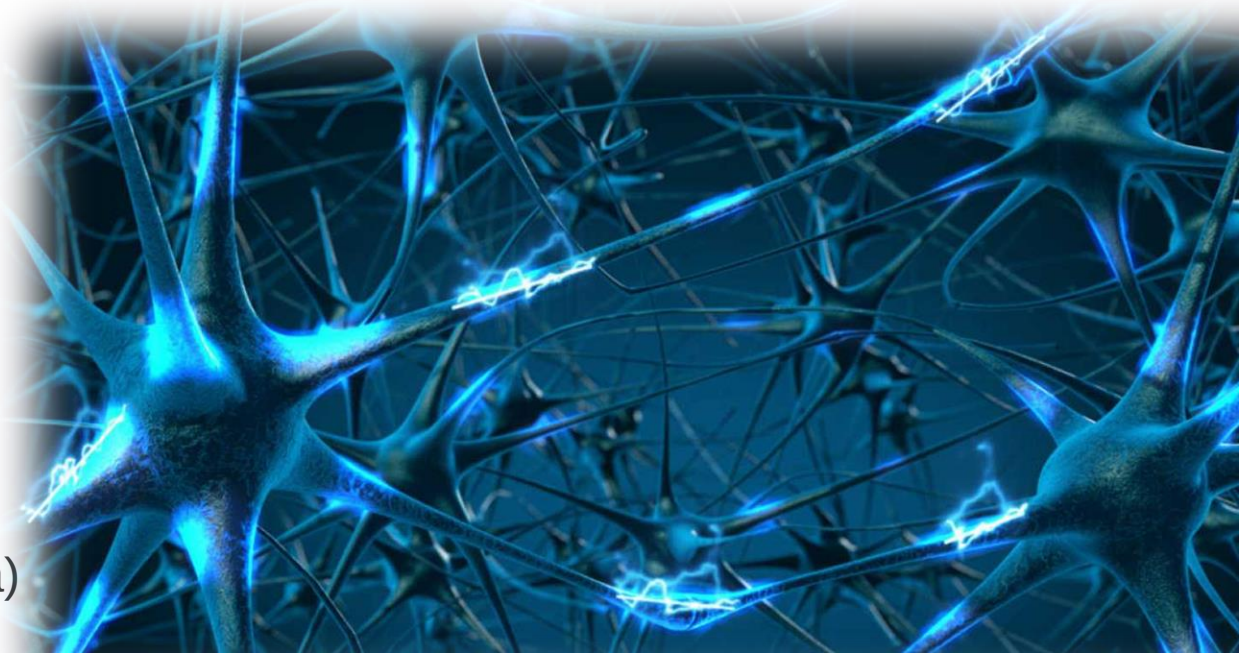
INTELLIGENZA ARTIFICIALE

ALGORITMI AVANZATI DI RICERCA (*)

Corsi di Laurea in Informatica, Ing. Gestionale, Ing. Informatica, Ing. di Internet

(a.a. 2021-2022)

Roberto Basili



(*) alcune *slides* sono di
Maria Simi (Univ. Pisa)

Overview

- Local Search (4.1.1, 4.1.2, 4.1.2)
- Ricerca in Spazi pazialmente osservabili
- On-Line Search (4.1.3)

Assunzioni fatte

- Gli agenti *risolutori di problemi* “classici” assumono:
  - Ambienti completamente osservabili
  - Ambienti deterministici
  - il piano generato è una sequenza di azioni che può essere generato *offline* ed eseguito senza imprevisti

Verso ambienti più realistici

- La ricerca sistematica o euristica nello spazio di stati è troppo costosa
  - Metodi di ricerca locale
- Assunzioni da riconsiderare
  - Azioni non deterministiche e ambiente parzialmente osservabile
    - Piani condizionali, ricerca AND-OR, stati credenza
  - Ambienti sconosciuti e problemi di esplorazione
    - Ricerca online

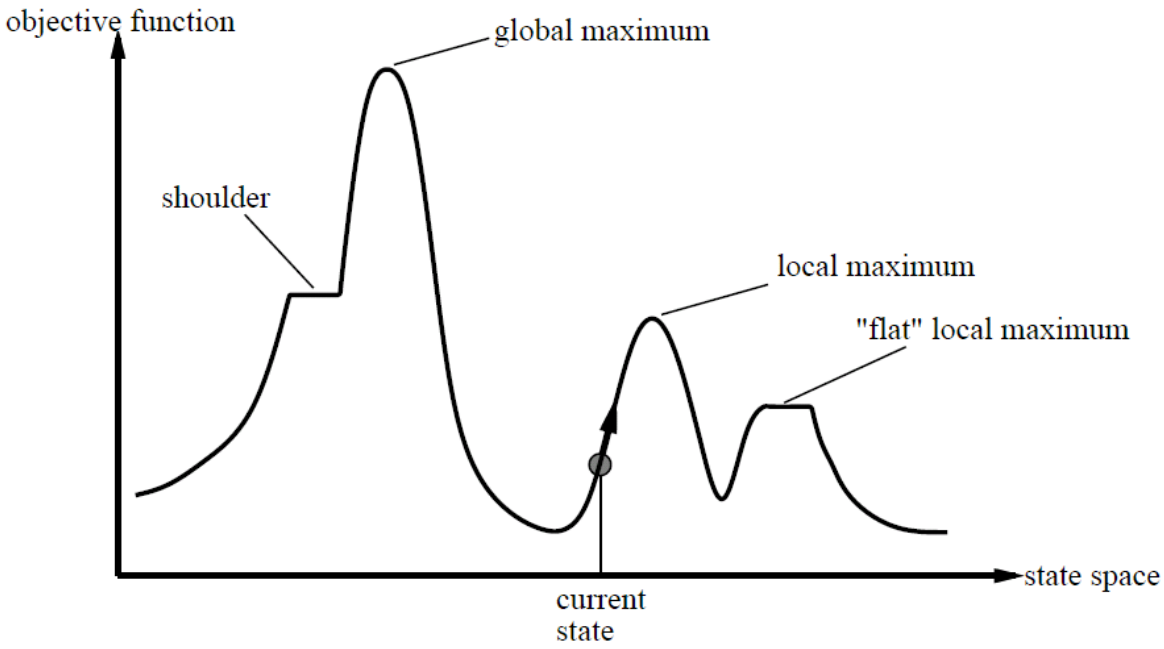
Assunzioni sui problemi

- Gli algoritmi visti esplorano gli spazi di ricerca alla ricerca di un goal e restituiscono un *cammino soluzione*
- Ma a volte lo stato goal è la soluzione del problema.
- Gli algoritmi di *ricerca locale* sono adatti per problemi in cui:
  - La sequenza di azioni non è importante: quello che conta è unicamente lo stato goal
  - Tutti gli elementi della soluzione sono nello stato ma alcuni vincoli sono violati.
    - Es. le regine nella formulazione a stato completo, quando tutte le regine si considerano posizionate anche se non rispettano i vincoli

Algoritmi di ricerca locale

- Non sono sistematici
- Tengono traccia solo del nodo corrente e si spostano su nodi adiacenti
- Non tengono traccia dei cammini
- Efficienti in occupazione di memoria
- Utili per risolvere problemi di ottimizzazione
  - *lo stato migliore secondo una funzione obiettivo*
  - *lo stato di costo minore*

Panorama dello spazio degli stati



- Uno stato ha una posizione sulla superficie e una altezza che corrisponde al valore della f. di valutazione
- Un algoritmo provoca movimento sulla superficie
- Trovare l'avvallamento più basso o il picco più alto

Ricerca in salita (*Hill climbing*)

- Ricerca locale *greedy*
- Vengono generati i successori e valutati; viene scelto un nodo che migliora la valutazione dello stato attuale (non si tiene traccia degli altri):
  - il migliore (salita rapida) ➔ Hill climbing a salita rapida
  - uno a caso ➔ Hill climbing stocastico
  - il primo ➔ Hill climbing con prima scelta
- Se non ci sono stati successori migliori l'algoritmo termina con fallimento

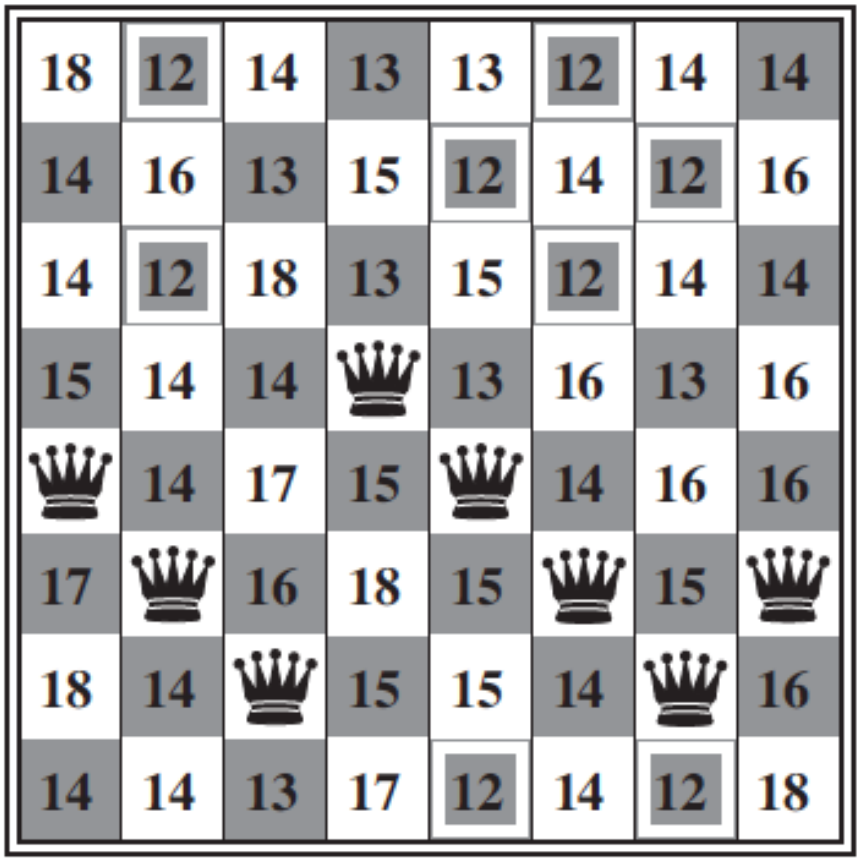
L'algoritmo Hill climbing

```
function Hill-climbing (problema)
    returns uno stato che è un massimo locale
    nodo-corrente = CreaNodo(problema.Stato-iniziale)
    loop do
            vicino = il successore di nodo-corrente di valore più alto
            if vicino.Valore ≤ nodo-corrente.Valore then
                    return nodo-corrente.Stato   // interrompe la ricerca
             nodo-corrente = vicino
```

