

Appunti di Introduzione all’Intelligenza Artificiale Unipi - Parte 1

Raffaele Apetino

Marzo 2020

Contents

1 Premessa	3
2 Introduzione	3
2.1 Test di Turing	3
2.2 Deep Learning	4
3 Agenti Intelligenti	4
3.1 Test di Turing	4
3.2 Agenti Razionali	5
3.3 Agenti Autonomi	5
4 Ambienti	5
4.1 Descrizione PEAS dei problemi	5
4.2 Proprietà dell’ambiente	6
4.3 Simulatore di ambienti	6
5 Agenti - Struttura di un agente	7
5.1 Agente basato su tabella	7
5.2 Agenti reattivi semplici	7
5.3 Agenti basati su modello	8
5.4 Agenti con obiettivo	8
5.5 Agenti con valutazione di utilità	9
5.6 Agenti che apprendono	9
6 Agenti risolutori di problemi	10
6.1 Formulazione di un problema	10
6.2 Algoritmi di ricerca	10
6.3 Il problema dell’itinerario	10
6.3.1 Formulazione del problema dell’itinerario	11
6.4 Il problema dell’aspirapolvere	11
6.4.1 Formulazione del problema dell’aspirapolvere	11
7 Ricerca della soluzione	12
7.1 Strategie non informate VS Strategie informate "euristiche"	13
7.2 Valutazione di una strategia	13
7.3 Ricerca in ampiezza - BF	13
7.3.1 BF-Albero	13
7.3.2 BF-Grafo	14

7.3.3	Analisi complessità BF	14
7.4	Ricerca in profondità - DF	15
7.4.1	Analisi complessità DF-Albero	15
7.5	Ricerca in profondità ricorsiva - RecursiveDF	15
7.6	Ricerca in profondità limitata - DL	15
7.6.1	Analisi complessità DL	15
7.7	Approfondimento Iterativo - ID	16
7.7.1	Analisi complessità ID	16
7.8	Ricerca Bidirezionale - Bidir.	16
7.8.1	Analisi complessità Bidir.	16
7.9	Problemi cammini ciclici e ridondanze	16
7.10	Ricerca di costo uniforme - UC	17
7.10.1	UC-Albero	18
7.10.2	UC-Grafo	18
7.10.3	Analisi complessità UC	18
7.11	Confronto finale delle strategie	19
8	Ricerca Euristica	19

1 Premessa

Questi appunti sono stati scritti e controllati più volte, ma come ben si sa:

Sa chi sa che nulla sa, e chi sa che nulla sa ne sa più di chi ne sa.

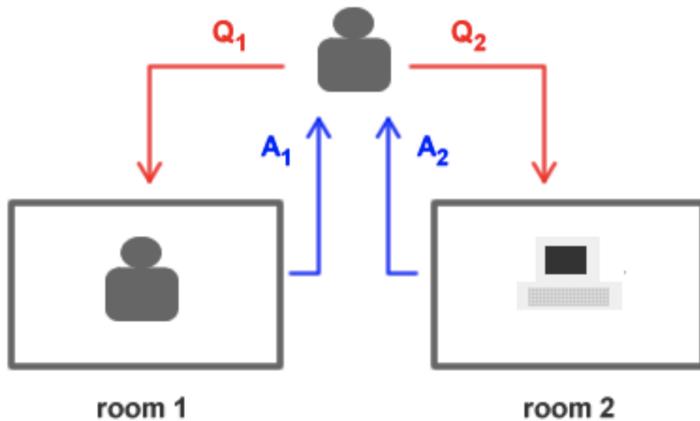
Quindi è bene non prendere queste mie parole come leggi, ma anzi, vi invito a guardare per bene le slide del corso e studiare su quelle. Magari utilizzate questi appunti per un ripasso veloce prima della gogna. Per segnalare errori o migliorie mandatemi una mail a r.apetino at studenti.unipi.it o fate una bella Pull Request su GitHub!

2 Introduzione

L'intelligenza artificiale si occupa della comprensione e riproduzione del comportamento intelligente. L'approccio psicologico (psicologia cognitiva) ha come obiettivo la comprensione dell'intelligenza umana e quindi risolvere i problemi con gli stessi processi usati dall'uomo. L'approccio informatico è quello di costruire entità dotate di razionalità e quindi si occupa dell'automazione del comportamento intelligente. Quest'ultimo viene eseguito attraverso la meccanizzazione del ragionamento, comprensione mediante modelli computazionali della psicologia e del comportamento degli uomini.

C'è però una domanda molto importante riguardo a cosa sia l'intelligenza: capacità di ragionamento? Buon senso? Capacità sociali e di comunicazione? Capacità di comprendere e provare emozioni?

2.1 Test di Turing



In due stanze separate ci sono una persona ed un computer, fuori da queste due stanze c'è una seconda persona che fa domande ad entrambi ad entrambi con la porta chiusa. Il test di Turing si basa su dopo quanto tempo l'uomo esterno riesce a capire chi è uomo e chi macchina.

Da qui ci viene una domanda fondamentale, dobbiamo dotare i computer di senso comune? (esistono già dei progetti nominati CYC e OpenMind) Una definizione di senso comune possiamo darla, è la capacità di un uomo di poter riconoscere in modo immediato ricorrendo all'uso della ragione naturale. Quindi possiamo anche dare una definizione di intelligenza: è la qualità mentale che consiste nell'abilità di apprendere dall'esperienza, di adattarsi a nuove situazioni, comprendere e gestire concetti astratti, utilizzare la conoscenza per agire sul proprio ambiente.

