Ricerca *online*

Maria Simi
a.a. 2011/2012

Ambienti più realistici

- Gli agenti *risolutori di problemi* "classici" assumono:
  - Ambienti completamente osservabili e deterministici
  - il piano generato può essere generato *offline* e eseguito senza imprevisti
- Assunzioni da riconsiderare
  - Azioni non deterministiche
  - Ambiente parzialmente osservabile
  - Ambienti sconosciuti e problemi di esplorazione

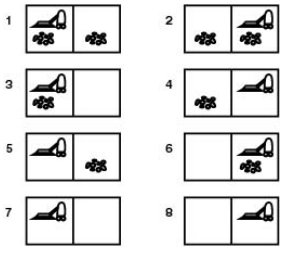
Soluzioni più complesse

- In un ambiente parzialmente osservabile e non deterministico le percezioni sono importanti
  - restringono gli stati possibili
  - informano sull'effetto dell'azione
- Più che un *piano* l'agente può elaborare una *strategia*, un *piano di contingenza* che tiene conto delle diverse eventualità
- L'aspirapolvere con assunzioni diverse …

Azioni non deterministiche
L'aspirapolvere imprevedibile

- Comportamento:
  - *Se aspira in un stanza sporca, la pulisce … ma talvolta pulisce anche una stanza adiacente*
  - *Se aspira in una stanza pulita, a volte rilascia sporco*
- Variazioni necessarie al modello
  - Il modello di transizione restituisce un insieme di stati e l'agente non sa in quale sarà
  - Il piano sarà un piano di contingenza (condizionale)

Esempio



- Esempio

Risultato(Aspira, 1) = {5, 7}

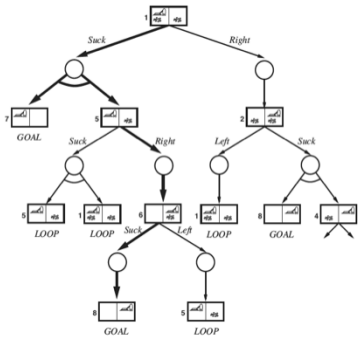
- Piano possibile

[Aspira,
  if stato=5
    then [Destra, Aspira]
    else [ ]
]

Alberi di ricerca AND-OR

- Nodi OR le scelte dell'agente/nodi AND le diverse contingenze (scelte dell'ambiente), da considerare tutte
- Una soluzione a un problema di ricerca AND-OR è un albero:
  - Con nodo obiettivo in ogni foglia
  - Specifica un'unica azione nei nodi OR
  - Include tutti gli archi uscenti da nodi AND

Esempio di ricerca AND-OR



Piano: [*Aspira*, ***if*** Stato=*5* ***then*** [*Destra, Aspira*] ***else*** [ ]]

Algoritmo ricerca grafi AND-OR

function Ricerca-Grafo-AND-OR (*problema)*
  returns un piano condizionale oppure *fallimento*
  Ricerca-OR(*problema.StatoIniziale, problema,* [ ])

function Ricerca-OR(*stato, problema, cammino*) // nodi OR
  returns un piano condizionale oppure *fallimento*
If *problema.*TestObiettivo(*stato*) **then return** [ ]
If *stato* è su *cammino* **then return** *fallimento*
for each *azione* **in** *problema.Azione(stato)* **do**
  piano ← Ricerca-AND (*Risultati*(*stato, azione*), *problema,* [*cammino*|*piano*])
  If *piano* ≠ *fallimento* **then return** [*azione* | *piano*]
return *fallimento*

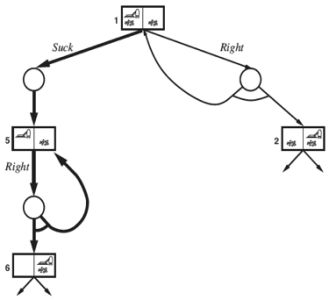
Algoritmo ricerca grafi AND-OR

function Ricerca-AND(*stati, problema, cammino*) // nodi AND
  returns un piano condizionale oppure *fallimento*
for each s_i **in** *stati* **do**
  $piano_i$ ← Ricerca-OR(s_i, *problema, cammino*)
  If $piano_i$ = *fallimento* **then return** *fallimento*
return
  [**if** s_1 **then** $piano_1$ **else**
    if s_2 **then** $piano_2$ **else**
      …
        if s_{n-1} **then** $piano_{n-1}$ **else** $piano_n$]

