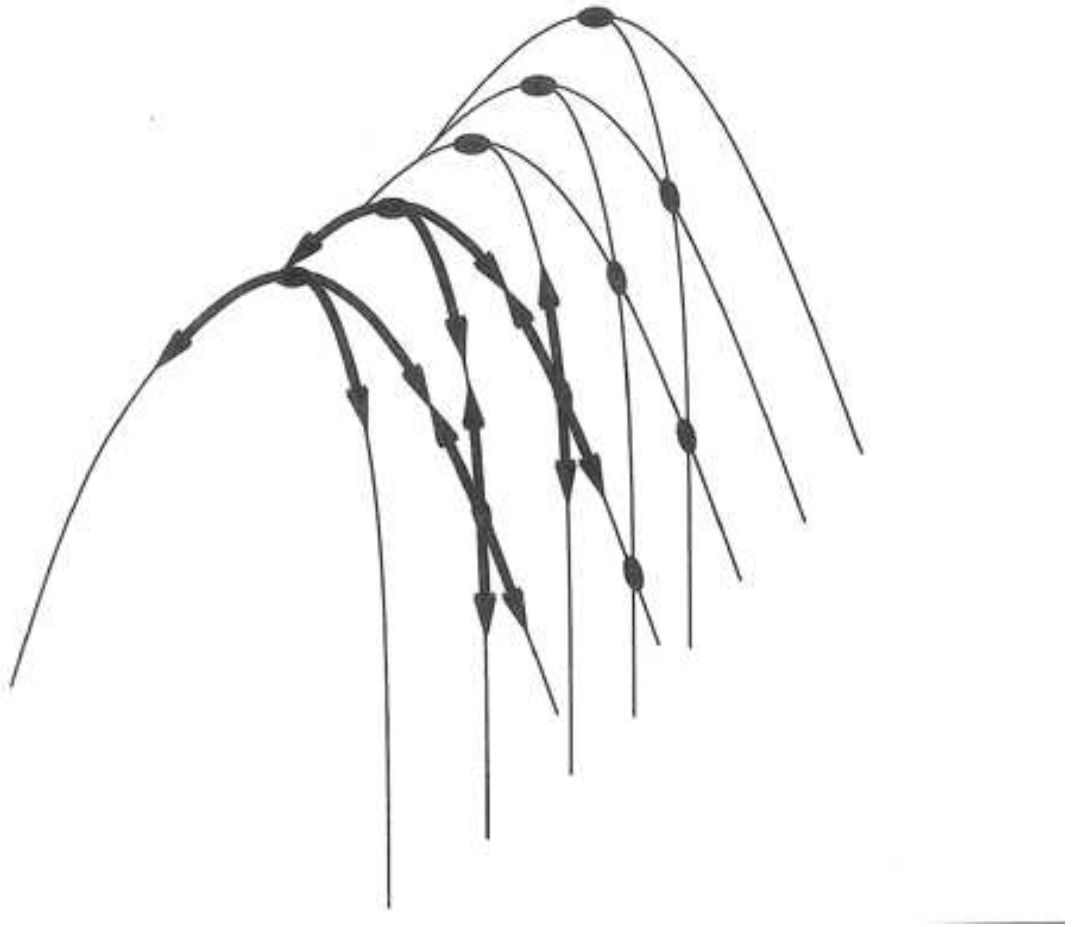
Hill-climbing: cresta (*ridge*)

Sequenza di massimi locali (a valore crescente muovendosi verso l'"interno" della figura) molto difficili da esplorare per Hill-climbing



Hill-climbing: alcune soluzioni

Possibili soluzioni:

- plateau (h piatta): "mossa laterale", cioè ci si sposta in uno stato con identico valore di h
  - bisogna stare attenti ad evitare cicli, specialmente nel caso di massimi (minimi) locali piatti
  - tipica soluzione: porre un limite massimo al numero consecutivo di mosse laterali
- massimi (minimi) locali: eseguire scelte stocastiche e/o più ricerche da stati iniziali diversi
  - Hill-climbing stocastico: scegliere a caso fra tutte le mosse che migliorano h, eventualmente usando una probabilità di selezione che è proporzionale al miglioramento (convergenza + lenta, ma spesso soluzioni migliori)
  - Hill-climbing con riavvio casuale: esegue più ricerche a partire da stati iniziali diversi (scelti a caso). Se p è la probabilità di trovare una soluzione ottima per una singola ricerca, il numero di ricerche atteso prima di trovare una soluzione ottima è $1/p$

Numero ricerche prima di trovare una soluzione

Perché il numero di ricerche atteso prima di trovare una soluzione ottima è $1/p$??