2.2 Deep Learning

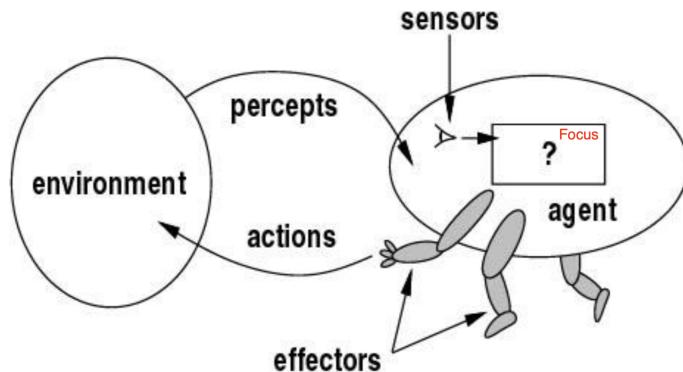
L'apprendimento profondo (deep learning) è quel campo di ricerca dell'apprendimento automatico (machine learning) e dell'intelligenza artificiale che si basa su diversi livelli di rappresentazione, corrispondenti a gerarchie di caratteristiche di fattori o concetti, dove i concetti di alto livello sono definiti sulla base di quelli di basso. In altre parole, per apprendimento profondo si intende un insieme di tecniche basate su reti neurali artificiali organizzate in diversi strati, dove ogni strato calcola i valori per quello successivo affinché l'informazione venga elaborata in maniera sempre più completa. L'apprendimento automatico si basa sull'estrazione di modelli statistici predittivi da immense quantità di dati (data mining).

L'intelligenza artificiale è pericolosa? Andrebbe regolata? Le macchine dovrebbero avere un etica? Ma su quale etica dovrebbero basarsi? L'intelligenza collettiva può essere estratta o inferita dai dati?

3 Agenti Intelligenti

L'approccio moderno dell'IA si basa sulla costruzione di agenti intelligenti e sulla creazione del programma agente da parte del programmatore. La visione ad agenti ci offre un quadro di riferimento e una prospettiva diversa dall'analisi dei sistemi software. Il nostro primo obiettivo è realizzare agenti per la risoluzione di problemi vista come ricerca in uno spazio di stati.

3.1 Test di Turing



Agenti Intelligenti:

- Sono situati: ricevono percezioni da un ambiente (tramite input dei sensori), agiscono sull'ambiente mediante azioni (sfruttando gli attuatori). La sequenza percettiva è la storia completa delle percezioni. La scelta dell'azione è funzione unicamente della sequenza percettiva
- Gli agenti hanno abilità sociale: sono capaci di comunicare, collaborare, difendersi da altri agenti.
- Gli agenti hanno credenze, obiettivi, intenzioni...
- Gli agenti sono embodied: hanno un corpo, fino a considerare i meccanismi delle emozioni.

La funzione agente definisce l'azione da compiere per ogni sequenza percettiva ed è implementata da un programma agente, come dicevamo prima, il nostro compito è proprio quello di progettare il programma agente.

3.2 Agenti Razionali

Un agente razionale interagisce con il suo ambiente in maniera efficace cioè "fa la cosa giusta". Serve quindi un criterio di valutazione oggettivo dell'effetto delle azioni dell'agente (vedremo che l'azione ha come conseguenza la creazione di un nuovo stato) come può essere il costo minimo di un cammino per arrivare alla soluzione. La razionalità è relativa alla misura delle prestazioni, alle conoscenze pregresse dell'ambiente e alle capacità dell'agente.

Definizione agente razionale:

Per ogni sequenza di percezioni compie l'azione che massimizza il valore atteso della misura delle prestazioni, considerando le sue percezioni passate e la sua conoscenza pregressa.

Raramente tutta la conoscenza sull'ambiente può essere fornita "a priori", l'agente razionale deve essere in grado di modificare il proprio comportamento con l'esperienza (percezioni passate oppure percezioni che è in grado di apprendere in futuro).

3.3 Agenti Autonomi

Modificare il proprio comportamento con l'esperienza implica la creazione di agenti autonomi:

Un agente è autonomo nella misura in cui il suo comportamento dipende dalla sua esperienza. Un agente il cui comportamento fosse determinato solo dalla sua conoscenza built-in, sarebbe non autonomo e poco flessibile

4 Ambienti

Definire un problema per un agente significa caratterizzare l'ambiente in cui l'agente opera

4.1 Descrizione PEAS dei problemi

- Performance (prestazione, obiettivo)
- Environement (ambiente, dove attua le sue azioni)
- Actuators (attuatori, meccanismi con cui agisco sull'ambiente)
- Sensor (sensori, percezioni)