Azioni non deterministiche *L'aspirapolvere slittante*

- Comportamento:
  - *Quando si sposta può scivolare e rimanere nella stessa stanza*
  - Es. Risultato(Destra, 1) = {1, 2}
- Variazioni necessarie
  - Continuare a provare …
  - Il piano sarà un piano di contingenza con cicli

Aspirapolvere slittante: soluzione



Piano: [*Aspira*, L_1: *Destra*, **if** Stato=*5* **then** L_1 **else** *Aspira*]

Ricerca con osservazioni parziali

- Le percezioni non sono sufficienti a determinare lo stato esatto, anche se l'ambiente è deterministico.
- Stato credenza: un insieme di stati possibili in base alle conoscenze dell'agente
- Problemi senza sensori (*sensorless* o conformanti)
- Si possono trovare soluzioni anche senza affidarsi ai sensori utilizzando stati-credenza

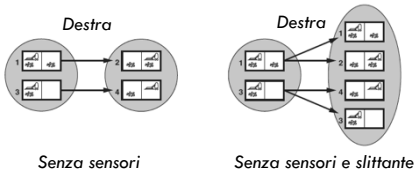
Aspirapolvere senza sensori

- L'aspirapolvere non percepisce la sua locazione, né se la stanza è sporca o pulita, ma conosce la geografia del suo mondo.
- Inizialmente tutti gli stati sono possibili
  - Stato iniziale = {1, 2, 3, 4, 5, 6, 7, 8}
- Le azioni riducono gli stati credenza
- Nello spazio degli stati credenza l'ambiente è osservabile (l'agente conosce le sue credenze)

Formulazione di problemi con stati-credenza

- Se N numero stati , 2^N sono i possibili stati credenza
- Stato-credenza iniziale $SC_0 \subseteq$ insieme di tutti gli N stati
- Azioni(b) = unione delle azioni *lecite* negli stati in b (ma se azioni illecite in uno stato hanno effetti dannosi meglio intersezione)
- Modello di transizione: gli stati risultanti sono quelli ottenibili applicando le azioni a uno stato qualsiasi (l' unione degli stati ottenibili dai diversi stati con le azioni eseguibili)

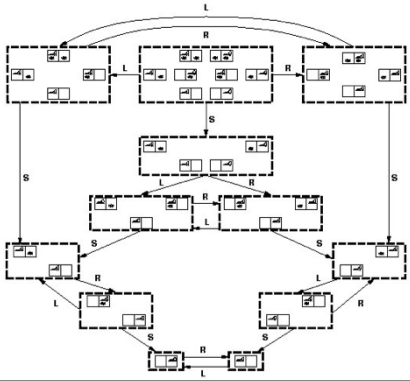
Problemi con stati-credenza (cnt.)



Senza sensori *Senza sensori e slittante*

- Test obiettivo: tutti gli stati nello stato credenza devono soddisfarlo
- Costo di cammino: il costo di eseguire un'azione potrebbe dipendere dallo stato, ma assumiamo di no

Il mondo dell'aspirapolvere senza sensori



Ricerca della soluzione

- Gli stati credenza possibili sono 2^8=256 ma solo 12 sono raggiungibili
- Si può effettuare un Ricerca-Grafo e controllare, generando s, se si è già incontrato uno stato credenza $s'=s$
- Si può fare anche di più nel "potare" …
  1. Se $s' \subseteq s$ (s' già incontrato) si può trascurare s
  2. Se $s \subseteq s'$ e da s' si è trovata una soluzione si può trascurare s

Problemi con spazi credenza

- Efficienza
  - Lo spazio degli stati può essere molto più grande
  - Ma ancora di più dimensione di uno spazio credenza può essere molto grande con la rappresentazione atomica. Non così con una rappresentazione più strutturata
- Soluzione incrementale
  - Si cerca una soluzione per stato 1 e poi si controlla che funzioni per 2 e i successivi; se no se ne cerca un'altra …
  - Scopre presto i fallimenti ma cerca un'unica soluzione che va bene per tutti gli stati

Ricerca con osservazioni

- Problema parzialmente osservabile
- Esempio: *l'aspirapolvere con sensori locali che percepisce la sua posizione e lo sporco nella stanza in cui si trova*
- Le percezioni diventano importanti