- Nota: si prosegue solo se il vicino è migliore dello stato corrente

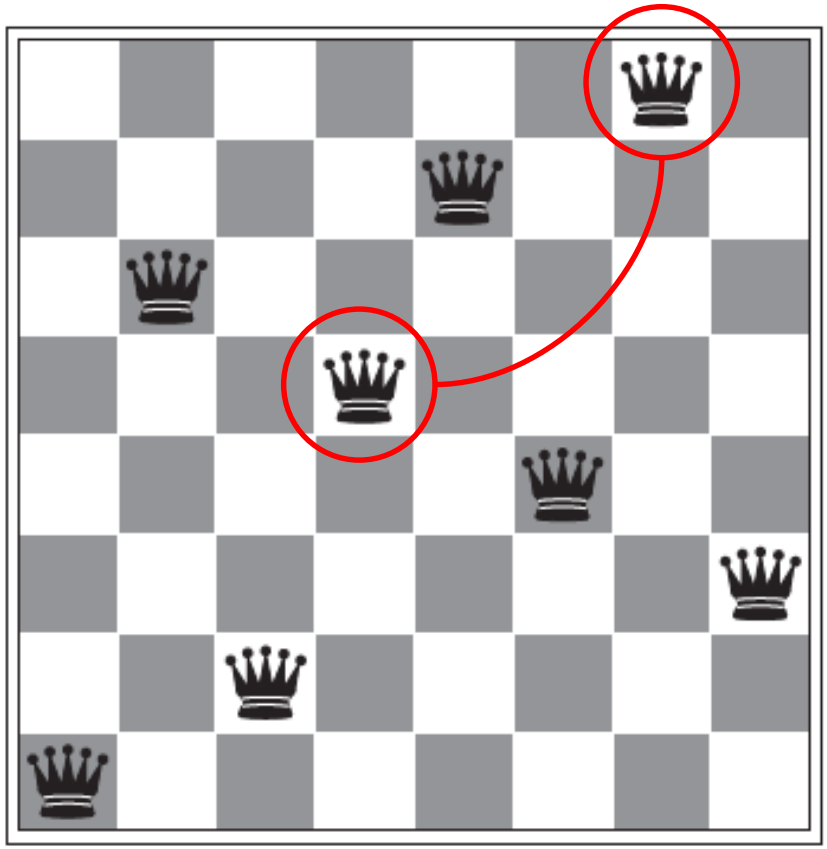
Il problema delle 8 regine

- *h: numero di coppie di regine che si attaccano a vicenda* (nell'es. a dx h(s)=17)
- I numeri sono i valori dei successori (7x8)
- Tra i migliori (valore 12) si sceglie a caso



Esempio di configurazione che è un massimo locale

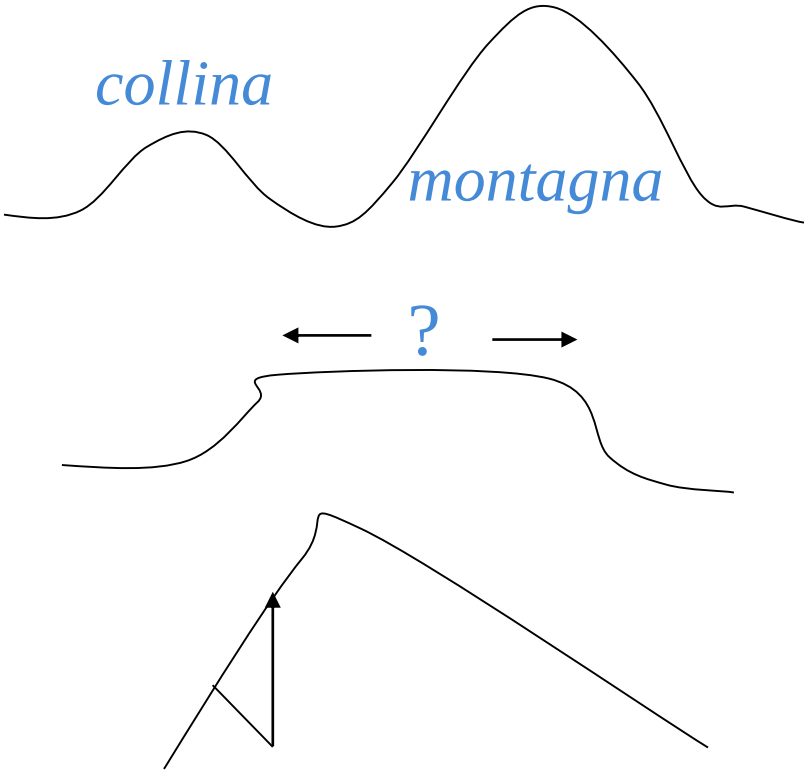
- *h* = 1 (*valore minimo*)
- Tutti gli stati successori peggiorano la situazione
- Hill-climbing si blocca
- Per le 8 regine questo succede l’86% delle volte
- In media 4 passi



Problemi con Hill-climbing

Se la f. è da ottimizzare i picchi sono massimi locali o soluzioni ottimali

- Massimi locali
- Altipiani (*plateaux*)
- Crinali (o creste)



Miglioramenti

1. Consentire un numero limitato di mosse *laterali*
   - L'algoritmo sulle 8 regine ha successo nel 94%, ma impiega in media 21 passi

2. Hill-climbing stocastico: si sceglie a caso tra le mosse in salita (magari tenendo conto della pendenza)
   - converge più lentamente ma a volte trova soluzioni migliori

3. Hill-climbing con prima scelta
   - può generare le mosse a caso fino a trovarne una migliore
   - più efficace quando i successori sono molti

Miglioramenti (cont.)

4. Hill-Climbing con *riavvio casuale* (*random restart*): ripartire da un punto scelto a caso
   - Se la probabilità di successo è *p* saranno necessarie in media 1/p ripartenze per trovare la soluzione (es. 8 regine, *p*=0.14, 7 iterazioni)
   - Hill-climbing con random-restart è tendenzialmente completo (basta insistere)
   - Per le regine: 3 milioni in meno di un minuto!
   - Se funziona o no dipende molto dalla forma del panorama degli stati

Simulated Annealing (Tempra simulate)

- L’ algoritmo di tempra simulata (Simulated annealing) [Kirkpatrick, Gelatt, Vecchi 1983] combina hill-climbing con una scelta stocastica (ma non del tutto casuale, perché poco efficiente…)
- Analogia con il processo di tempra dei metalli in metallurgia

Simulated Annealing

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(*problem*.INITIAL-STATE)
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow next.\text{VALUE} - current.\text{VALUE}$
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$

Figure 4.5 The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. Downhill moves are accepted readily early in the annealing schedule and then less often as time goes on. The *schedule* input determines the value of the temperature T as a function of time.

Tempra simulata

- Ad ogni passo si sceglie un successore a caso:
  - se migliora lo stato corrente viene espanso
  - se no (caso in cui $\Delta E = f(n') - f(n) < 0$) quel nodo viene scelto con probabilità $p = e^{\Delta E / T}$ $[0 \leq p \leq 1]$

    [Si genera un numero casuale tra *0* e *1*:
    se questo è $< p$ il successore viene scelto,
    altrimenti no]
  - *p* è inversamente proporzionale al peggioramento
  - *T* descresce col progredire dell'algoritmo (quindi anche *p*) secondo un piano definito

Tempra simulata: analisi

- La probabilità di una mossa in discesa diminuisce col tempo e l'algoritmo si comporta sempre di più come Hill Climbing.
- Se T viene decrementato abbastanza lentamente siamo sicuri di raggiungere la soluzione ottimale.
- Analogia col processo di tempra dei metalli
  - T corrisponde alla temperatura
  - ΔE alla variazione di energia

Tempra simulata: parametri

- Valore iniziale e decremento di T sono parametri.
- Valori per T determinati sperimentalmente: il valore iniziale di T è tale che per valori medi di ΔE, $p=e^{\Delta E/T}$ sia all'incirca 0.5

Ricerca *local beam*

- Si tiene traccia di *k* stati anziché uno solo
- Ad ogni passo si generano i successori di tutti i *k* stati
  - Se si trova un goal ci si ferma
  - Altrimenti si prosegue con i *k* migliori tra questi

Beam search stocastica

- Nella variante stocastica della *local beam,* si scelgono *k* successori a caso con probabilità maggiore per i migliori.
- … come in un processo di selezione naturale
  - *organismo, progenie, fitness*
- Algoritmi genetici: varianti della beam search stocastica in cui gli stati successori sono ottenuti combinando due stati padre (anziché per evoluzione)

RICERCA *ONLINE*

CAP 4 – *OLTRE LA RICERCA CLASSICA*

dalle slide di Maria Simi (Uni. Pisa)

Ambienti più realistici

- Gli agenti *risolutori di problemi* “classici” assumono:
  - Ambienti completamente osservabili e deterministici
  - il piano generato è una sequenza di azioni che può essere generata *offline ed* eseguita senza imprevisti
  - Le percezioni non servono se non nello stato iniziale

Soluzioni più complesse

- In un ambiente parzialmente osservabile e non deterministico le percezioni sono importanti
  - restringono gli stati possibili (*vincoli provenienti dall'ambiente*)
  - informano sull'effetto dell'azione (*esperienza*)

- Più che un *piano* l'agente può elaborare una *strategia,* che tiene conto delle diverse eventualità: un piano con contingenza

- L'aspirapolvere con assunzioni diverse
  - Vediamo prima il non determinismo.