4.2 Proprietà dell'ambiente

- L'ambiente è osservato dall'agente che ne apprende le sue caratteristiche.
 - Completamente osservabile: conoscenza completa dell'ambiente, non c'è bisogno di mantenere uno stato del mondo esterno.
 - Parzialmente osservabile: sono presenti limiti o inaccuratezze che riguardano la conoscenza del mondo.
- Il mondo può cambiare anche per eventi, non necessariamente per azioni di agenti.
 - Agente singolo.
 - Multi-agente: può essere a sua volta competitivo oppure cooperativo quindi con lo stesso obiettivo (comunicano).
- Si possono predire i cambiamenti del mondo.
 - Deterministico: se lo stato successivo è completamente determinato dallo stato corrente e dall'azione.
 - Stocastico: esistono elementi di incertezza con associata probabilità.
 - Non deterministico: non si può sapere come evolve il mondo quindi si tiene traccia di più stati possibili risultanti da una azione eseguita.
- – Episodico: l'esperienza dell'agente è divisa in episodi atomici indipendenti.
- – Sequenziale: ogni decisione influenza le successive.
- – Statico: il mondo non cambia mentre l'agente è fermo e sta decidendo l'azione.
- – Dinamico: il mondo cambia nel tempo, tardare equivale a non agire.
- – Semi-dinamico: l'ambiente non cambia ma la valutazione dell'agente sì.
- – Discreto: i valori del mondo sono limitati (ad esempio gli stati possono essere di numero finito)
- – Continuo: i valori del mondo possono essere infiniti (ad esempio il tempo può essere infinito)
- Lo stato di conoscenza dell'agente può essere:
 - Noto: conosco l'ambiente, questo non significa che sia osservabile (ad esempio in un gioco di carte, le carte sono note ma se sono coperte non sono osservabili)!
 - Ignoto: devo compiere azioni esplorative per conoscerlo tutto.

Gli ambienti reali sono parzialmente osservabili, stocastici, sequenziali, dinamici, continui, multi-agente, ignoti.

4.3 Simulatore di ambienti

E' uno strumento software che si occupa di generare stimoli per gli agenti, raccogliere le azioni di risposta, aggiornare lo stato dell'ambiente e valutare le prestazioni dell'agente.

5 Agenti - Struttura di un agente

Struttura Agente = Architettura + Programma

Funzione Agente¹ = Ag : Percezioni → Azioni

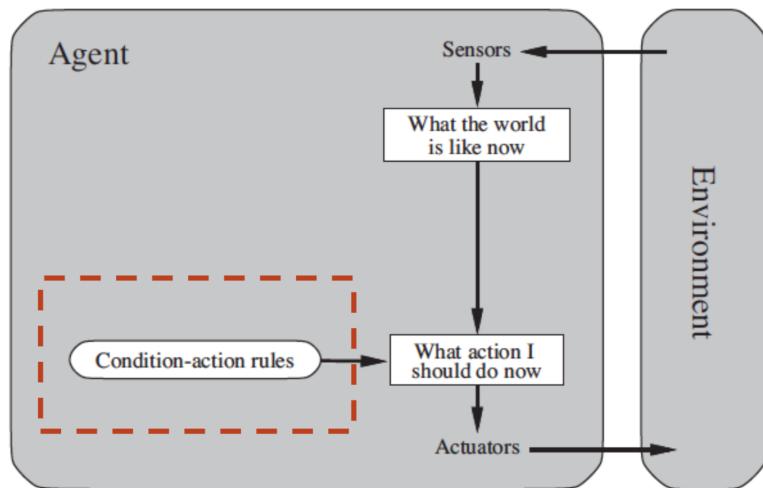
Il programma dell'agente implementa la funzione Ag.

5.1 Agente basato su tabella

La scelta dell'azione è un accesso a una tabella che associa un'azione ad ogni possibile sequenza di percezioni. Ci sono ovvi problemi:

1. Dimensione: Per giocare a scacchi la tabella ha un numero di righe molto maggiore di 10^{80} perciò la situazione è ingestibile.
2. Difficile da costruire.
3. Nessuna autonomia.
4. Di difficile aggiornamento, apprendimento complesso.

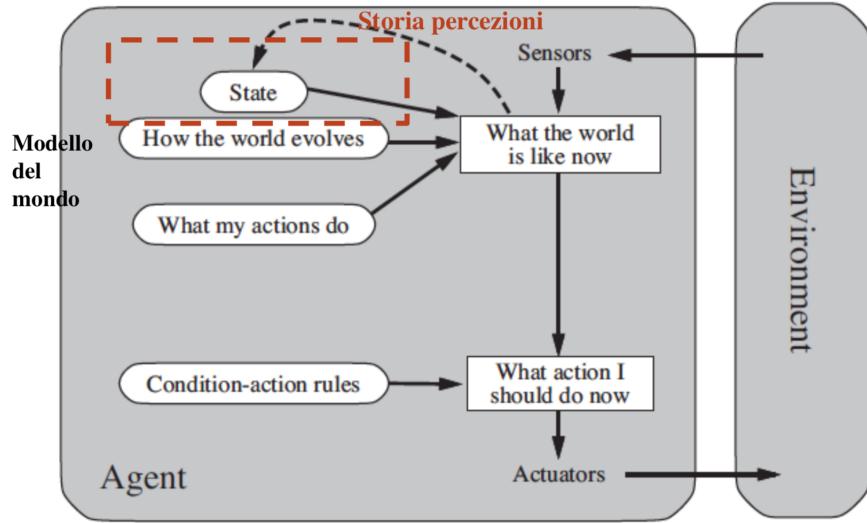
5.2 Agenti reattivi semplici



Sono presenti delle regole if-then costruite a priori che mi dicono quale azione fare in base allo stato e alle regole in quel dato istante.