Numero ricerche prima di trovare una soluzione

Perché il numero di ricerche atteso prima di trovare una soluzione ottima è $1/p$??

Consideriamo le variabili aleatorie binarie $\mathbf{x}_i$ tali che

$$\mathbf{x}_i = \begin{cases} 0 & \text{se la } i\text{-esima ricerca non trova una soluzione ottima} \\ 1 & \text{se la } i\text{-esima ricerca trova una soluzione ottima} \end{cases}$$

Sappiamo che $\forall i$, $P(\mathbf{x}_i = 1) = p$, e $P(\mathbf{x}_i = 0) = 1 - p$

Inoltre le variabili $\mathbf{x}_i$ sono mutuamente indipendenti:
il risultato di una ricerca non influenza il risultato di un'altra ricerca

Abbiamo quindi un Processo di Bernoulli per cui:
il numero di ricerche prima di trovare una soluzione ottima rispetta una distribuzione Geometrica

Numero ricerche prima di trovare una soluzione

Infatti, la probabilità che la k-esima ricerca (su k) sia la prima a trovare una soluzione ottima è

$$P((\sum_{j=1}^{k-1} \mathbf{x}_j = 0) \wedge (\mathbf{x}_k = 1)) = (1-p)^{k-1}p$$

Il valore atteso della quantità che a noi interessa è quindi

$$\begin{aligned}
\sum_{k=1}^{\infty} k(1-p)^{k-1}p &= p\sum_{k=1}^{\infty} k(1-p)^{k-1} \\
&= p\sum_{k=1}^{\infty} \frac{d(1-p)^k}{dp} \\
&= -p\frac{d}{dp}\left(\sum_{k=0}^{\infty}(1-p)^k - 1\right) = -p\frac{d}{dp}\left(\frac{1}{p} - 1\right) \\
&= p\frac{1}{p^2} = \frac{1}{p}
\end{aligned}$$

da cui deriva il risultato

Hill-climbing e le 8 regine

Numero stati: 8^8 (circa 17 milioni)

- Hill-climbing standard
  - soluzione (ottima) trovata il 14% delle volte
  - in media circa 4 passi per trovare una soluzione, altrimenti circa 3 passi in caso di soluzione subottima
- Hill-climbing con mosse lateriali (non più di 100 consecutive)
  - soluzione (ottima) trovata il 94% delle volte
  - in media circa 21 passi per trovare una soluzione, altrimenti circa 64 passi in caso di soluzione subottima
- Hill-climbing con riavvio casuale
  - soluzione (ottima) trovata con probabilità $p = 0,14$, quindi circa 7 ricerche per trovare una soluzione ottima ($(1-p)/p = 6,14$ fallimenti $+ 1$ successo). Numero di passi complessivo atteso: $3(1-p)/p + 4 = 22,43$
  - con mosse laterali: soluzione (ottima) trovata con probabilità $p = 0,94$, quindi circa $1,06$ ricerche per trovare una soluzione ottima ($(1-p)/p = 0,06$ fallimenti $+ 1$ successo). Numero di passi complessivo atteso: $0,06(1-p)/p + 21 = 25,08$

Simulated annealing

Idea: evitare i massimi locali permettendo delle mosse "cattive"
ma gradualmente decrementare la loro grandezza e frequenza

```
function SIMULATED-ANNEALING(problema, raffreddamento) returns uno stato soluzione
    inputs: problema, un problema
            velocità_raffreddamento, una corrispondenza dal tempo alla "temperatura"
    variabili locali: nodo_corrente, un nodo
                      successivo, un nodo
                      T, una "temperatura" che controlla la probabilità
                           di compiere passi verso il basso

    nodo_corrente ← CREA-NODO(STATO-INIZIALE[problema])
    for t ← 1 to ∞ do
        T ← velocità_raffreddamento[t]
        if T = 0 then return nodo_corrente
        successivo ← un successore scelto a caso di nodo_corrente
        ΔE ← VALORE[successivo] – VALORE[nodo_corrente]
        if ΔE > 0 then nodo_corrente ← successivo
        else nodo_corrente ← successivo solo con probabilità e^(ΔE/T)
```

