return node.solution(explored_set = explored)
  frontier = FIFOQueue() # la frontiera e' una coda FIFO
  frontier.insert(node)
  while not frontier.isempty(): # seleziona il nodo per l'espansione
    node = frontier.pop()
    explored.append(node.state) # inserisce il nodo nell'insieme dei nodi esplorati
    for action in problem.actions(node.state):
      child_node = node.child_node(problem,action)
      if (child_node.state not in explored) and
      (not frontier.contains_state(child_node.state)):
        if problem.goal_test(child_node.state):
        return child_node.solution(explored_set = explored)
      # se lo stato non e' uno stato obiettivo allora inserisci il nodo nella frontiera
      frontier.insert(child_node)
  return None # in questo caso ritorna con fallimento
```

Analisi della complessità spazio-temporale Assumiamo:

b = fattore di ramificazione (**b**ranching)

d = profondità del nodo obiettivo più superficiale (**d**epth)
Più vicino all'iniziale

m = lunghezza massima dei cammini nello spazio degli stati (**m**ax)

Analisi:

Strategia **completa**

Strategia **ottimale** *se gli operatori hanno tutti lo stesso costo k* cioè g(n) = k · depth(n), dove g(n) è il costo del cammino per arrivare ad n.

Complessità nel tempo (nodi generati)
$T(b, d) = b + b^2 + \ldots + b^d = O(b^d)$, con b figli per ogni nodo.

Complessità nello spazio (nodi in memoria): $O(b^d)$