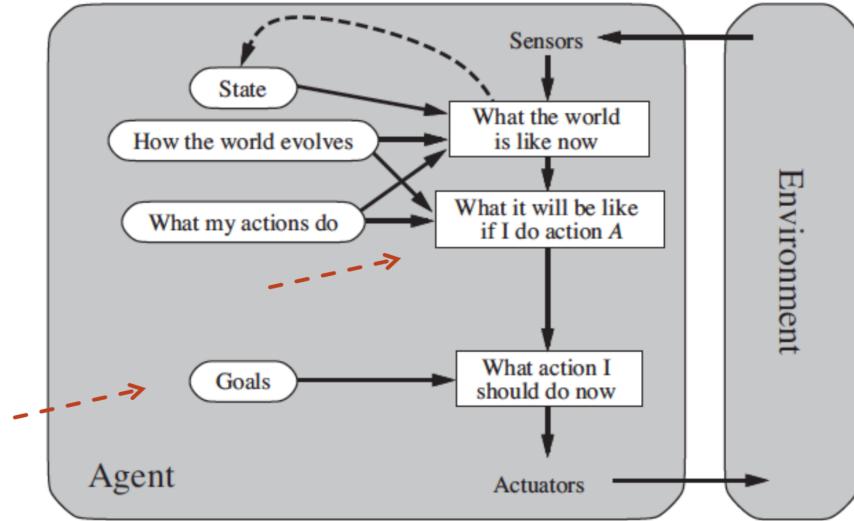
¹La funzione agente definisce l'azione da compiere per ogni sequenza percettiva

5.3 Agenti basati su modello



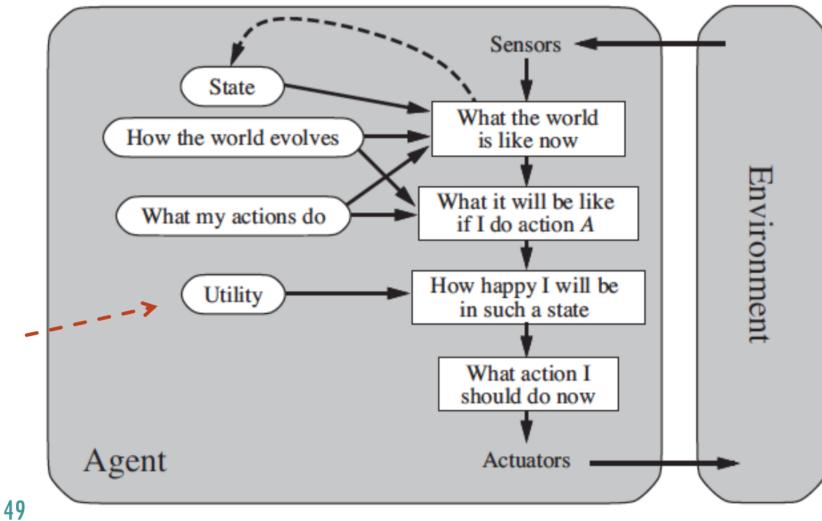
E' presente un modello del mondo che comprende lo stato aggiornato con la storia delle percezioni. Sono ancora presenti le regole if-then.

5.4 Agenti con obiettivo



Sono agenti guidati da un obiettivo nella scelta dell'azione (viene fornito un goal esplicito, ad esempio raggiungere una città). L'azione migliore dipende da quale obiettivo bisogna raggiungere e pianificano le proprie azioni in base al goal. L'agente si preoccupa di capire come sarà il mondo dopo aver eseguito una azione.

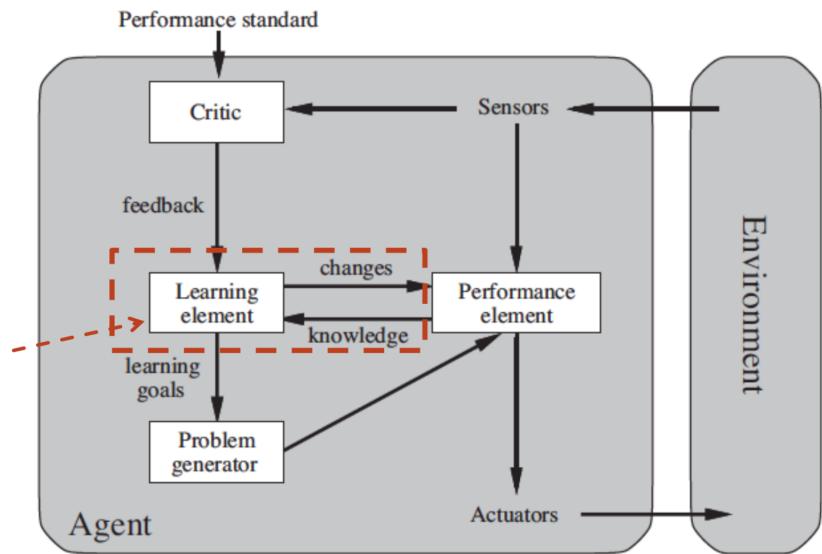
5.5 Agenti con valutazione di utilità



49

Ci sono obiettivi alternativi magari più facilmente raggiungibili. L'agente deve decidere verso quali di questi muoversi. E' necessaria una funzione di utilità che associa ad uno stato un numero reale. La funzione di utilità tiene conto anche della probabilità di successo e di utilità attesa.

5.6 Agenti che apprendono



E' presente una componente di apprendimento che produce cambiamenti al programma agente, migliora le prestazioni adattando i suoi componenti, apprendendo dall'ambiente. L'elemento esecutivo è il programma agente, l'elemento critico osserva e dà feedback sul comportamento. Infine è presente un generatore di problemi, suggerisce nuove situazioni da esplorare.