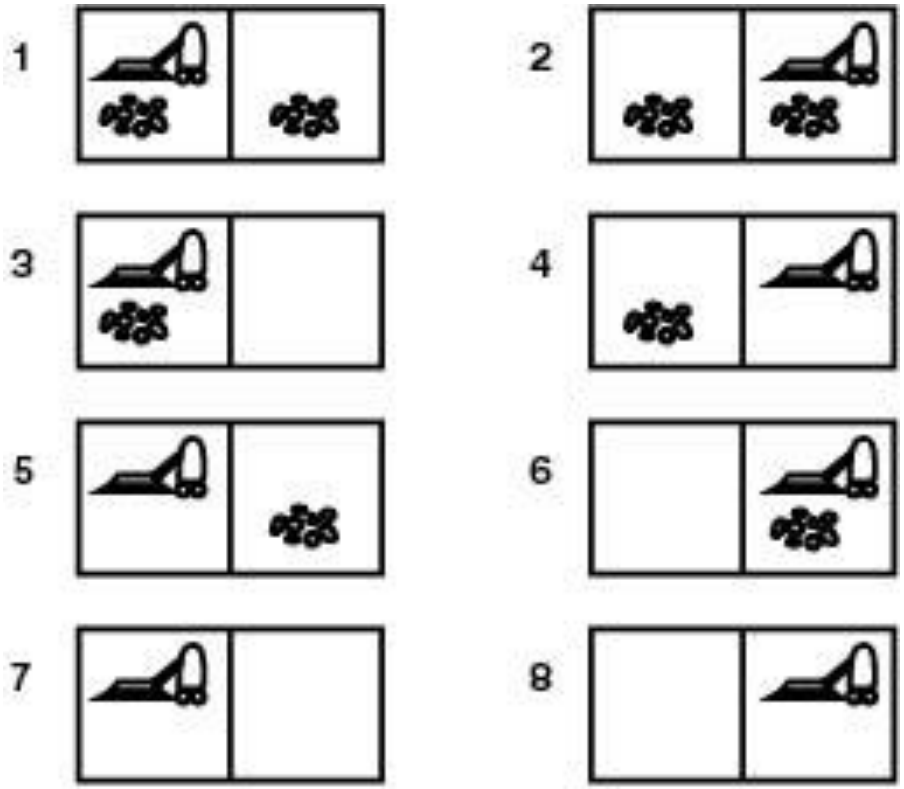
Azioni non deterministiche
L'aspirapolvere imprevedibile

- 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: l'agente non sa in quale finirà per trovarsi
  - Il piano di contingenza sarà un piano condizionale e magari con cicli

Esempio

Se aspira in un stanza sporca, la pulisce … ma talvolta pulisce anche una stanza adiacente



- Esempio

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

- Piano possibile

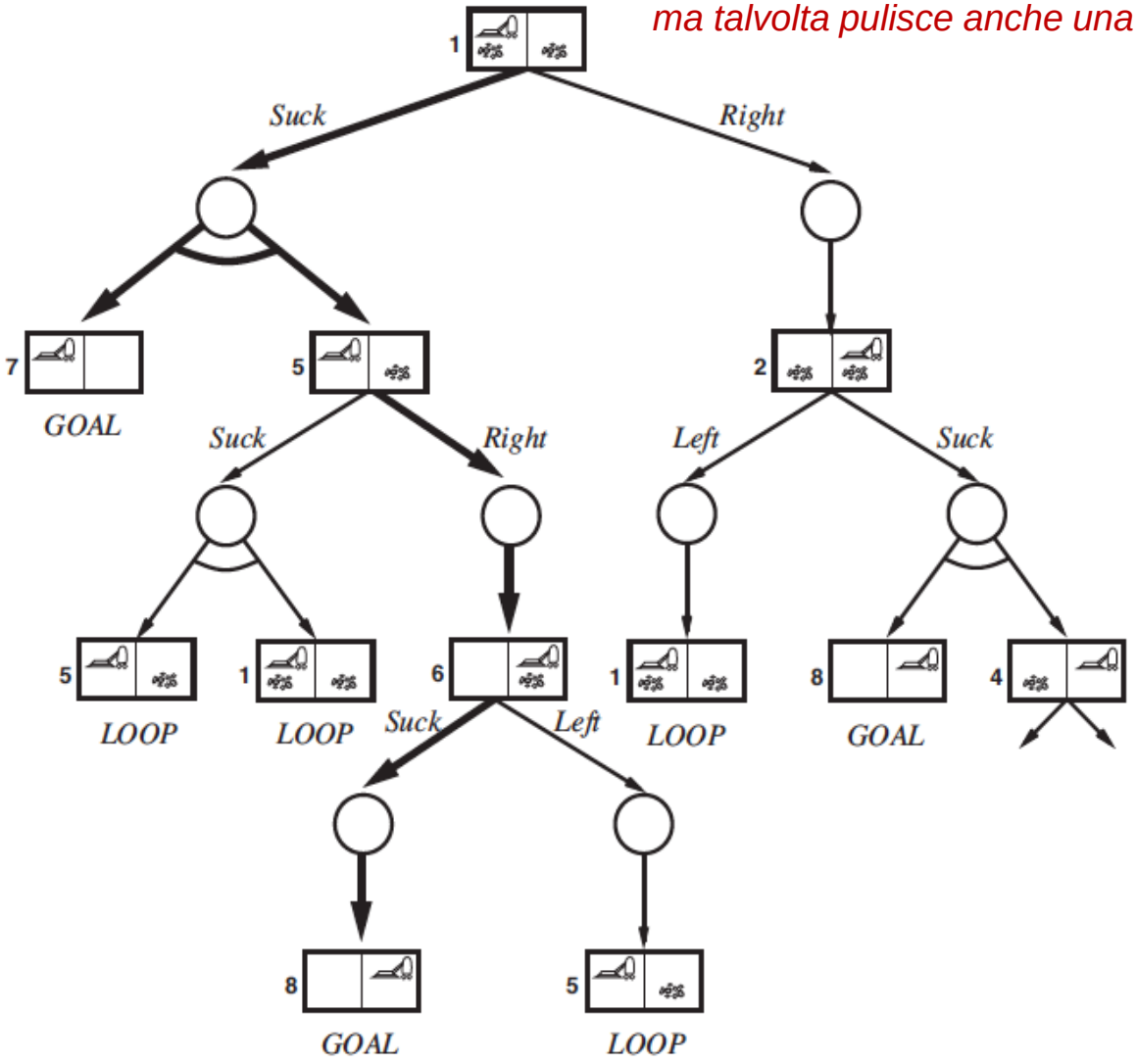
[Aspira,
  if stato=5
    then [Destra, Aspira]
    else [ ]
]

Come si pianifica: *alberi di ricerca AND-OR*

- Nodi OR le scelte *alternative* dell'agente
- Nodi AND le diverse contingenze (le *scelte* determinate dall'evoluzione dell'ambiente), da considerare *tutte*
- Una soluzione a un problema di ricerca AND-OR è un albero che:
  - ha un 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

Se aspira in un stanza sporca, la pulisce …
ma talvolta pulisce anche una stanza adiacente



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** [ ] // piano vuoto

If *stato* è su *cammino* **then return** *fallimento* // spezza i cicli

for each *azione* **in** *problema.Azione(stato)* **do**

piano ← Ricerca-AND (*Risultati*(*stato, azione*), *problema,* [*stato*|*cammino*])

If *piano* ≠ *fallimento* **then return** [*azione* | *piano*]

return *fallimento*

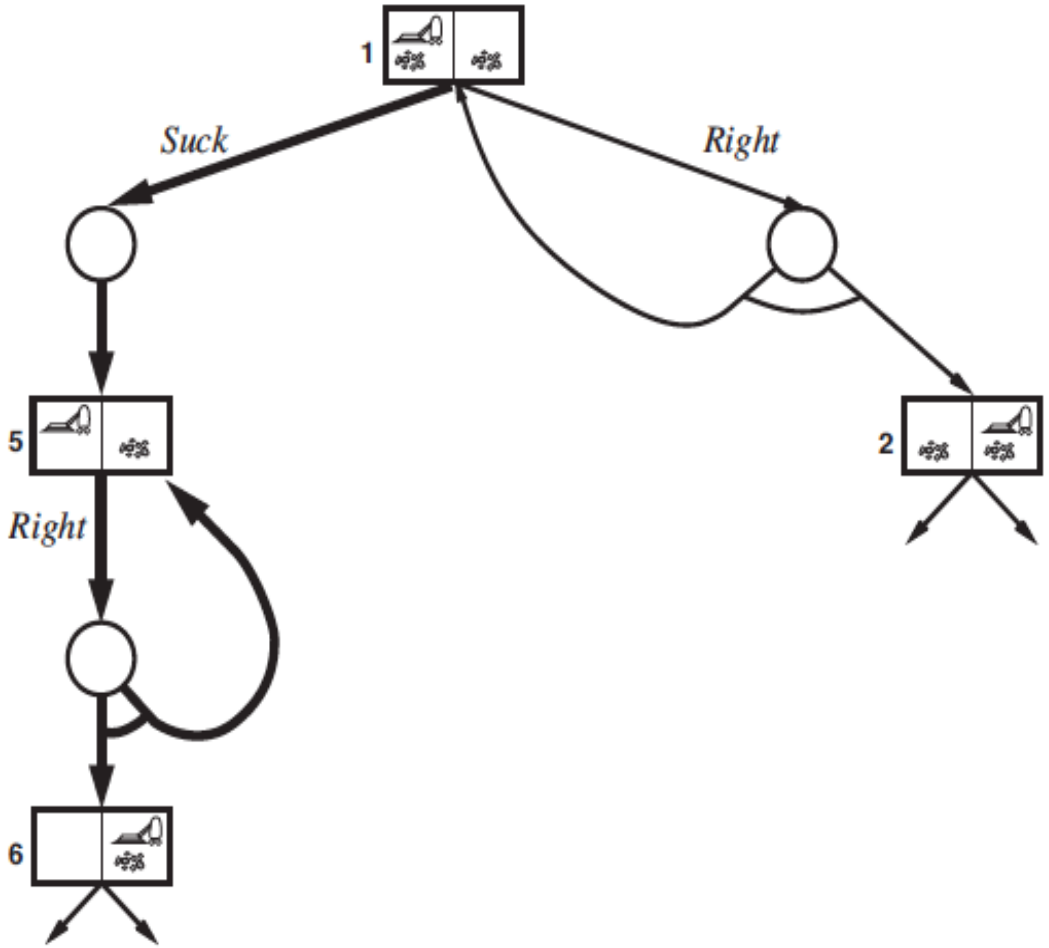
Algoritmo ricerca grafi AND-OR

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** $piano_n$]

Ancora azioni non deterministiche
L'aspirapolvere slittante

- Qui la azione non provoca alcun cambiamento di stato
- Comportamento:
  - *Quando si sposta può scivolare e rimanere nella stessa stanza*
  - Es. Risultato(Destra, 1) = {1, 2}
- Variazioni necessarie
  - Continuare a provare …
  - Il piano di contingenza potrà avere dei cicli

Aspirapolvere slittante: soluzione



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

[*Aspira*, L_1: *Destra*, while Stato=5 do L_1 …

Osservazione

- Bisogna distinguere tra:
  1. Osservabile e non deterministico
     (es. aspirapolvere slittante)
  2. Non osservabile e deterministico
     (es. non so se la chiave aprirà la porta)
- In questo secondo caso si può provare all' infinito ma niente cambierà!