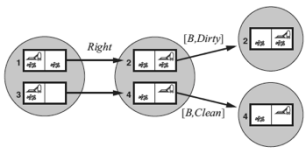
Ricerca con osservazioni parziali

- Le percezioni assumono un ruolo
  - Percezioni(s) = *null* in problemi *sensorless*
  - Percezioni(s) = s ambienti osservabili
  - Percezioni(s) = percezioni nello stato *s*
  - Esempio: [A, *Sporco*] percezione stato iniziale
- Le percezioni restringono l' insieme di stati possibili
  - Esempio: Stato iniziale = {1, 3}

Il modello di transizione si complica

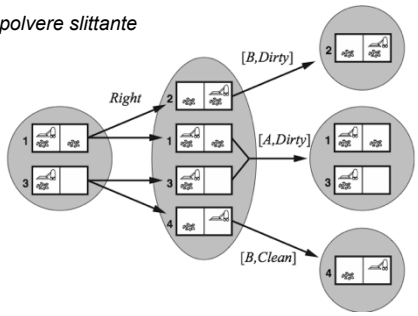
La transizione avviene in tre fasi:

1. Predizione dello stato credenza: *Predizione*(b, a)=b'
2. Predizione dell'osservazione: *Percezioni-possibili*(b')
3. Calcolo nuovi stato credenza (insieme di stati compatibili con la percezione):
   b'' = *Aggiorna*(Predizione(b, a), o) data l'osservazione in b'

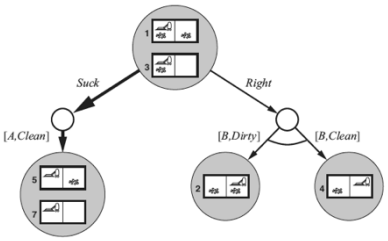


Transizione con azioni non deterministiche

Aspirapolvere slittante



Aspirapolvere con sensori locali



[*Aspira*, *Destra*, **if** statoCredenza = {6} **then** *Aspira* **else** []]

Ricerca *online*

- Ricerca *offline* e ricerca *online*
- L'agente alterna pianificazione e azione

1. Utile in ambienti dinamici o semidinamici
   - Non c'è troppo tempo per pianificare
2. Utile in ambienti non deterministici
   1. Pianificare vs agire
3. Necessaria per ambienti ignoti tipici dei problemi di esplorazione

Problemi di esplorazione

- I problemi di esplorazione sono casi estremi di problemi con contingenza in cui l'agente deve anche pianificare azioni esplorative
- Assunzioni per un problema di esplorazione:
  - Solo lo stato corrente è osservabile, l'ambiente è ignoto
  - Non si conosce l'effetto delle azioni e il loro costo
  - Gli stati futuri e le azioni che saranno possibili non sono conosciute a priori
  - Si devono compiere azioni esplorative come parte della risoluzione del problema
- Il labirinto come esempio tipico

Assunzioni

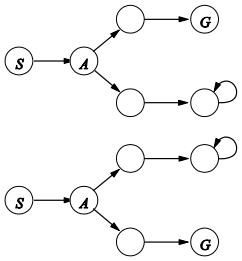
- Cosa conosce un agente *online* in *s* …
  - Le azioni legali nello stato attuale *s*
  - Risultato(*s*, *a*) ma deve eseguire *a*
  - Il costo della mossa c(*s*, *a*, *s'*), solo dopo aver eseguito a
  - Goal-test(s)
  - La stima della distanza: dal goal: h(s)

Costo soluzione

- Il costo del cammino è quello effettivamente percorso
- Il rapporti tra questo costo e quello ideale (conoscendo l'ambiente) è chiamato rapporto di competitività
- Tale rapporto può essere infinito
- Le prestazioni sono in funzione dello spazio degli stati

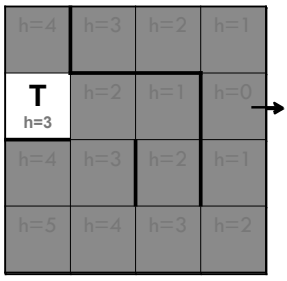
Assunzione ulteriore

- Ambienti esplorabili in maniera sicura
  - non esistono azioni irreversibili
  - lo stato obiettivo può sempre essere raggiunto
  - diversamente non si può garantire una soluzione



Esempio: Teseo con mappa e senza

- Con mappa
  - applicabili tutti gli algoritmi di pianificazione visti
- Senza mappa
  - l'agente non può pianificare può solo esplorare nel modo più razionale possibile
  - Ricerca online