Proprietà del simulated annealing

A "temperatura" fissata T, la probabilità di occupazione degli stati raggiunge la distribuzione di Boltzmann

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

T diminuito abbastanza lentamente $\Longrightarrow$ si raggiunge sempre lo stato migliore (Metropolis et al., 1953, per problemi di modellazione di processi fisici)

Ampiamente usato in applicazioni pratiche, come la progettazione di circuiti VLSI e la definizione degli orari dei voli delle linee aeree

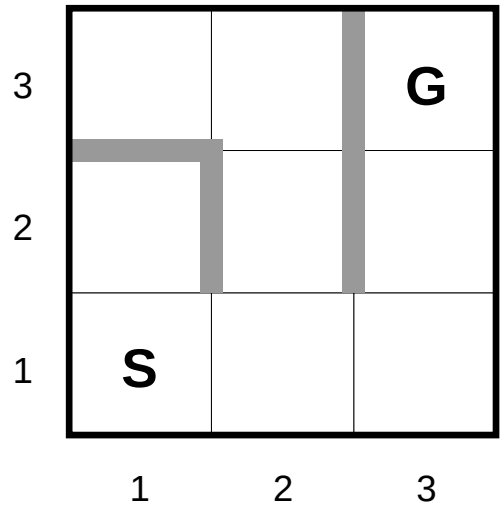
Ricerca online

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



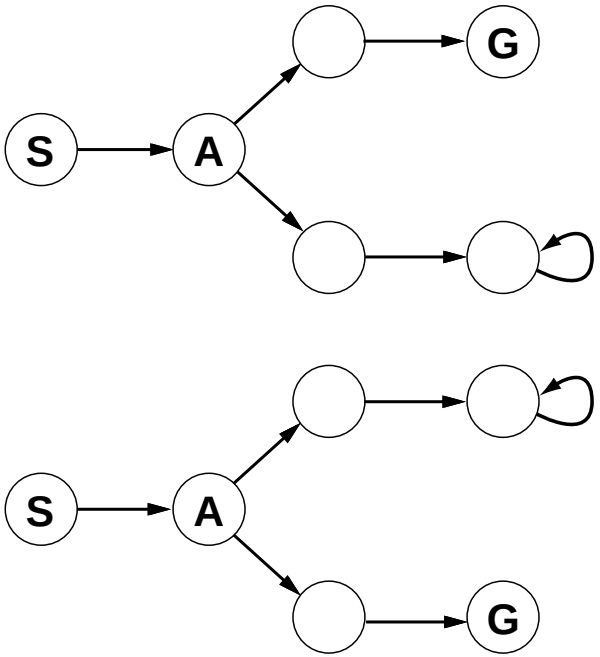
Si assume che l'agente conosca solo quanto segue:

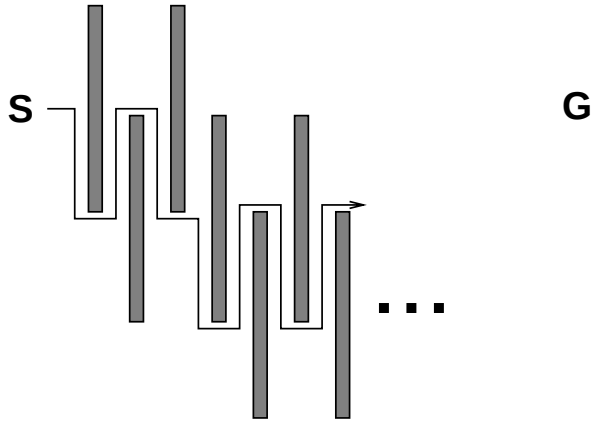
- AZIONI(s), che restituisce una lista di azioni permesse nello stato s
- il costo $c(s, a, s')$ della azione a per andare dallo stato s allo stato s'
- TEST-OBIETTIVO(s)