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

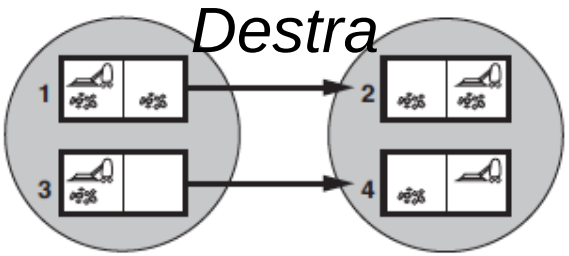
Ambiente non osservabile: *Aspirapolvere senza sensori*

- L'aspirapolvere:
  - non percepisce la sua locazione, né se la stanza è sporca o pulita ma …
  - conosce la geografia del suo mondo e l'effetto delle azioni
- Inizialmente (in generale) tutti gli stati sono possibili
  - Stato iniziale = {1, 2, 3, 4, 5, 6, 7, 8}
- Le azioni determinano gli stati credenza
- Nota: nello spazio degli stati credenza l'ambiente è osservabile per definizione (l'agente conosce le sue credenze)

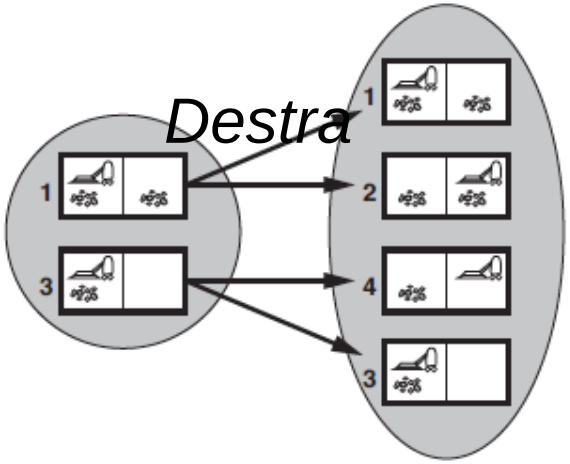
Formulazione di problemi con stati-credenza

- 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.)



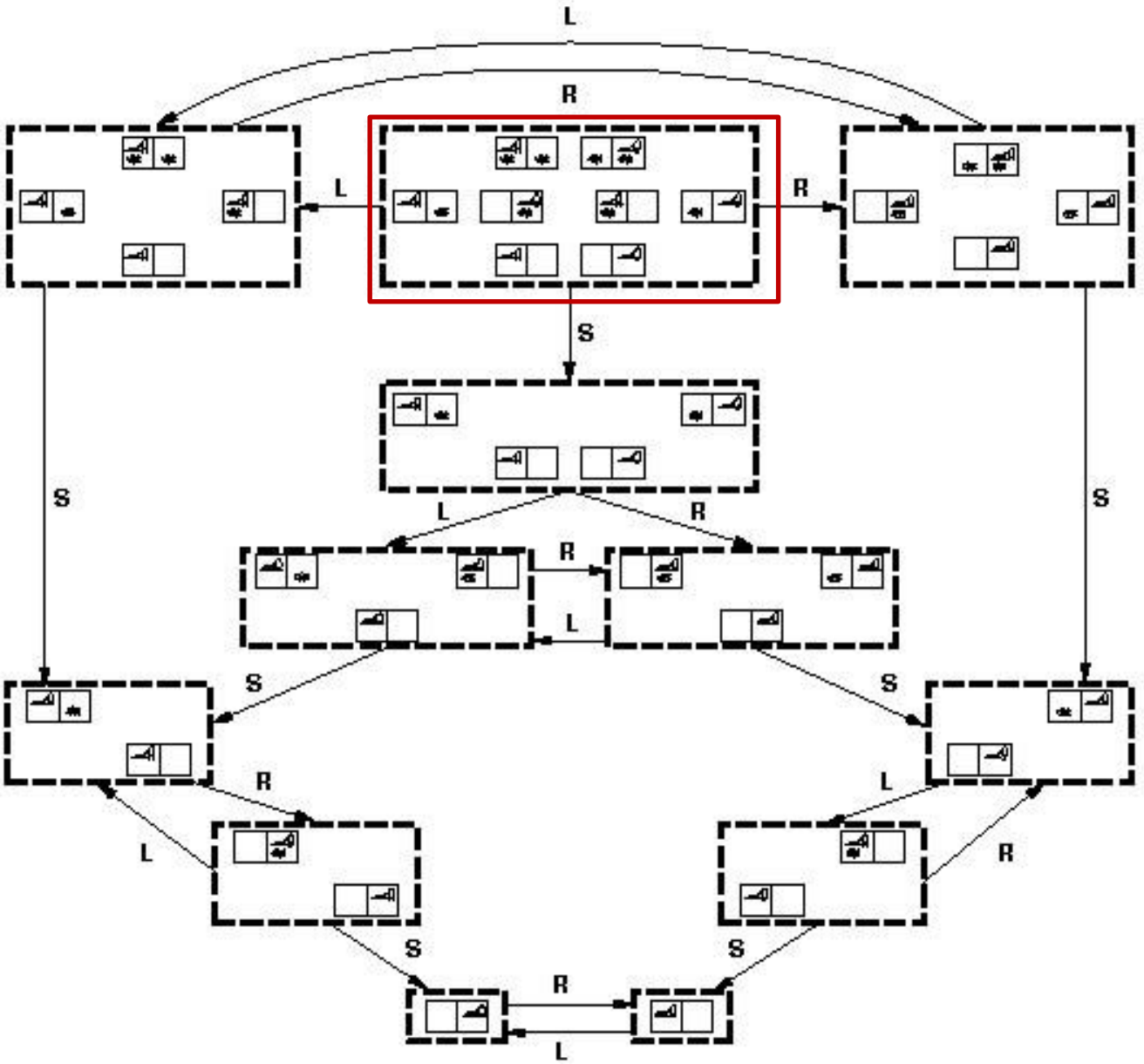
Senza sensori deterministico



Senza sensori e slittante (non det.)

- 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* e trascurare *s*
- Si può anche "potare" in modo più efficace …
  1. Se *s'* $\subseteq$ *s* (con *s'* stato credenza 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
  - La rappresentazione atomica obbliga a elencare tutti gli stati. Non è molto “compatta”. Non così con una rappresentazione più strutturata (lo vedremo)
- Soluzione incrementale
  - Dovendo trovare una soluzione per {1, 2, 3 …} 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 per 1 …
  - Scopre presto i fallimenti ma cerca un’unica soluzione che va bene per tutti gli stati

Ricerca con osservazioni

- Ambiente **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

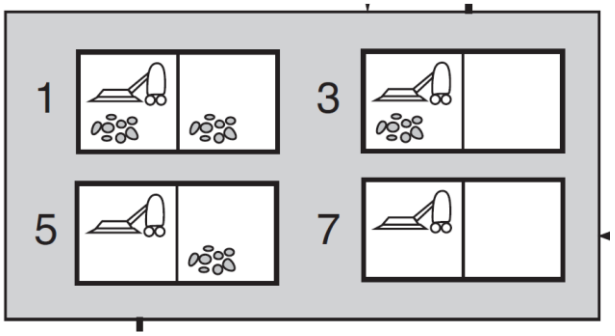
Ricerca con osservazioni parziali

- Le percezioni assumono un ruolo
  - Percezioni(s) = *null* in problemi *sensorless*
  - Percezioni(s) = s, ambienti osservabili
  - Percezioni(s) = percezioni [possibili] nello stato *s*

- Le percezioni restringono l'insieme di stati possibili

  Esempio: [A, *Sporco*] percezione stato iniziale

  Stato iniziale = {1, 3}

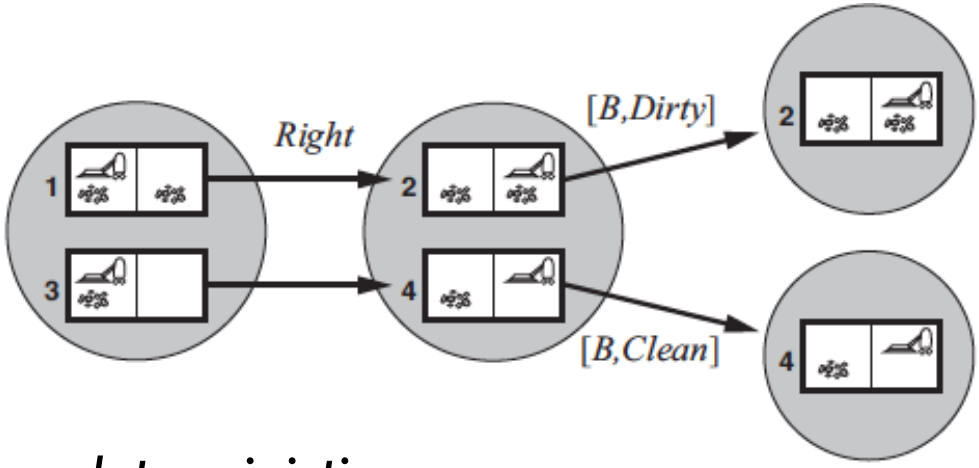


Il modello di transizione si complica

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):

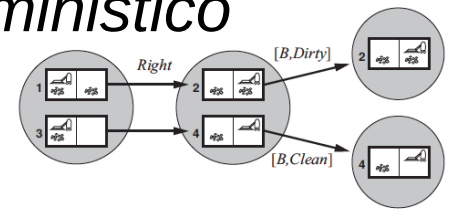
b’’ = *Aggiorna*(Predizione(b, a), o) data l’osservazione o in b’