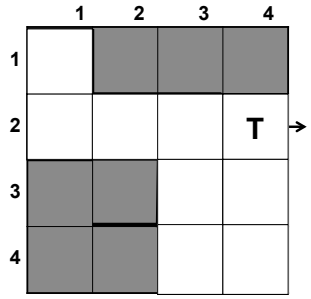


Ricerca in profondità *online*

- Gli agenti *online* ad ogni passo decidono l'azione da fare (non il piano) e la eseguono.
- Ricerca in profondità *online*
  - Esplorazione sistematica delle alternative
    - *nonProvate*[s] mosse ancora da esplorare in s
  - È necessario ricordarsi ciò che si è scoperto
    - *Risultato*[*a*, *s*] = *s'*
  - Il backtracking significa tornare sui propri passi
    - *backtrack*[s] stati a cui si può tornare

Esempio

- Sceglie il primo tra (1,1) e (2,2)
- In (1, 1) ha solo l'azione per tornare indietro
- …
- Nella peggiore delle ipotesi esplora ogni casella due volte



Algoritmo in profondità *online*

function Agente-Online-DFS(*s*) returns un'azione
  static: *Risultato*, *nonProvate*, *backtrack*,
    s- (*stato precedente*), a- (*ultima azione*)
if Goal-Test(*s*) **then return** stop
if s è un nuovo stato **then** *nonProvate*[*s*] ← Azioni(*s*)
if s- *non è null* **then** risultato[s-, a-] ← s; *backtrack*[*s*] ← s-;
if *nonProvate*[*s*] *vuoto* **then**
  if *backtrack*[s] *vuoto* **then return** stop
  else *a* ← *azione per tornare in* POP(*backtrack*[*s*])
else *a* ← POP(*nonProvate*[*s*])
s- ← s; return *a*

Ricerca euristica *online*

- Nella ricerca online si conosce il valore della funzione euristica una volta esplorato lo stato.
- Un algoritmo di tipo Best First non funzionerebbe.
- Serve un metodo locale
- *Hill-climbing* con *random-restart* non praticabile
- Come sfuggire a minimi locali?

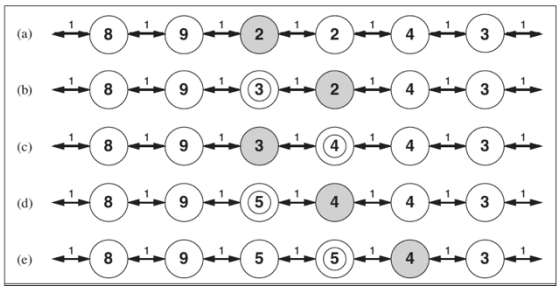
Due soluzioni

1. Random-walk
   - si fanno mosse casuali in discesa
2. Apprendimento Real-Time:
   - esplorando si aggiustano i valori dell'euristica per renderli più realistici

Idea dell'algoritmo LRTA*

- H(s): migliore stima trovata fin qui
- Si valutano i successori:
  Costo-LRTA*(s, a, s', H) =
    h(s) *se s' indefinito* (non esplorato)
    H(s') + costo(s,a,s') *altrimenti*
- Ci si sposta sul successore di Costo-LRTA* minore
- Si aggiorna la H dello stato da cui si proviene

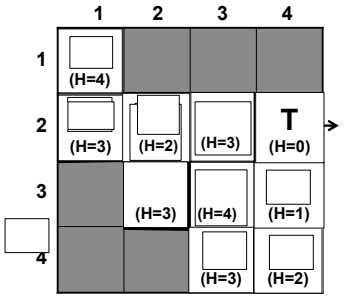
LRTA* supera i minimi locali



LRTA*

function Agente-LRTA*(s) **returns** un'azione
static: risultato, H, s-, a-
  if Goal-Test(s) **then return** stop
  if s *nuovo (non in H)* **then** H[s] ← h[s]
1. if s- non è null *//si aggiusta il costo H del predecessore*
    risultato[a-, s-] ← s
    H[s-] ← *min* Costo-LRTA*(s-, b, risultato[b, s-], H)
2. a ← *un'azione b tale che* minimizza b ∈ Azioni(s)
    Costo-LRTA*(s, b, risultato[b, s], H)
s- ← s; return a

Esempio di LRTA*



Considerazioni su LRTA*

- LRTA* cerca di simulare A* con un metodo locale: tiene conto del costo delle mosse come può aggiornando al volo la H
- Completo in spazi esplorabili in maniera sicura
- Nel caso pessimo visita tutti gli stati due volte ma è mediamente più efficiente della profondità *online*
- Non ottimale, a meno di usare una euristica perfetta (non basta una $f=g+h$ con h ammissibile)