6 Agenti risolutori di problemi

Adottano il paradigma della risoluzione di problemi come ricerca in uno spazio di stati. Sono agenti con modello che adottano una rappresentazione atomica dello stato, hanno un obiettivo e pianificano l'intera sequenza di mosse prima di agire.

Passi da seguire:

1. Determinare un obiettivo (un insieme di stati tali che l'obiettivo è soddisfatto)
2. Formulare un problema (rappresentazione degli stati e delle azioni)
3. Determinare soluzione mediante ricerca
4. Esecuzione soluzione

L'ambiente è statico, osservabile, discreto e deterministico.

6.1 Formulazione di un problema

Un problema può essere definito formalmente mediante 5 componenti:

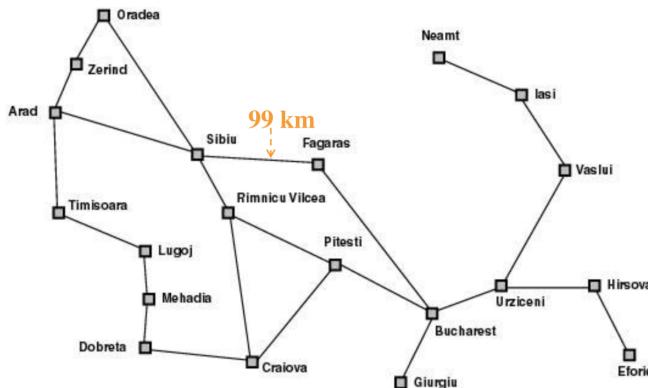
1. Stato iniziale
 2. Azioni possibili nello stato
 3. Modello di transizione: Risultato(stato, azione) = nuovo stato
 4. Test obiettivo: insieme di stati obiettivo
 5. Costo del cammino: somma dei costi delle azioni, costo di passo definito come $c(s,a,s')$
- 1, 2 e 3 definiscono implicitamente lo spazio degli stati.

6.2 Algoritmi di ricerca

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

Gli algoritmi di ricerca prendono in input un problema e restituiscono un cammino soluzione. La misura delle prestazioni è definita come: Costo totale = costo della ricerca + costo del cammino soluzione. Valuteremo algoritmi riguardo la ricerca ottimizzando il cammino soluzione.

6.3 Il problema dell'itinerario



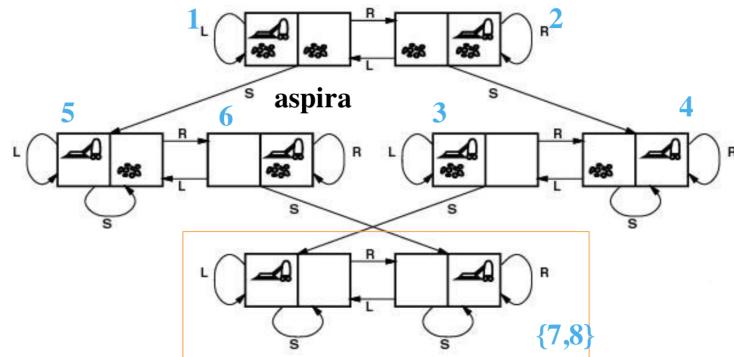
Il problema dell'itinerario riguarda la ricerca del percorso più breve da una città di partenza a una città di arrivo.

6.3.1 Formulazione del problema dell'itinerario

1. Stati: le città
2. Stato iniziale: la città da cui si parte ($In(Arad)$)
3. Azioni: spostarsi su una città vicina collegata ($Azioni(In(Arad)) = \{Go(Sibiu), Go(Zerind), \dots\}$)
4. Modello di transizione ($Risultato(In(Arad), Go(Sibiu)) = In(Sibiu)$)
5. Test obiettivo $\{In(Bucarest)\}$
6. Costo del cammino: somma delle lunghezze delle strade

Lo spazio degli stati coincide con la rete (grafo) di collegamenti tra città.

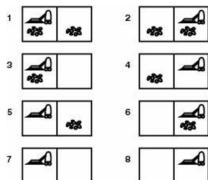
6.4 Il problema dell'aspirapolvere



Il problema dell'aspirapolvere riguarda la pulizia di due stanze adiacenti con il minimo sforzo.

6.4.1 Formulazione del problema dell'aspirapolvere

1. Stati:



2. Stato iniziale: stati 1 o 2 del grafo
3. Percezioni: Sporco - Non sporco
4. Azioni: Sinistra (L) - Destra (R) - Aspira (S)
5. Modello di transizione: Aspira(stanza) \rightarrow stanza pulita, Destra \rightarrow si sposta nella stanza a destra, Sinistra \rightarrow si sposta nella stanza a sinistra
6. Test obiettivo: rimuovere lo sporco (stati 7 o 8)
7. Costo del cammino: ogni azione ha costo 1

7 Ricerca della soluzione

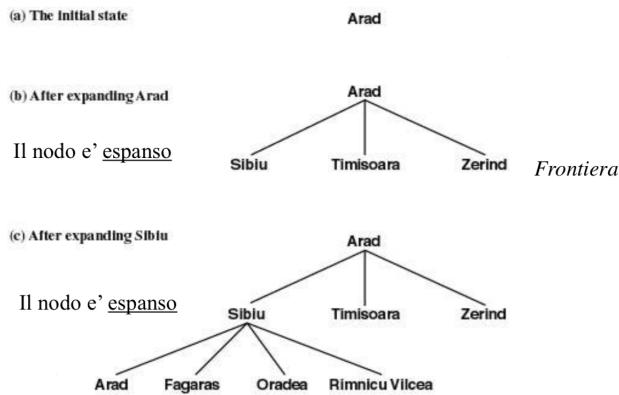
Si tratta di generare un albero di ricerca sovrapposto allo spazio degli stati (generato da possibili sequenze di azioni) senza controllare se i nodi (stati) siano già stati esplorati. Questo controllo lo vedremo sui grafi.