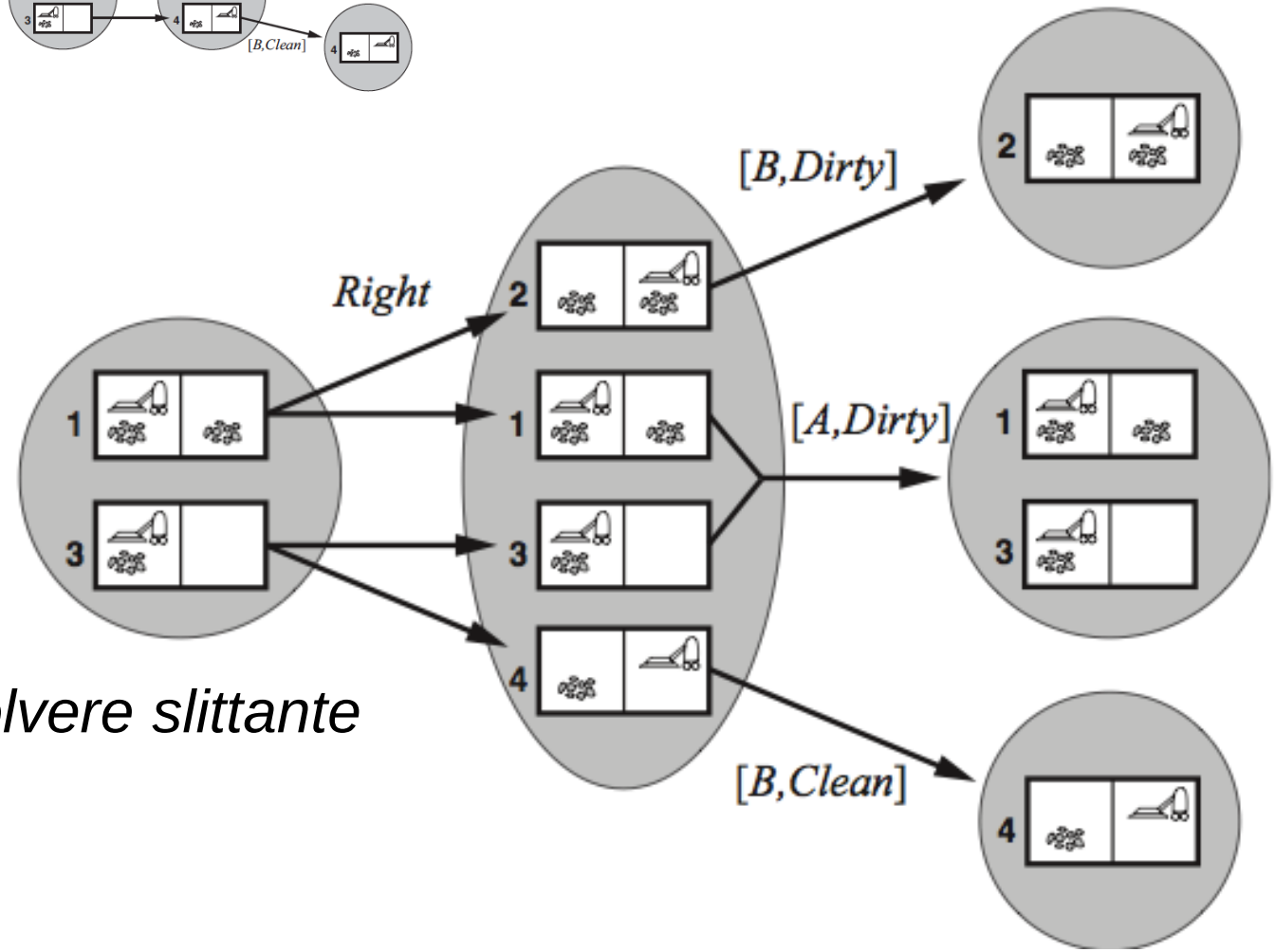


Caso deterministico

Transizione con azioni non deterministiche

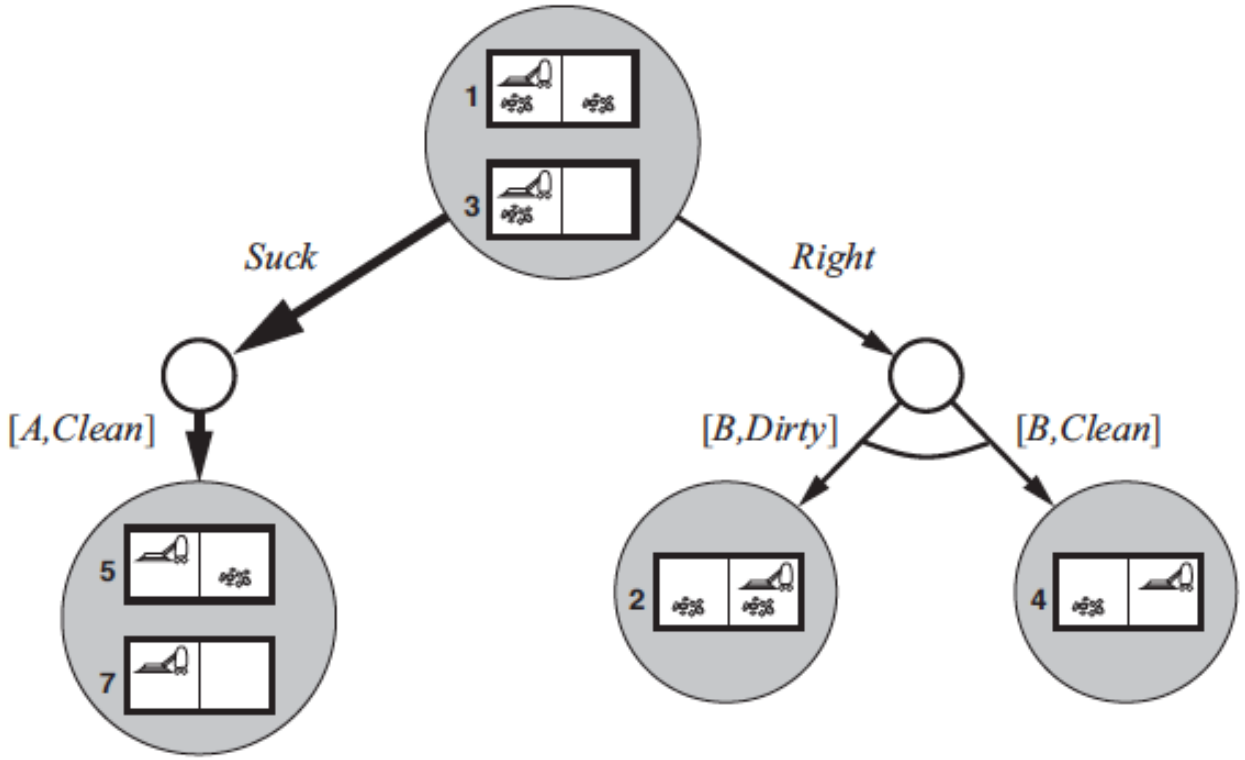
Caso deterministico



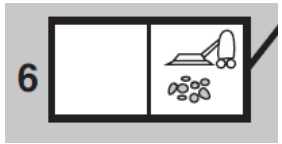


Aspirapolvere slittante

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

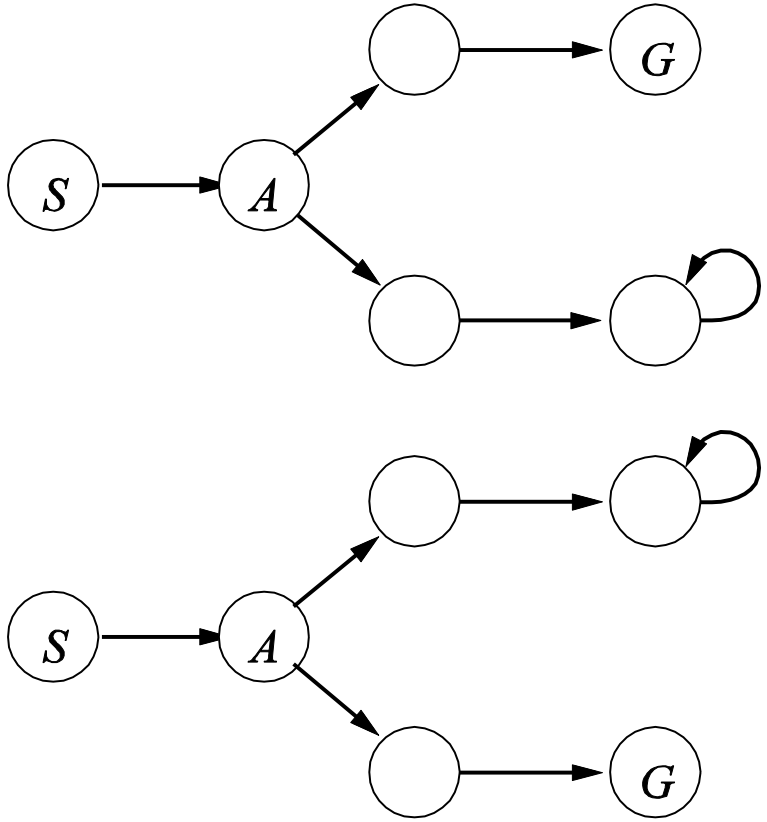
- Cosa conosce un agente *online* in *s* …
  - Le azioni legali nello stato attuale *s*
  - RISULTATO(*s, a*), ma dopo aver eseguito *a*
  - Il costo della mossa c(*s, a, s'*), solo dopo aver eseguito *a*
  - GOAL-TEST(*s*)
  - La stima della distanza: dal goal: *h*(*s*)

