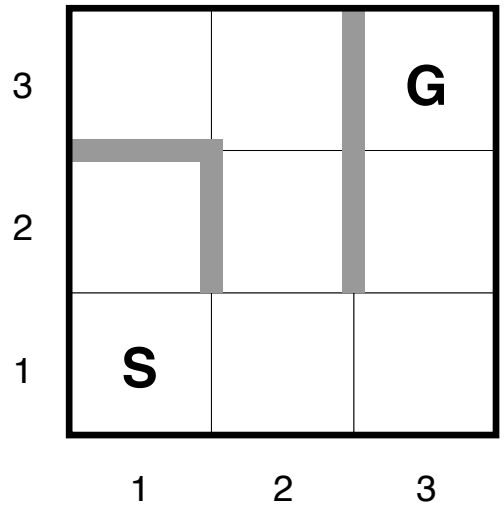
Ricerca online

Quando il problema non è totalmente osservabile o è dinamico (non-stazionario), bisogna interagire con l'ambiente per estrarre informazione:

- non è possibile pianificare a tavolino tutte le possibili azioni
- bisogna che l'agente alterni percezione con azione

In questo caso si parla di ricerca online

La ricerca online è particolarmente adatta a problemi di esplorazione



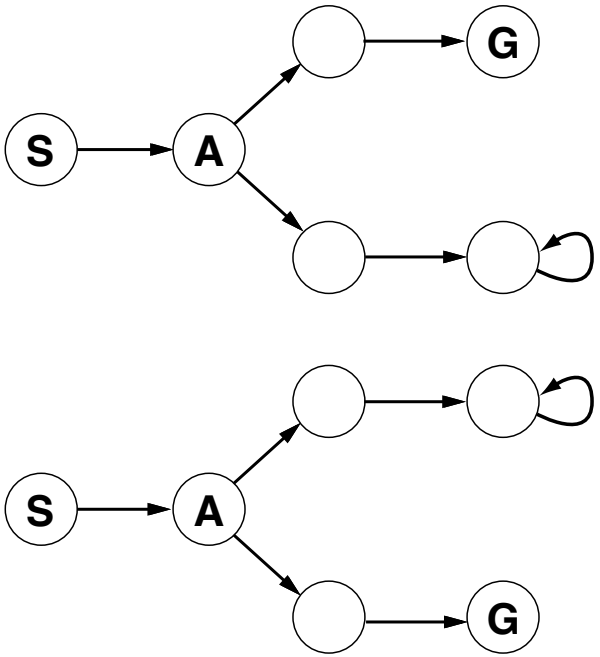
Si assume che l'agente conosca solo quanto segue:

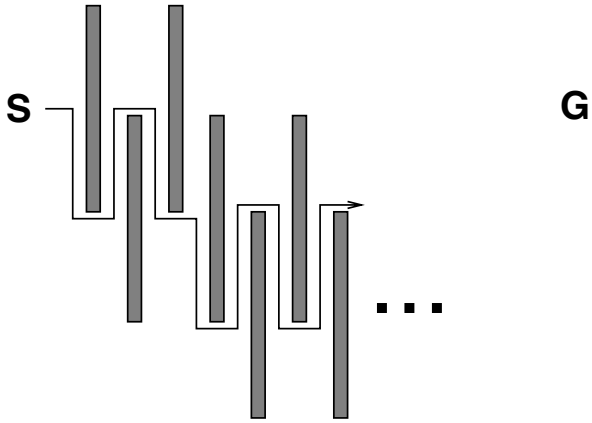
- AZIONI(s), che restituisce una lista di azioni permesse nello stato s
- il costo $c(s, a, s')$ della azione a per andare dallo stato s allo stato s'
- TEST-OBIETTIVO(s)

Problemi della ricerca online

Poiché l'agente ha informazione parziale, rischia di finire in vicoli ciechi

In generale, non è possibile evitare questo problema





Essendo la ricerca online inerentemente locale, si può usare una ricerca DFS

Ricerca in profondità online

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

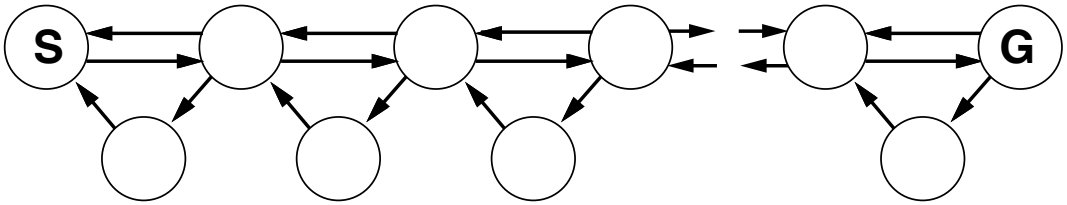
  if GOAL-TEST(s') **then return** *stop*
  if s' is a new state **then** *unexplored*[s'] $\leftarrow$ ACTIONS(s')
  if s is not null **then do**
    result[a, s] $\leftarrow s'$
    add s to the front of *unbacktracked*[s']
  if *unexplored*[s'] is empty **then**
    if *unbacktracked*[s'] is empty **then return** *stop*
    else $a \leftarrow$ an action b such that *result*[b, s'] = POP(*unbacktracked*[s'])
  else $a \leftarrow$ POP(*unexplored*[s'])
  $s \leftarrow s'$
  return a