Un nodo n è una struttura dati con quattro componenti:

1. Uno stato: n.estado
2. Il nodo padre: n.padre
3. L'azione effettuata per generarlo: n.azione
4. Il costo del cammino a partire dal nodo iniziale: n.costocammino → $g(n) = n.padre.costocammino + \text{costo dell'ultimo passo}$

La frontiera è lista dei nodi in attesa di essere espansi (le foglie dell'albero di ricerca). Essa è implementata come una coda FIFO (viene estratto l'elemento più vecchio), LIFO (viene estratto quello più recentemente inserito) o con priorità (viene estratto quello con priorità più alta). Su di essa sono definite le seguenti operazioni:

- Vuota(coda) // mi dice se la coda è vuota
- Pop(coda) // estrae il primo elemento dalla coda in base alla strategia utilizzata
- Inserisci(elemento,coda) // inserisce un elemento della coda



Per prima cosa inizializzo la frontiera con lo stato iniziale (Arad), ad ogni ciclo controllo se la frontiera è vuota, se lo è FAIL altrimenti scelgo un nodo della frontiera e lo rimuovo. Se il nodo rimosso è contenuto negli stati obiettivo OK altrimenti espando il nodo e aggiungo i successori alla frontiera. Il problema quindi è quale tra i nodi della frontiera scelgo?

7.1 Strategie non informate VS Strategie informate "euristiche"

Strategie non informate:

- Ricerca in ampiezza (BF)
- Ricerca in profondità (DF)
- Ricerca di costo uniforme (UC)
- Ricerca in profondità limitata (DL)
- Ricerca con approfondimento iterativo (ID)

Le strategie informate "euristiche" le vedremo dopo, fanno uso di informazioni riguardo alla distanza stimata dalla soluzione.

7.2 Valutazione di una strategia

- Completezza: se la soluzione viene trovata, quindi esiste
- Ottimalità (ammissibilità): trova la soluzione migliore con costo minore
- Complessità in tempo: tempo richiesto per trovare la soluzione
- Complessità in spazio: memoria richiesta

7.3 Ricerca in ampiezza - BF

Esplorare il grafo dello spazio degli stati a livelli progressivi di stessa profondità. La frontiera è implementata con una coda che inserisce alla fine (FIFO).

7.3.1 BF-Albero

```
function Ricerca-Aampiezza-A (problema)
    returns soluzione oppure fallimento
    nodo = un nodo con stato il problema.stato-iniziale e costo-di-cammino=0
    if problema.Test-Obiettivo(nodo.Stato) then return Soluzione(nodo)
    frontiera = una coda FIFO con nodo come unico elemento
loop do
    if Vuota?(frontiera) then 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) then return Soluzione(figlio)
                    frontiera = Inserisci(figlio, frontiera) /* frontiera gestita come coda FIFO
end
```

Nota che in questa versione gli stati sono goal-tested al momento in cui sono generati. → più efficiente, si ferma appena trova goal prima di espandere

7.3.2 BF-Grafo

```

function Ricerca-Ampiezza-g (problema)
    returns soluzione oppure fallimento
    nodo = un nodo con stato il problema.stato-iniziale e costo-di-cammino=0
    if problema.Test-Obiettivo(nodo.Stato) then return Soluzione(nodo)
    frontiera = una coda FIFO con nodo come unico elemento
    esplorati = insieme vuoto
    loop do
        if Vuota?(frontiera) then 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 è in esplorati e non è in frontiera then
                if Problema.TestObiettivo(figlio.Stato) then return Soluzione(figlio)
                frontiera = Inserisci(figlio, frontiera) /* in coda

```

Aggiunte in verde per gestire gli stati ripetuti

Nota che in questa versione gli stati sono goal-tested al momento in cui sono generati. → più efficiente, si ferma appena trova goal prima di espandere

"esplorati" è una lista dei nodi già visitati. Prima di espandere un nodo si controlla se lo stato era già stato incontrato o è già nella frontiera.

7.3.3 Analisi complessità BF

Assumiamo che:

b = fattore di ramificazione (branching)

d = profondità del nodo obiettivo più superficiale (depth)

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

- Strategia Completa: SI
- Strategia Ottimale: SI se gli operatori hanno tutti lo stesso costo k^2
- Complessità Tempo: $O(b^d)$ (*b* figli per ogni nodo)
- Complessità Spazio: $O(b^d)$ (occupa un sacco di memoria)

²cioè $g(n) = k * \text{depth}(n)$ dove $g(n)$ è il costo del cammino per arrivare a *n*

7.4 Ricerca in profondità - DF

Esplorare il grafo dello spazio degli stati arrivando in profondità per ogni nodo. La frontiera è implementata con una coda che inserisce i successori in testa alla lista (LIFO). Cancella rami già completamente esplorati ma tiene tutti i fratelli del path corrente, occupa così in memoria solo b^*m .