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

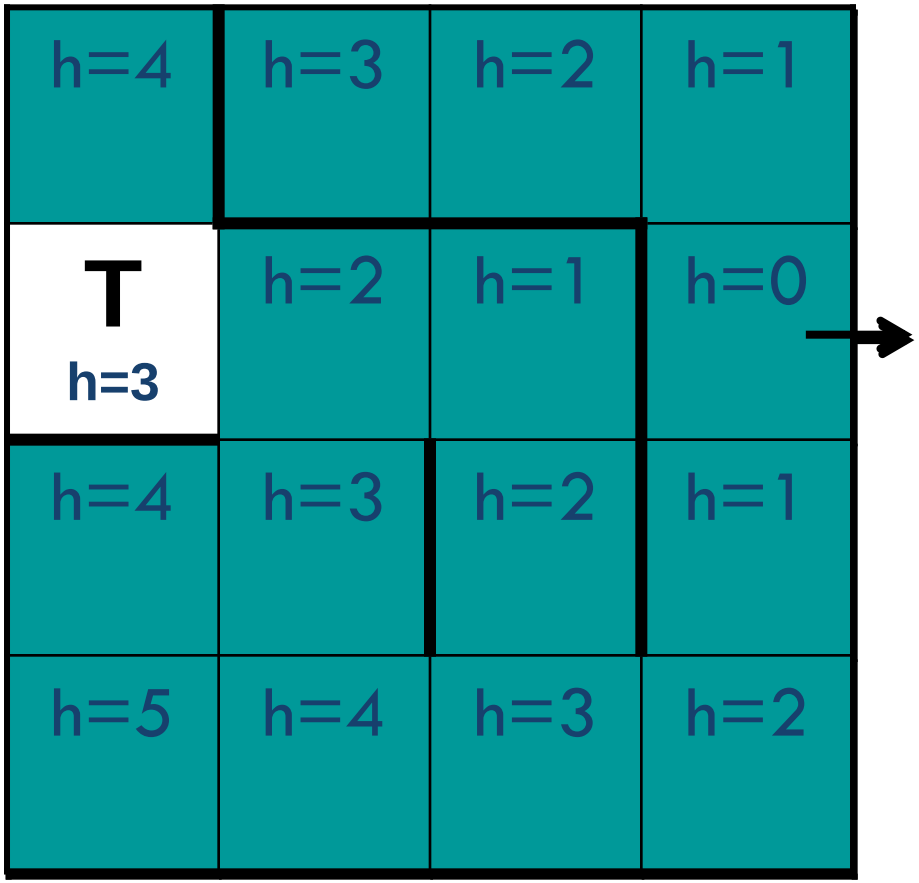
Assunzione ulteriore

- Ambienti esplorabili in maniera sicura solo se
  - 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

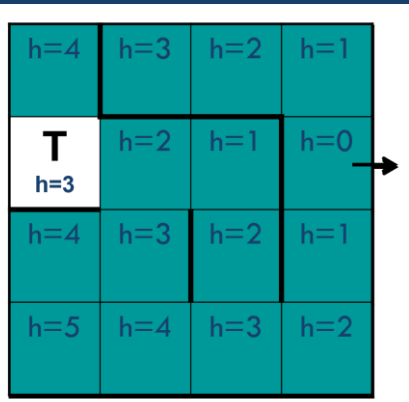


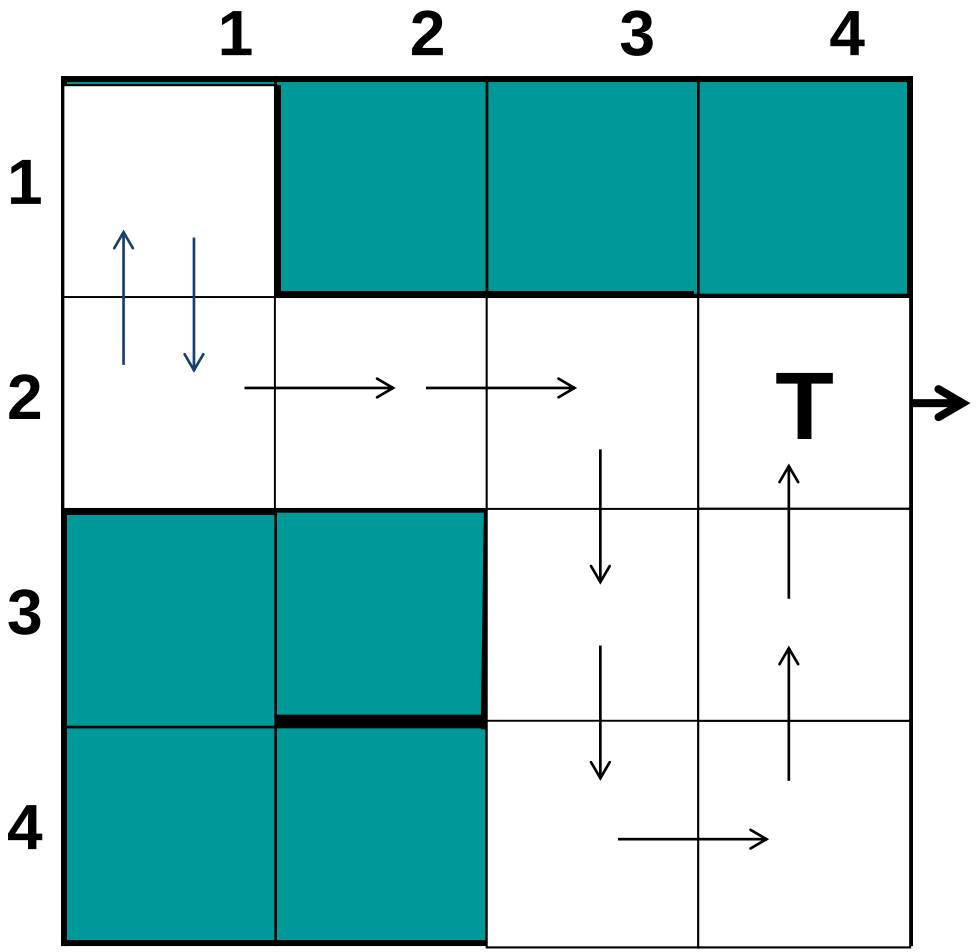
Ricerca in profondità *online*

- 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 ONLINE-DFS-AGENT(s') returns an action
  inputs: s', a percept that identifies the current state
  persistent: result, a table indexed by state and action, initially empty
              untried, a table that lists, for each state, the actions not yet tried
              unbacktracked, a table that lists, for each state, the backtracks not yet tried
              s, a, the previous state and action, initially null