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

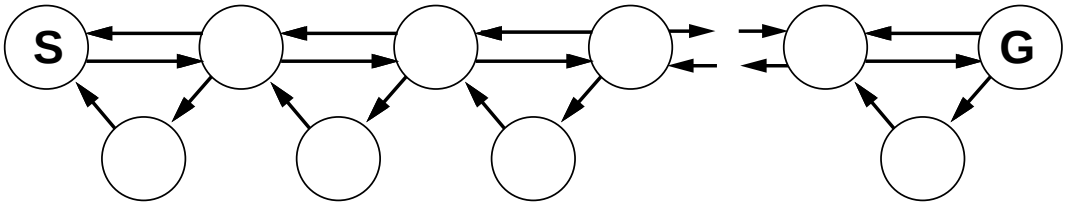
  if GOAL-TEST(s') **then return** *stop*
  if s' is a new state **then** *unexplored*[s'] $\leftarrow$ ACTIONS(s')
  if s is not null **then do**
    result[a, s] $\leftarrow s'$
    add s to the front of *unbacktracked*[s']
  if *unexplored*[s'] is empty **then**
    if *unbacktracked*[s'] is empty **then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

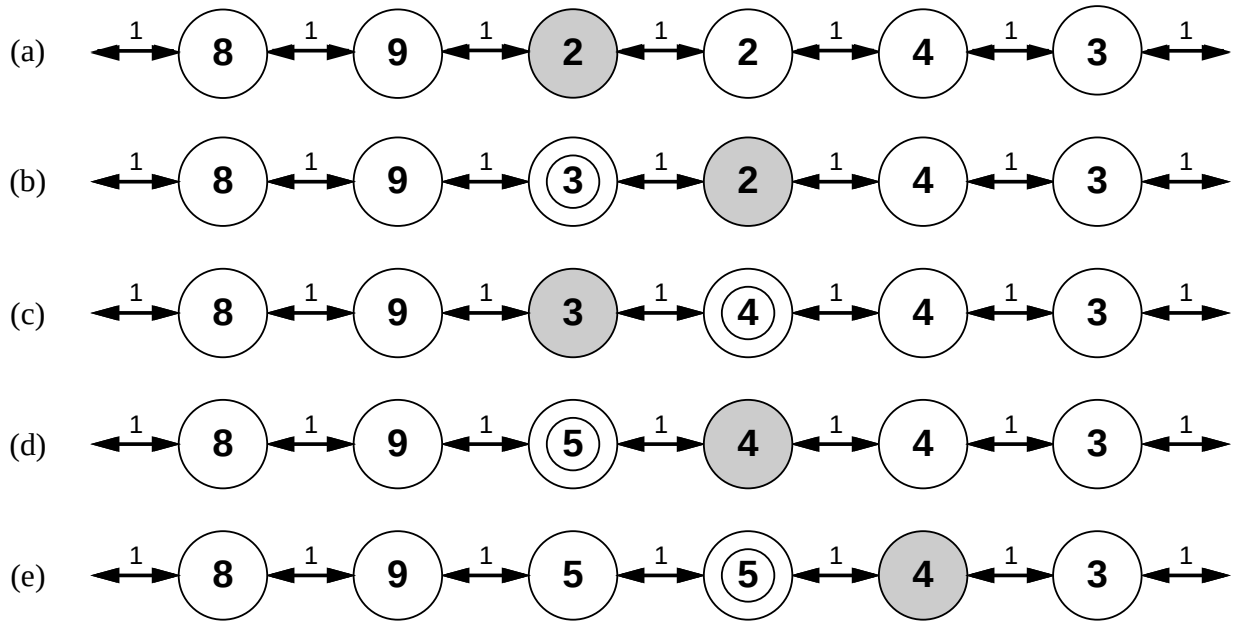
Si può fare di meglio se si utilizza Hill-climbing con memoria e fornita di una strategia per superare gli ottimi locali: Learning Real-Time A* (LRTA*)

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 $c(s, a, s') + H[s']$