7.4.1 Analisi complessità DF-Albero

- Strategia Completa: NO, si possono creare dei loop
- Strategia Ottimale: NO
- Complessità Tempo: $O(b^m)$ (che può essere maggiore di $O(b^d)$)
- Complessità Spazio: $O(b * m)$ (drastico risparmio di memoria)

In caso di DF con visita su grafo si perdono i vantaggi della memoria: la memoria torna ad essere $O(b^d)$ ma così DF diventa completa su spazi degli stati finiti (al caso pessimo estende tutti i nodi) resta comunque non completa su spazi infiniti.

7.5 Ricerca in profondità ricorsiva - RecursiveDF

Ancora più efficiente in occupazione di memoria perché mantiene in memoria solo il cammino corrente (solo m nodi al caso pessimo). L'algoritmo è realizzato con "backtracking" che non necessita di tenere in memoria b nodi per ogni livello, ma salva lo stato su uno stack a cui torna in caso di fallimento per fare altri tentativi.

```
function Ricerca-DF-ricorsiva(nodo, problema)
    returns soluzione oppure fallimento
    if problema.TestObiettivo(nodo.Stato) then return Soluzione(nodo)
    else
        for each azione in problema.Azioni(nodo.Stato) do
            figlio = Nodo-Figlio(problema, nodo, azione)
            risultato = Ricerca-DF-ricorsiva(figlio, problema)
            if risultato ≠ fallimento then return risultato
        return fallimento
```

7.6 Ricerca in profondità limitata - DL

Si procede in profondità fino ad un certo livello predefinito l .

7.6.1 Analisi complessità DL

- Strategia Completa: solo per problemi in cui si conosce un limite superiore per la profondità della soluzione cioè se $d < l$
- Strategia Ottimale: NO
- Complessità Tempo: $O(b^l)$
- Complessità Spazio: $O(bl)$

7.7 Approfondimento Iterativo - ID

Ad ogni iterazione aumento il limite della ricerca in profondità limitata e rincomincio dalla radice.

7.7.1 Analisi complessità ID

- Strategia Completa: SI
- Strategia Ottimale: se gli operatori hanno tutti lo stesso costo
- Complessità Tempo: $O(b^d)$
- Complessità Spazio: $O(b * d)$

Miglior compromesso tra BF e DF, i nodi dell'ultimo livello sono generati una volta, quello del penultimo due, ..., quelli del primo d volte.

7.8 Ricerca Bidirezionale - Bidir.

Un problema che possiamo valutare è la direzione della ricerca.

- Ricerca in avanti: ricerca guidata dai dati, si esplora lo spazio di ricerca dallo stato iniziale allo stato obiettivo
- Ricerca all'indietro: ricerca guidata dall'obiettivo, si esplora lo spazio di ricerca a partire da uno stato goal e riconducendosi a sotto-goal fino a trovare uno stato iniziale.

In quale direzione conviene procedere? Conviene procedere nella direzione in cui il fattore di diramazione è minore. Procediamo in avanti quando gli obiettivi sono molti e abbiamo una serie di dati da cui partire. Procediamo all'indietro quando l'obiettivo è chiaramente definito oppure i dati del problema non sono noti e la loro acquisizione può essere guidata dall'obiettivo.

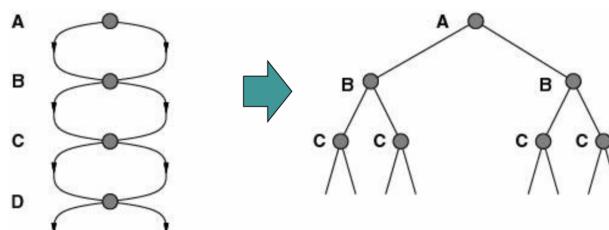
Nella ricerca bidirezionale si procede nelle due direzioni fino ad incontrarsi.

7.8.1 Analisi complessità Bidir.

- Strategia Completa: SI
- Strategia Ottimale: SI
- Complessità Tempo: $O(b^{d/2})$
- Complessità Spazio: $O(b^{d/2})$

7.9 Problemi cammini ciclici e ridondanze

I cammini ciclici rendono gli alberi di ricerca infiniti. Su spazi di stati a grafo si generano più volte gli stessi nodi (o meglio nodi con stesso stato) nella ricerca, anche in assenza di cicli.