  if GOAL-TEST(s') then return stop
  if s' is a new state (not in untried) then untried[s'] ← ACTIONS(s')
  if s is not null then
      result[s, a] ← s'
      add s to the front of unbacktracked[s']
  if untried[s'] is empty then
      if unbacktracked[s'] is empty then return stop
      else a ← an action b such that result[s', b] = POP(unbacktracked[s'])
  else a ← POP(untried[s'])
  s ← s'
  return a
```

Figure 4.21 An online search agent that uses depth-first exploration. The agent is applicable only in state spaces in which every action can be "undone" by some other action.

Ricerca euristica *online*

- Nella ricerca online si conosce il valore della funzione euristica solo quando è stato 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. (Estendere HC con iniezioni di randomicità) Random-walk
   - si fanno mosse casuali in discesa

2. (Estendere HC con l'uso della memorizzazione) Apprendimento Real-Time:
   - esplorando si aggiustano i valori dell'euristica per renderli più realistici
   - Algoritmo **LRTA***
     - *Learning Real Time A**

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*

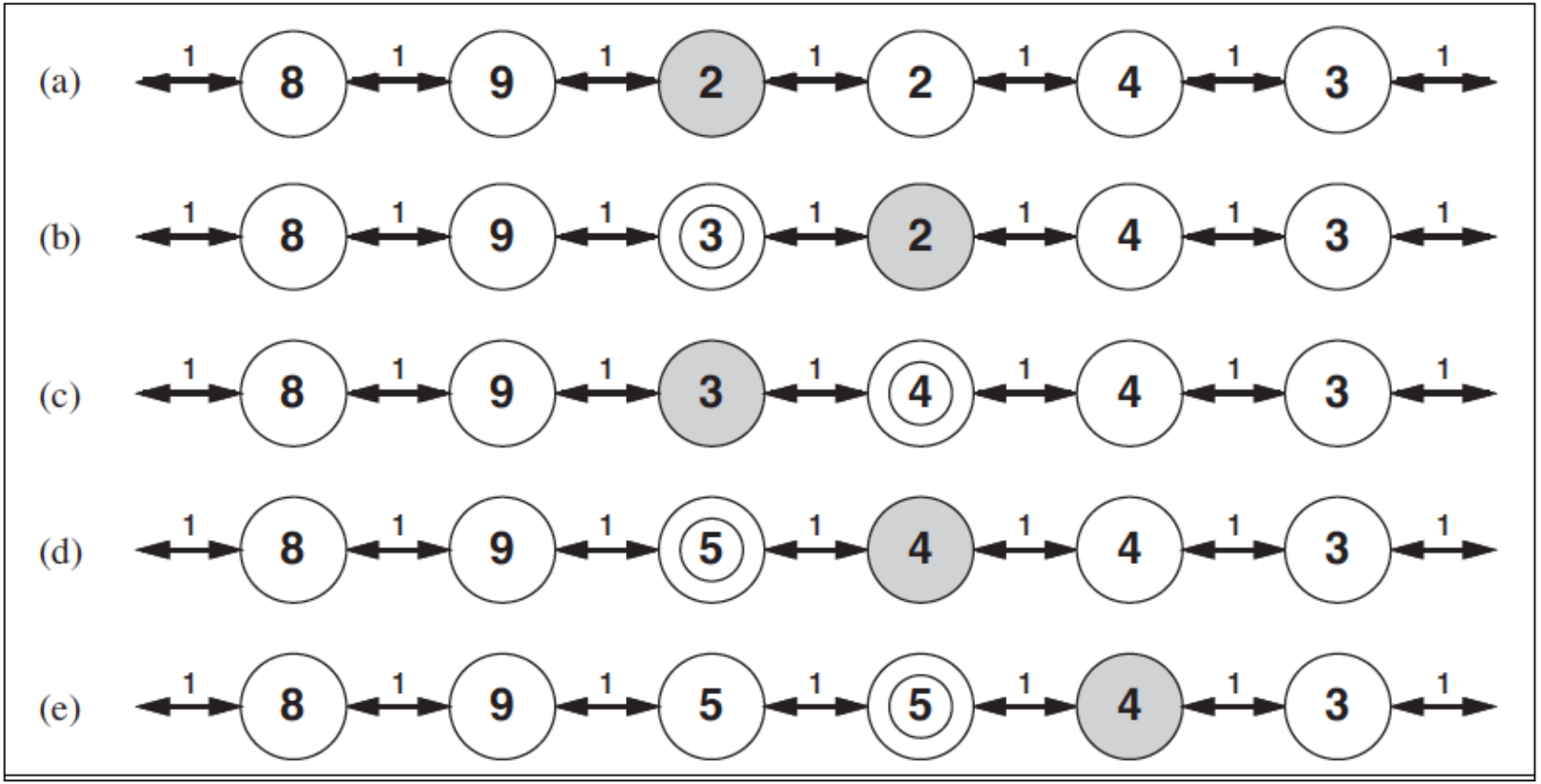
function LRTA*-AGENT(s') **returns** an action
  inputs: s', a percept that identifies the current state
  persistent: $result$, a table, indexed by state and action, initially empty
    H, a table of cost estimates indexed by state, initially empty
    s, a, the previous state and action, initially null

  if GOAL-TEST(s') **then return** $stop$
  if s' is a new state (not in H) **then** $H[s'] \leftarrow h(s')$
  if s is not null
    $result[s, a] \leftarrow s'$
    $H[s] \leftarrow \min_{b \in \text{ACTIONS}(s)}$ LRTA*-COST($s, b, result[s, b], H$)
  $a \leftarrow$ an action b in ACTIONS(s') that minimizes LRTA*-COST($s', b, result[s', b], H$)
  $s \leftarrow s'$
  return a

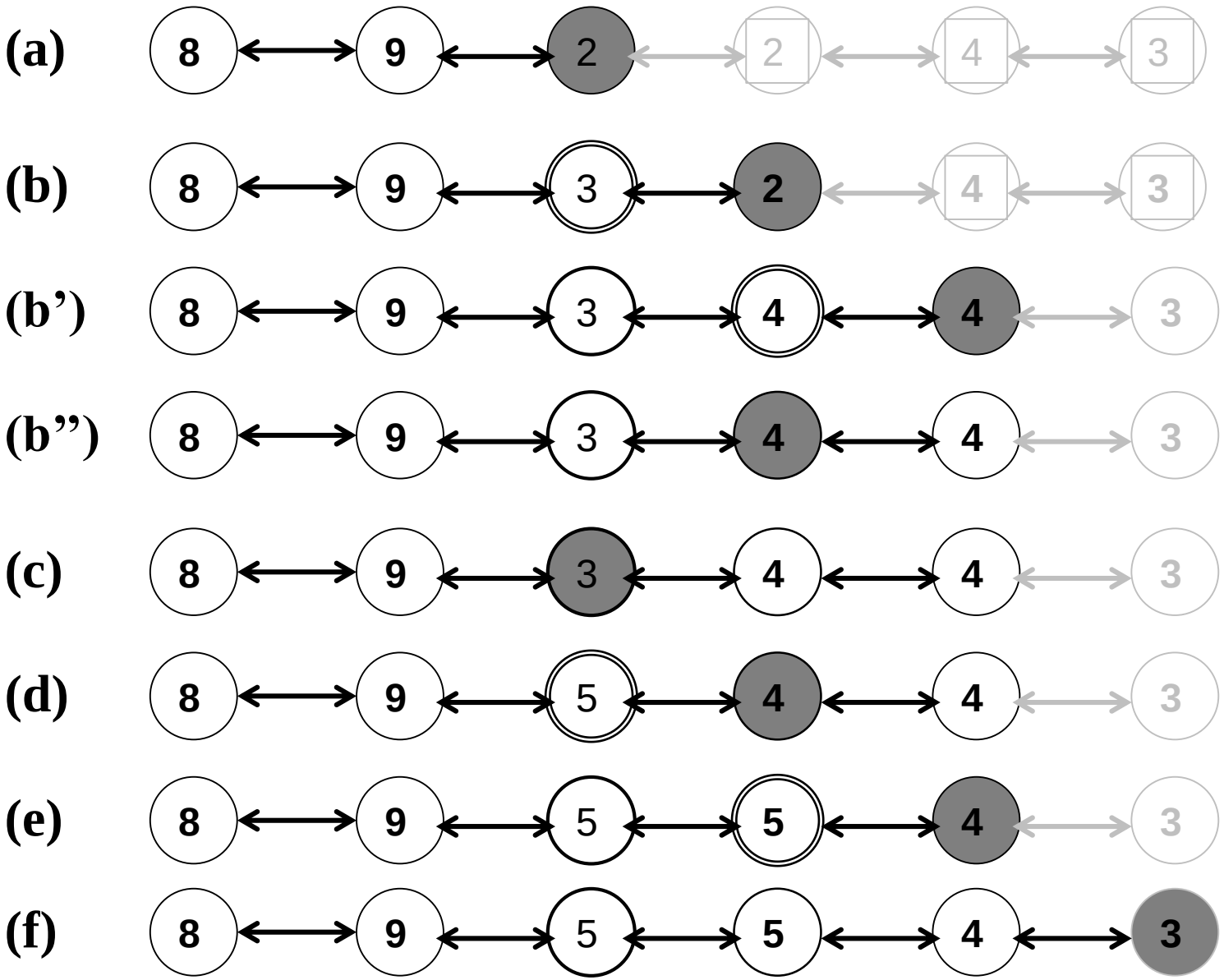
function LRTA*-COST(s, a, s', H) **returns** a cost estimate
  if s' is undefined **then return** $h(s)$
  else return $c(s, a, s') + H[s']$