Ricerca casuale

Una alternativa è data da una ricerca che opera una scelta casuale sulle azioni da intraprendere (random walk):

- si possono favorire le azioni non utilizzate fino a quel momento
- si può dimostrare che prima o poi la ricerca ha successo (in spazi finiti)

La ricerca però può essere MOLTO lenta, come nel seguente caso



dove la probabilità di tornare indietro è doppia rispetto a quella di procedere verso il goal

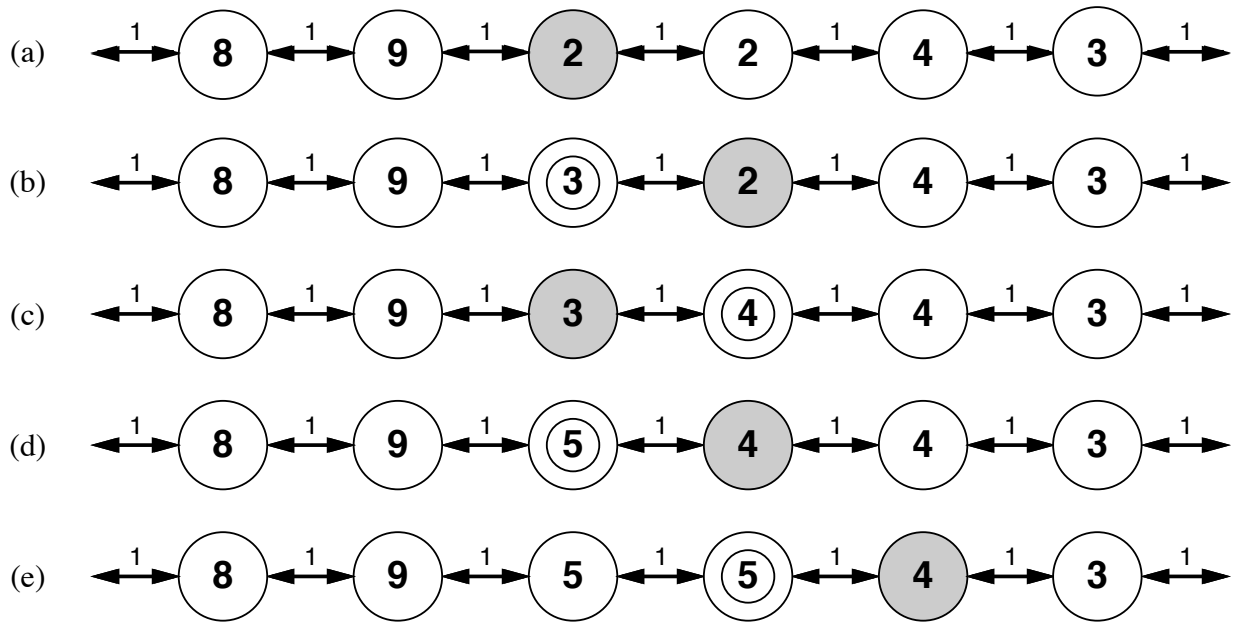
Si può fare di meglio se si utilizza Hill-climbing con memoria e fornita di una strategia per superare gli ottimi locali: Learning Real-Time A* (LRTA*)

Ricerca LRTA*

L'idea di base consiste:

– nel memorizzare la miglior stima corrente $H(s)$ del costo per raggiungere il goal da ogni stato visitato

– $H(s)$ inizialmente coincide con $h(s)$, ma successivamente è aggiornata con l'esperienza



Ricerca LRTA*

function LRTA*-AGENT(s') **returns** an action
  inputs: s', a percept that identifies the current state
  static: *result*, a table, indexed by action and state, 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** $H[s'] \leftarrow h(s')$
  unless s is null
    $result[a, s] \leftarrow s'$
    $H[s] \leftarrow \min_{b \in \text{ACTIONS}(s)} \text{LRTA*-COST}(s, b, result[b, s], H)$
  $a \leftarrow$ an action b in ACTIONS(s') minimizing LRTA*-COST($s', b, result[b, s'], H$)
  $s \leftarrow s'$
  return a

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