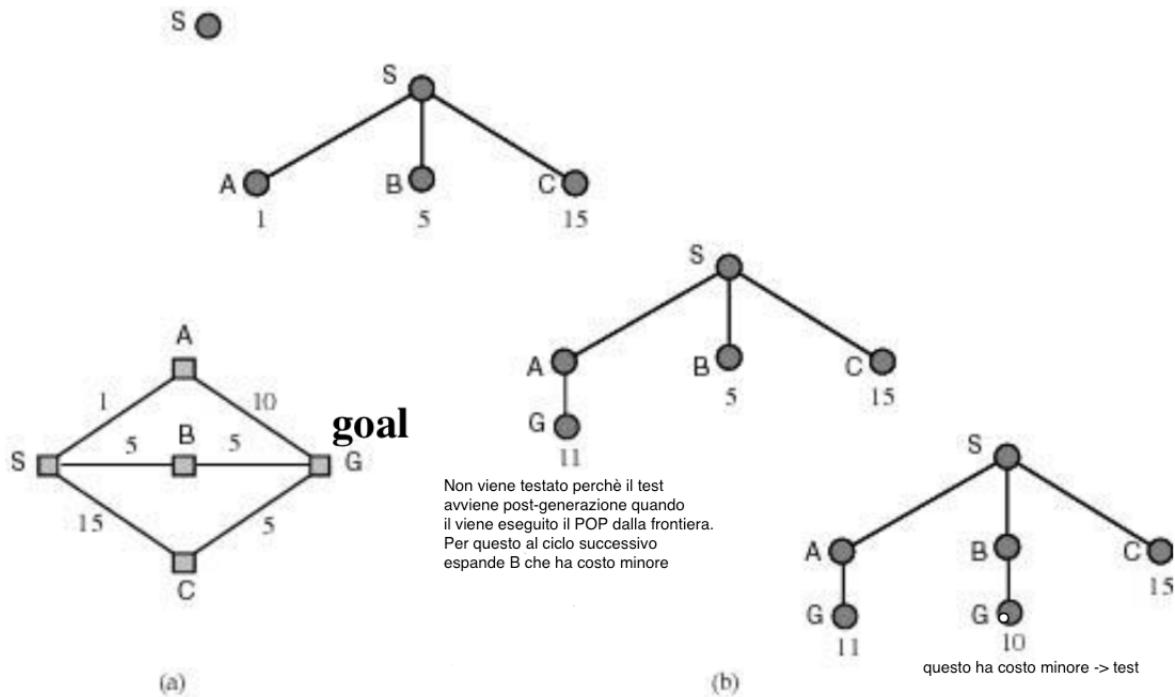


Ricordare gli stati già visitati occupa spazio ma ci consente di evitare di visitarli di nuovo. Ci sono tre soluzioni:

1. Non tornare nello stato da cui si proviene, si elimina il padre dai nodi successori (ma non evita cammini ridondanti).
2. Per evitare di creare cammini con cicli si controlla che i successori non siano antenati del nodo corrente.
3. Per non generare nodi con stati già visitati/esplorati teniamo in memoria ogni nodo visitato con complessità in spazio di $O(\text{Stati Possibili})$.

7.10 Ricerca di costo uniforme - UC

E' una generalizzazione della ricerca in ampiezza dove i costi di ogni operatore sono diversi. Si sceglie il nodo di costo minore sulla frontiera (costo $g(n)$) e si espande. La frontiera è implementata da una coda ordinata per costo cammino crescente (cioè per primi i nodi di costo minore)



7.10.1 UC-Albero

```

function Ricerca-UC-A (problema)
    returns soluzione oppure fallimento
    nodo = un nodo con stato il problema.stato-iniziale e costo-di-cammino=0
    frontiera = una coda con priorità con nodo come unico elemento
loop do
    if Vuota?(frontiera) then return fallimento
    nodo = POP(frontiera)
    if problema.TestObiettivo(nodo.Stato) then return Soluzione(nodo)
    for each azione in problema.Azioni(nodo.Stato) do
        figlio = Nodo-Figlio(problema, nodo, azione)
        frontiera = Inserisci(figlio, frontiera) /* in coda con priorità
    end

```

7.10.2 UC-Grafo

```

function Ricerca-UC-G (problema)
    returns soluzione oppure fallimento
    nodo = un nodo con stato il problema.stato-iniziale e costo-di-cammino=0
    frontiera = una coda con priorità con nodo come unico elemento
    esplorati = insieme vuoto
loop do
    if Vuota?(frontiera) then return fallimento
    nodo = POP(frontiera);
    if problema.TestObiettivo(nodo.Stato) then return Soluzione(nodo)
    aggiungi nodo.Stato a esplorati
    for each azione in problema.Azioni(nodo.Stato) do
        figlio = Nodo-Figlio(problema, nodo, azione)
        if figlio.Stato non è in esplorati e non è in frontiera then
            frontiera = Inserisci(figlio, frontiera) /* in coda con priorità
        else if figlio.Stato è in frontiera con Costo-cammino più alto then
            sostituisci quel nodo frontiera con figlio

```

g(n)

7.10.3 Analisi complessità UC

- Strategia Completa: SI se il costo degli archi x sia tale che $x \geq \alpha > 0$
- Strategia Ottimale: SI se il costo degli archi x sia tale che $x \geq \alpha > 0$
- Complessità Tempo: $O(b^{1+\lfloor C^*/\alpha \rfloor})$
- Complessità Spazio: $O(b^{1+\lfloor C^*/\alpha \rfloor})$

Assumendo C^* come costo della soluzione ottima e $\lfloor C^*/\alpha \rfloor$ come numero di mosse al caso peggiore, arrotondato per difetto.

7.11 Confronto finale delle strategie

	BF	UC	DF	DL	ID	Bidir.
Completa	SI	SI (-)	NO	SI (+)	SI	SI
Ottimale	SI (*)	SI (-)	NO	NO	SI (*)	SI
Tempo	$O(b^d)$	$O(b^{1+\lfloor C^*/\alpha \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Spazio	$O(b^d)$	$O(b^{1+\lfloor C^*/\alpha \rfloor})$	$O(b * m)$	$O(b * l)$	$O(b * d)$	$O(b^{d/2})$

(*) se gli operatori hanno tutti lo stesso costo

(-) per costo degli archi x tale che $x \geq \alpha > 0$

(+) per problemi per cui si conosce un limite alla profondità della soluzione (se $d < l$)

8 Ricerca Euristică