Figure 4.24 LRTA*-AGENT selects an action according to the values of neighboring states, which are updated as the agent moves about the state space.

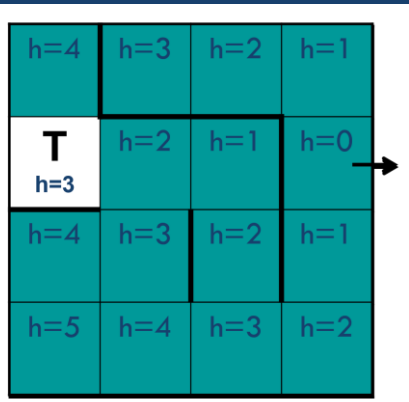
LRTA* supera i minimi locali

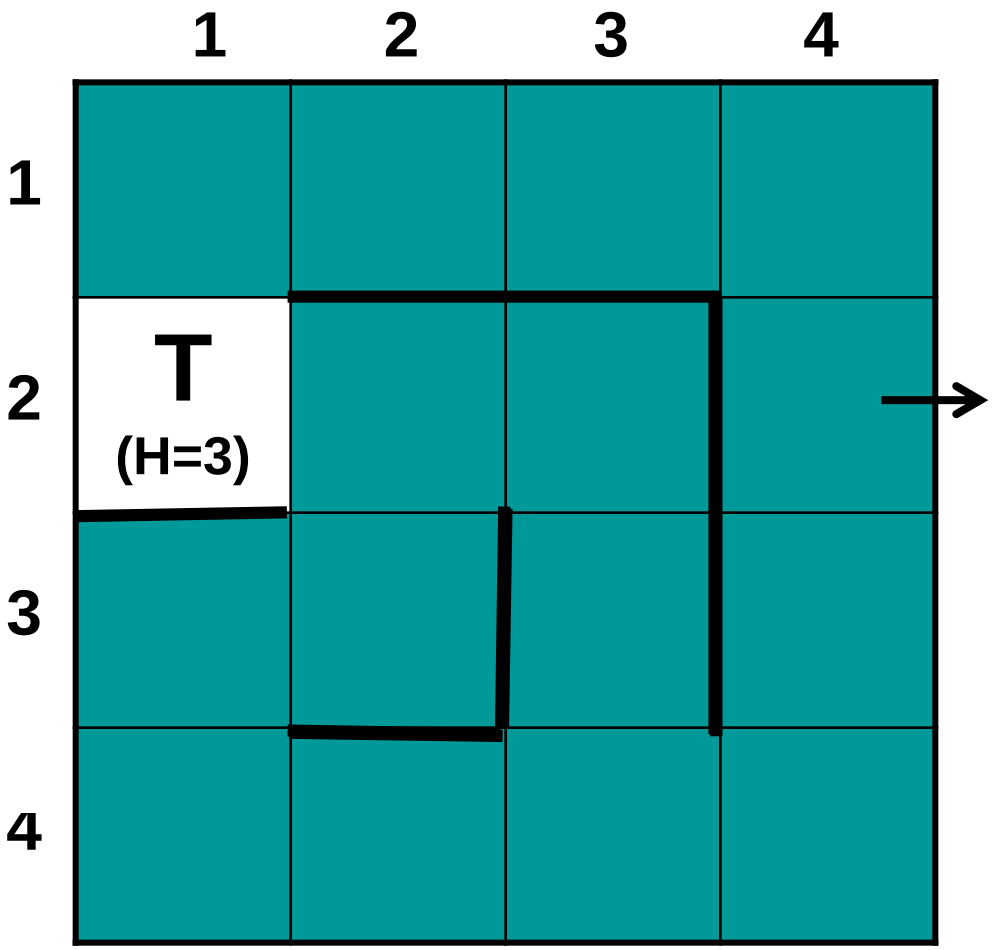


LRTA* supera i minimi locali (rev)

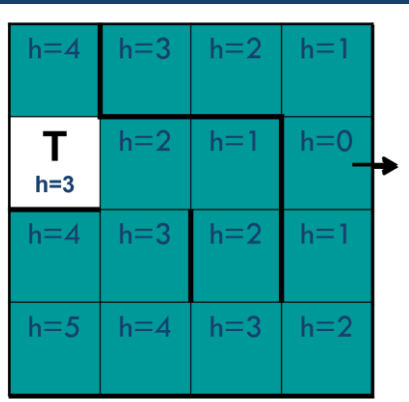


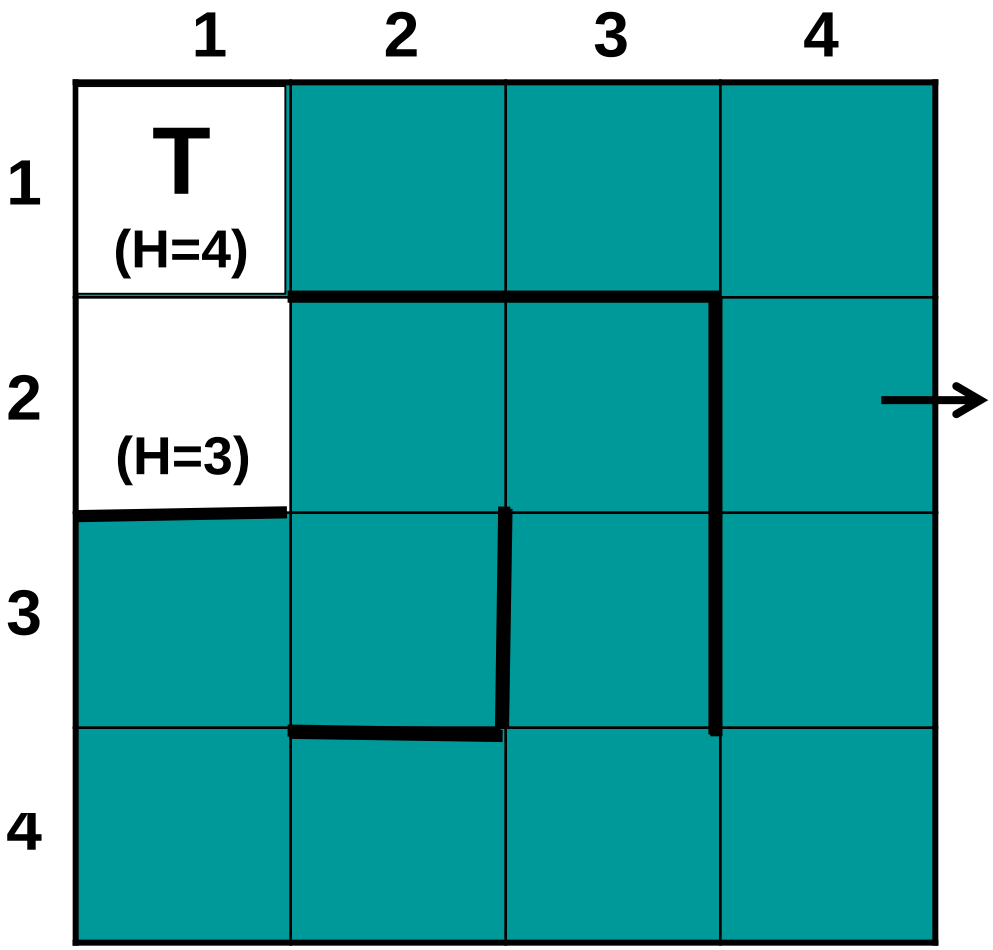
Esempio di LRTA*



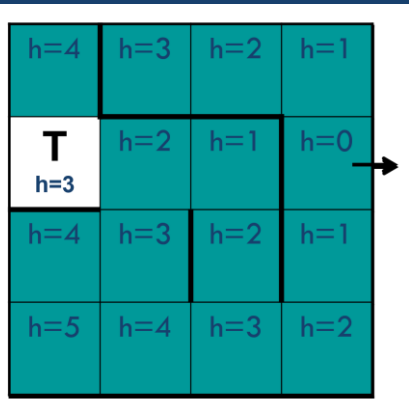


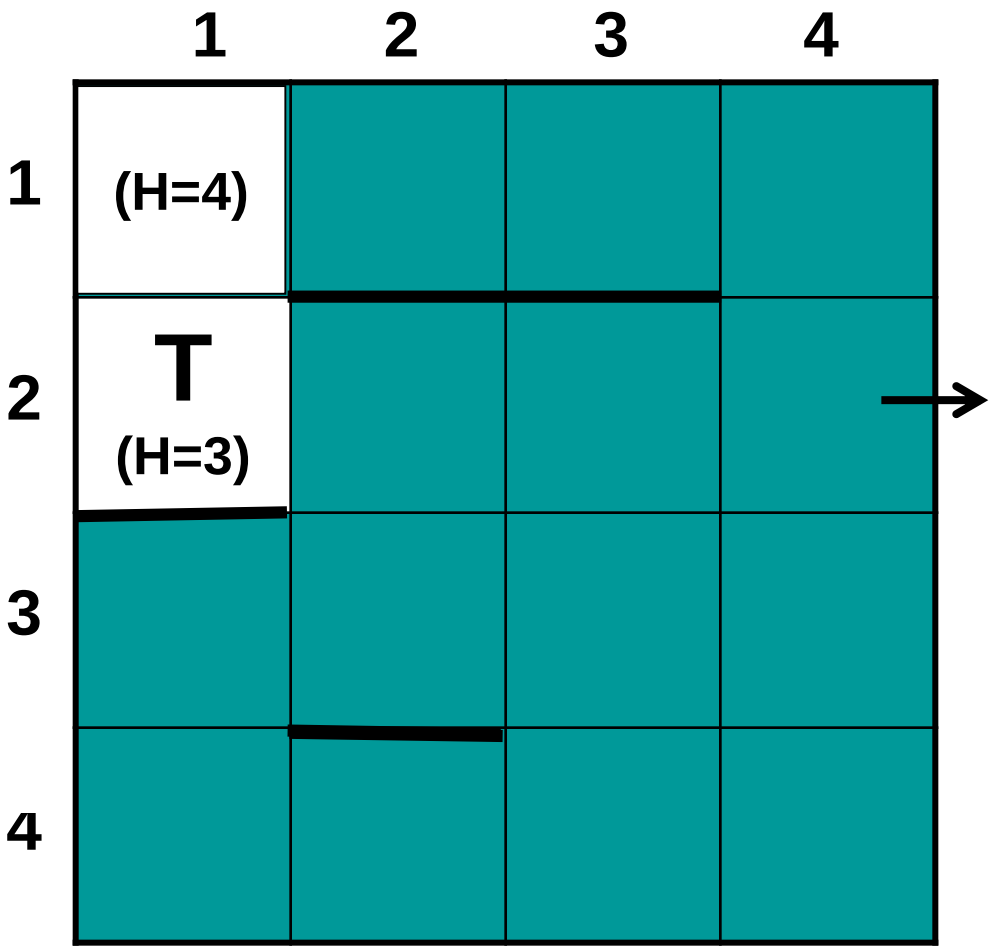
Esempio di LRTA*



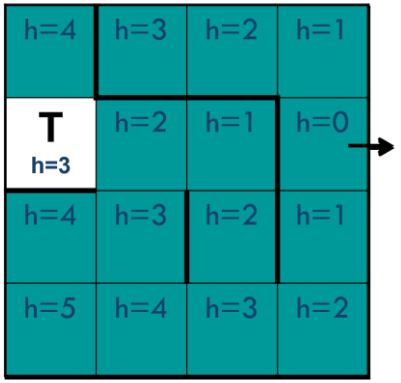


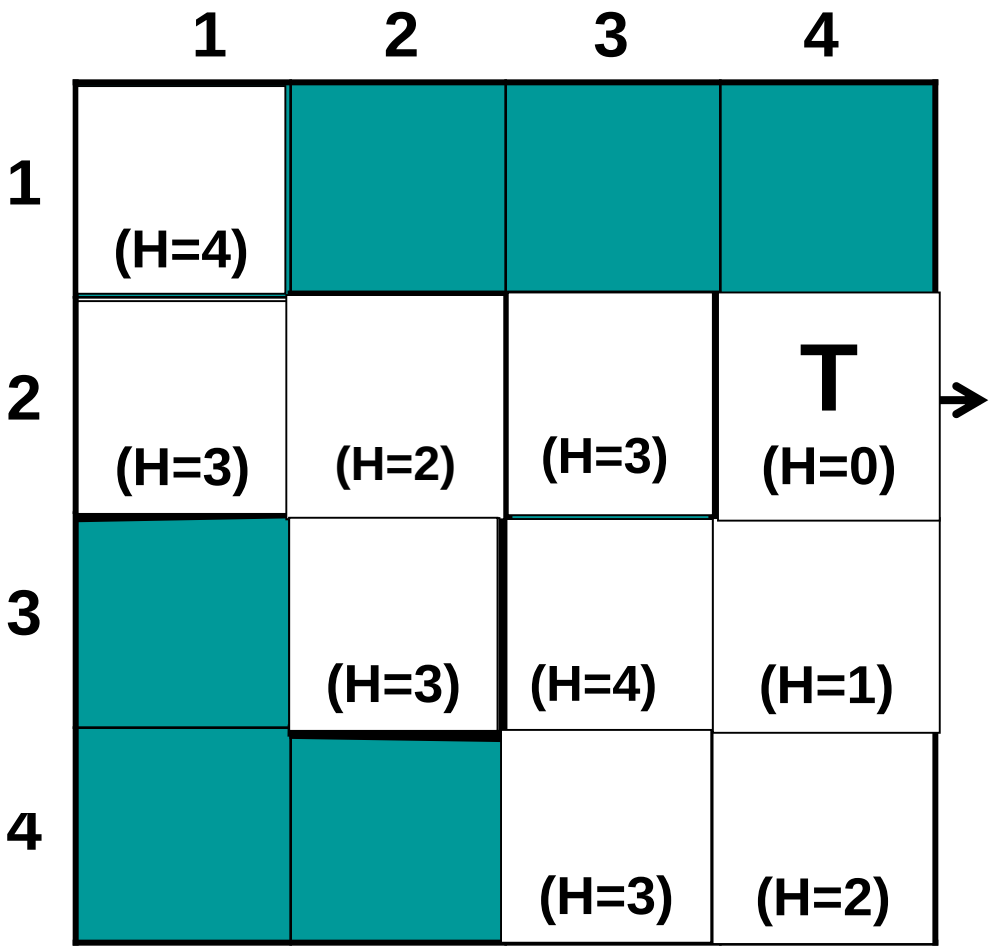
Esempio di LRTA*





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)

SummarAIzing

- Modelli avanzati di ricerca di soluzioni considerano ambienti osservabili parzialmente e non deterministici.
- Metodi di ricerca locali (*Local search) quali l'***hill climbing** operano su formulazioni complete dello stato ma mantengono solo un piccolo numero di nodi in memoria. Sono stati proposti diversi algoritmi stocastici, tra cui il **simulated annealing**: questo restituisce la soluzione ottima attraverso l'iniezione di livelli decrescenti di casualità, mediante l'uso di un appropriato modello di raffreddamento (temperature decrescenti nel tempo)
- Negli ambienti **nondeterministici**, gli agenti possono applicare metodi di **AND–OR search** per la generazione di piani **contingenti** in grado di raggiungere l'obbiettivo indipendentemente dagi avvenimenti che sorgono durante la esecuzione. Se l'ambiente è parzialmente osservabile, viene utilizzato un **spazio delle credenze** (**belief state)** che rappresenta l'insieme dei possibili stati in cui può trovarsi l'agente.
- **Nei problemi di esplorazione** l'agente non ha idea degli stati (e azioni) del suo ambiente. Per esplorazioni sicure, agenti dotati di capacità di **on-line search** possono costruire una mappa e trovare una soluzione quando questa esiste. In tali casi è utile poter aggiornare le funzioni euristiche a partire dalla esperienza, metodo utile a sfuggire ai minimi locali.